Amazon EMR on EKS 開発ガイド

Amazon EMR

Amazon EMR: Amazon EMR on EKS 開発ガイド

Table of Contents

EKS での Amazon EMR とは

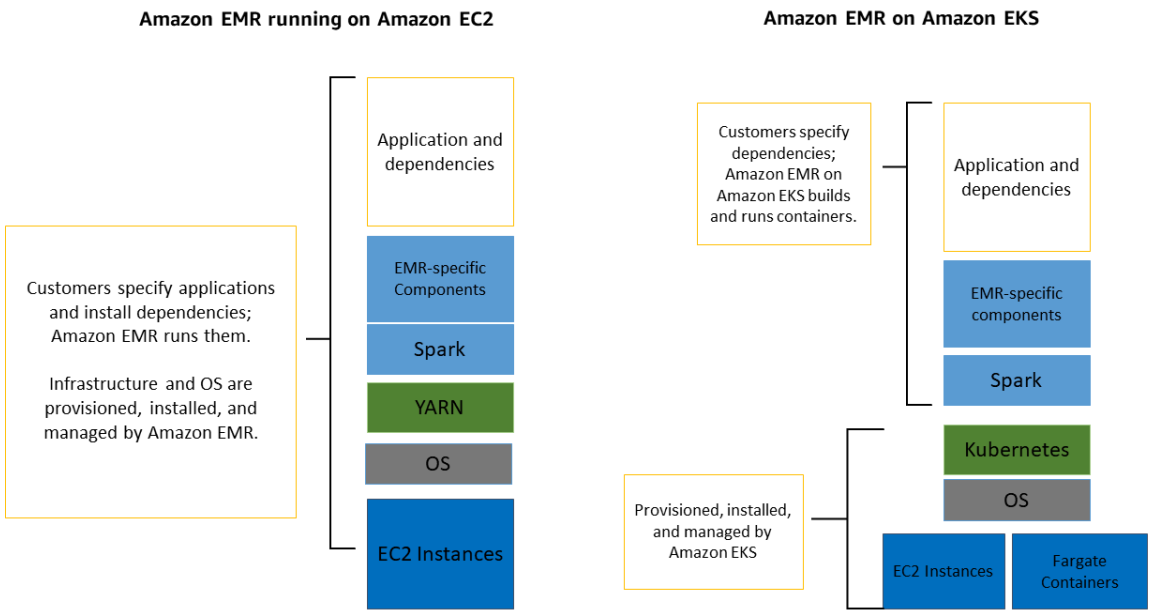
Amazon EMR on EKS には、Amazon Elastic Kubernetes Service (Amazon EKS) でオープンソースのビッグデータフレームワークを実行できる Amazon EMR のデプロイオプションが用意されています。Amazon EMR on EKS ではオープンソースアプリケーションのコンテナが自動的に構築、設定、管理されるため、ユーザーは分析ワークロードの実行に集中できます。

Amazon EMR を既に使用しているユーザーは、Amazon EMR ベースのアプリケーションを他のタイプのアプリケーションと共に同じ Amazon EKS クラスター上で実行できるようになります。さらに、このデプロイオプションを使用するとリソース使用率が向上し、複数のアベイラビリティーゾーンにわたるインフラストラクチャ管理が簡素化されます。Amazon EKS でビッグデータフレームワークを既に実行している場合は、Amazon EMR を使用してプロビジョニングと管理を自動化し、Apache Spark をより迅速に実行できるようになります。

Amazon EMR on EKS を使用すると、チームはより効率的にコラボレーションでき、大量のデータをより簡単かつ費用対効果の高い方法で処理できます。

- インフラストラクチャをプロビジョニングしなくても、共通のリソースプールでアプリケーションを実行できます。Amazon EMR Studio と AWS SDK または AWS CLI を使用して、EKS クラスターで実行されている分析アプリケーションを開発、送信、診断できます。Amazon EMR on EKS でスケジュールされたジョブは、セルフマネージドの Apache Airflow またはAmazon Managed Workflows for Apache Airflow (MWAA) を使用して実行できます。
- インフラストラクチャチームは、共通のコンピューティングプラットフォームを一元管理して、Amazon EMR ワークロードを他のコンテナベースのアプリケーションと統合できます。一般的な Amazon EKS ツールを使用してインフラストラクチャ管理を簡素化し、さまざまなバージョンのオープンソースフレームワークを必要とするワークロードに共有クラスターを利用できます。さらに、Kubernetes クラスターの管理と OS のパッチ適用が自動化されるため、運用上のオーバーヘッドを削減できます。Amazon EC2 と を使用すると AWS Fargate、複数のコンピューティングリソースを有効にして、パフォーマンス、運用、または財務の要件を満たすことができます。

次の図は、Amazon EMR の 2 つの異なるデプロイモデルを示しています。



トピック

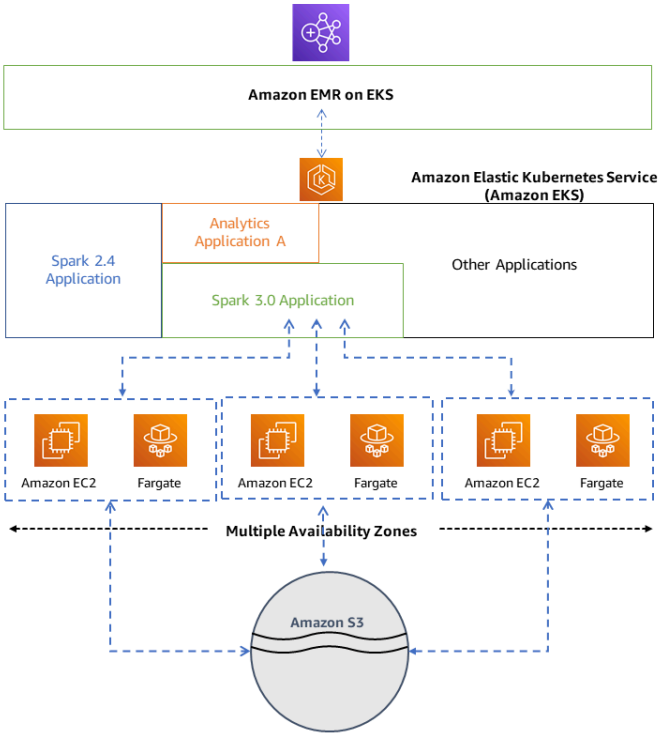
- Amazon EMR on EKS のアーキテクチャ
- Amazon EMR on EKS の概念と用語の理解
- Amazon EMR on EKS 仮想クラスターに作業を送信するとどうなるか

Amazon EMR on EKS のアーキテクチャ

Amazon EMR on EKS では、アプリケーションとそれが実行されるインフラストラクチャが疎結合されます。各インフラストラクチャレイヤーで、後続レイヤーのオーケストレーションが提供されます。ユーザーが Amazon EMR にジョブを送信する際のジョブ定義には、アプリケーション固有のパラメータがすべて含まれます。Amazon EMR はこれらのパラメータを使用して、デプロイするポッドとコンテナを Amazon EKS に指示します。次に、Amazon EKS は Amazon EC2 からコンピューティングリソースをオンラインにして、ジョブの実行に AWS Fargate 必要です。

このようなサービスの疎結合により、安全に分離された複数のジョブを同時に実行できます。また、異なるコンピューティングバックエンドで同じジョブをベンチマークしたり、複数のアベイラビリティーゾーンにジョブを分散して可用性を向上させたりすることもできます。

次の図は、Amazon EMR on EKS が他の AWS サービスとどのように連携するかを示しています。



Amazon EMR on EKS の概念と用語の理解

Amazon EMR on EKS には、Amazon Elastic Kubernetes Service (Amazon EKS) でオープンソースのビッグデータフレームワークを実行できる Amazon EMR のデプロイオプションが用意されています。このトピックでは、処理のために送信する作業単位である名前空間、仮想クラスター、ジョブ実行など、一般的な用語の一部に関するコンテキストについて説明します。

Kubernetes 名前空間

Amazon EKS では、クラスターリソースを複数のユーザーとアプリケーションに分割するために Kubernetes 名前空間が使用されます。これらの名前空間はマルチテナント環境の基盤です。Kubernetes 名前空間は、Amazon EC2 または をコンピューティングプロバイダー AWS Fargate として持つことができます。この柔軟性により、ジョブを実行する際のパフォーマンスやコストについてさまざまなオプションを選択できます。

仮想クラスター

仮想クラスターとは、Amazon EMR が登録されている Kubernetes 名前空間です。Amazon EMR では、仮想クラスターを使用してジョブを実行し、エンドポイントをホストします。複数の仮想クラスターを同じ物理クラスターでバックアップできますが、各仮想クラスターは EKS クラスター上の 1 つの名前空間にマッピングされます。仮想クラスターでは、請求に適用されるアクティブなリソースや、サービスの外部でライフサイクル管理を必要とするアクティブなリソースは作成されません。

ジョブ実行

ジョブ実行とは、Amazon EMR on EKS に送信する Spark jar、PySpark スクリプト、SparkSQL クエリなどの作業単位です。1 つのジョブに複数のジョブ実行を設定できます。ジョブ実行の送信時には、次の情報を含めます。

- ジョブを実行する仮想クラスター。
- ジョブを識別するジョブ名。
- 実行ロール。これはジョブを実行するスコープ付き IAM ロールで、ジョブからアクセスできるリソースを指定できます。
- Amazon EMR リリースラベル。使用するオープンソースアプリケーションのバージョンを指定します。
- ジョブの送信時に使用するアーティファクト (spark-submit パラメータなど)。

デフォルトでは、ログは Spark 履歴サーバーにアップロードされ、 AWS Management Consoleからアクセス可能です。イベントログ、実行ログ、メトリクスを、Amazon S3 と Amazon CloudWatch にプッシュすることもできます。

Amazon EMR コンテナ

Amazon EMR コンテナとは、Amazon EMR on EKS の API 名です。次のシナリオで `emr-containers` プレフィックスを使用します。

- Amazon EMR on EKS の CLI コマンドのプレフィックスです。例えば、`aws emr-containers start-job-run` と指定します。
- Amazon EMR on EKS の IAM ポリシーアクションの前に使用するプレフィックスです。例えば、`"Action": [ "emr-containers:StartJobRun"]` と指定します。詳細については、Amazon EMR on EKS でのポリシーアクションを参照してください。

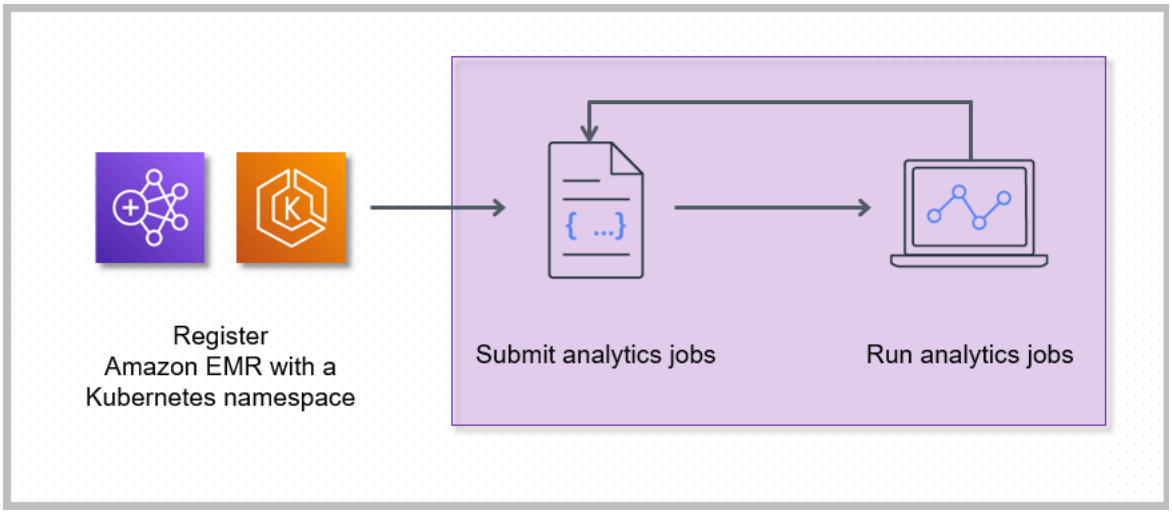
- Amazon EMR on EKS サービスエンドポイントで使用するプレフィックスです。例えば、`emr-containers.us-east-1.amazonaws.com` と指定します。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

Amazon EMR on EKS 仮想クラスターに作業を送信するとどうなるか

Amazon EMR を Amazon EKS 上の Kubernetes 名前空間に登録すると、仮想クラスターが作成されます。これで、Amazon EMR はその名前空間で分析ワークロードを実行できるようになります。ユーザーが Amazon EMR on EKS を使って Spark ジョブを仮想クラスターに送信すると、Amazon EMR on EKS は Amazon EKS 上の Kubernetes スケジューラにポッドのスケジュールをリクエストします。

次の手順と図は、Amazon EMR on EKS のワークフローを示しています。

- 既存の Amazon EKS クラスターを使用するか、eksctl コマンドラインユーティリティまたは Amazon EKS コンソールを使用してクラスターを作成します。
- Amazon EMR を EKS クラスター上の名前空間に登録して、仮想クラスターを作成します。
- AWS CLI または SDK を使用して、仮想クラスターにジョブを送信します。



Amazon EMR on EKS は、ユーザーが実行するジョブごとに、Amazon Linux 2 ベースイメージ、Apache Spark、および関連する依存関係を持つコンテナを作成します。各ジョブはポッドで実行され、コンテナがダウンロードされて実行が開始されます。ポッドはジョブの終了後に終了します。コンテナのイメージが既にノードにデプロイされている場合は、キャッシュされたイメージが使

用されてダウンロードがバイパスされます。ログフォワーダーやメトリクスフォワーダーなどのサイドカーコンテナをポッドにデプロイできます。ジョブが終了した後でも、Amazon EMR コンソールの Spark アプリケーション UI を使用してジョブをデバッグできます。

Amazon EMR on EKS のご利用開始にあたって

このトピックを参照し、仮想クラスターに Spark アプリケーションをデプロイすることで、EKS での Amazon EMR を使い始めるのに役立ちます。これには、正しいアクセス許可を設定し、ジョブを開始する手順が含まれています。開始する前に、「Amazon EMR on EKS のセットアップ」の手順を完了しておくようにしてください。これにより、仮想クラスターを作成する前に、 AWS CLI セットアップなどのツールを取得できます。開始時に役立つ他のテンプレートについては、GitHub の「EMR Containers Best Practices Guide」を参照してください。

セットアップ手順の次の情報が必要になります。

- Amazon EMR に登録された Amazon EKS クラスターおよび Kubernetes 名前空間の仮想クラスター ID

> **Important**
>
> EKS クラスターを作成する際は、m5.xlarge をインスタンスタイプとして使用するか、CPU とメモリがそれよりも高いその他のインスタンスタイプを使用してください。CPU またはメモリが m5.xlarge よりも低いインスタンスタイプを使用すると、クラスターで使用可能なリソースが不足することによるジョブの失敗につながる可能性があります。

- ジョブの実行に使用する IAM ロールの名前
- Amazon EMR リリースのリリースラベル (たとえば、`emr-6.4.0-latest` など)
- ロギングおよびモニタリングの送信先ターゲット:
  - Amazon CloudWatch ロググループ名とログストリームのプレフィックス
  - イベントログとコンテナログを保存する Amazon S3 の場所

> **Important**
>
> Amazon EMR on EKS ジョブでは、モニタリングとログの送信先ターゲットとして Amazon CloudWatch と Amazon S3 を使用します。これらの送信先に送信されるジョブログを表示することで、ジョブの進行状況をモニタリングし、エラーのトラブルシューティングを行うことができます。ログを有効にするには、ジョブの実行の IAM ロールに関連付けられた IAM ポリシーに、ターゲットリソースへのアクセスに必要なアクセス許可が必要です。必要なアクセス許可が IAM ポリシーにない場合は、このサンプルジョブを実行する前に、ジョブ実

行ロールの信頼ポリシーを更新する、Configure a job run to use Amazon S3 logs、および Configure a job run to use CloudWatch Logs のステップに従う必要があります。

Spark アプリケーションの実行

次のステップに従って、EKS での Amazon EMR でシンプルな Spark アプリケーションを実行します。Spark Python アプリケーションのアプリケーション `entryPoint` ファイルは、`s3://`*`REGION`*`.elasticmapreduce/emr-containers/samples/wordcount/scripts/wordcount.py` にあります。*`REGION`* は、*`us-east-1`* など、Amazon EMR on EKS 仮想クラスターが存在するリージョンです。

1. 次のポリシーステートメントで示すように、必要なアクセス許可があるジョブ実行ロールの IAM ポリシーを更新します。

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

```
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-destination-bucket/*",
                "arn:aws:s3:::amzn-s3-demo-logging-bucket/*"
            ]
        },
        {
            "Sid": "DescribeAndCreateCloudwatchLogStream",
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:DescribeLogGroups",
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:*:*:*"
            ]
        },
        {
            "Sid": "WriteToCloudwatchLogs",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:*:*:log-group:my_log_group_name:log-
stream:my_log_stream_prefix/*"
            ]
        }
    ]
}
```

- このポリシーの最初のステートメントである ReadFromLoggingAndInputScriptBuckets は、ListBucket および GetObjects に次の Amazon S3 バケットへのアクセスを許可します。
  - *REGION*.elasticmapreduce - アプリケーション entryPoint ファイルが配置されているバケット。
  - *amzn-s3-demo-destination-bucket* - 出力データ用に定義するバケット。
  - *amzn-s3-demo-logging-bucket* - ログ記録データ用に定義するバケット。

- このポリシーの 2 番目のステートメントである WriteToLoggingAndOutputDataBuckets は、出力バケットとログバケットにそれぞれデータを書き込むアクセス許可をジョブに付与します。
- 3 番目のステートメントである DescribeAndCreateCloudwatchLogStream は、Amazon CloudWatch Logs を記述して作成するアクセス許可をジョブに付与します。
- 4 番目のステートメントである WriteToCloudwatchLogs は、*my_log_stream_prefix* という名前のログストリームの下にある *my_log_group_name* という名前の Amazon CloudWatch ロググループにログを書き込むためのアクセス許可を付与します。

2. Spark Python アプリケーションを実行するには、次のコマンドを使用します。置き換え可能なすべての*##########*値を適切な値に置き換えます。*REGION* は、*us-east-1* など、Amazon EMR on EKS 仮想クラスターが存在するリージョンです。

```
aws emr-containers start-job-run \
--virtual-cluster-id cluster_id \
--name sample-job-name \
--execution-role-arn execution-role-arn \
--release-label emr-6.4.0-latest \
--job-driver '{
  "sparkSubmitJobDriver": {
    "entryPoint": "s3://REGION.elasticmapreduce/emr-containers/samples/wordcount/
scripts/wordcount.py",
    "entryPointArguments": ["s3://amzn-s3-demo-destination-bucket/
wordcount_output"],
    "sparkSubmitParameters": "--conf spark.executor.instances=2 --
conf spark.executor.memory=2G --conf spark.executor.cores=2 --conf
 spark.driver.cores=1"
  }
}' \
--configuration-overrides '{
  "monitoringConfiguration": {
    "cloudWatchMonitoringConfiguration": {
      "logGroupName": "my_log_group_name",
      "logStreamNamePrefix": "my_log_stream_prefix"
    },
    "s3MonitoringConfiguration": {
       "logUri": "s3://amzn-s3-demo-logging-bucket"
    }
  }
}'
```

このジョブからの出力データは、s3://*amzn-s3-demo-destination-bucket*/wordcount_output で使用できます。

ジョブ実行に指定されたパラメータを使用して、JSON ファイルを作成することもできます。次に、JSON ファイルへのパスを指定して start-job-run コマンドを実行します。詳細については、StartJobRun でジョブ実行を送信する を参照してください。ジョブ実行パラメータの設定について詳しくは、ジョブ実行を構成するためのオプション を参照してください。

3. Spark SQL アプリケーションを実行するには、次のコマンドを使用します。すべての*########*値を適切な値に置き換えます。*REGION* は、*us-east-1* など、Amazon EMR on EKS 仮想クラスターが存在するリージョンです。

```
aws emr-containers start-job-run \
--virtual-cluster-id cluster_id \
--name sample-job-name \
--execution-role-arn execution-role-arn \
--release-label emr-6.7.0-latest \
--job-driver '{
  "sparkSqlJobDriver": {
    "entryPoint": "s3://query-file.sql",
    "sparkSqlParameters": "--conf spark.executor.instances=2 --
conf spark.executor.memory=2G --conf spark.executor.cores=2 --conf
 spark.driver.cores=1"
  }
}' \
--configuration-overrides '{
  "monitoringConfiguration": {
    "cloudWatchMonitoringConfiguration": {
      "logGroupName": "my_log_group_name",
      "logStreamNamePrefix": "my_log_stream_prefix"
    },
    "s3MonitoringConfiguration": {
       "logUri": "s3://amzn-s3-demo-logging-bucket"
    }
  }
}'
```

SQL クエリファイルの例を以下に示します。テーブルのデータが保存される S3 などの外部ファイルストアが必要です。

```
CREATE DATABASE demo;
```

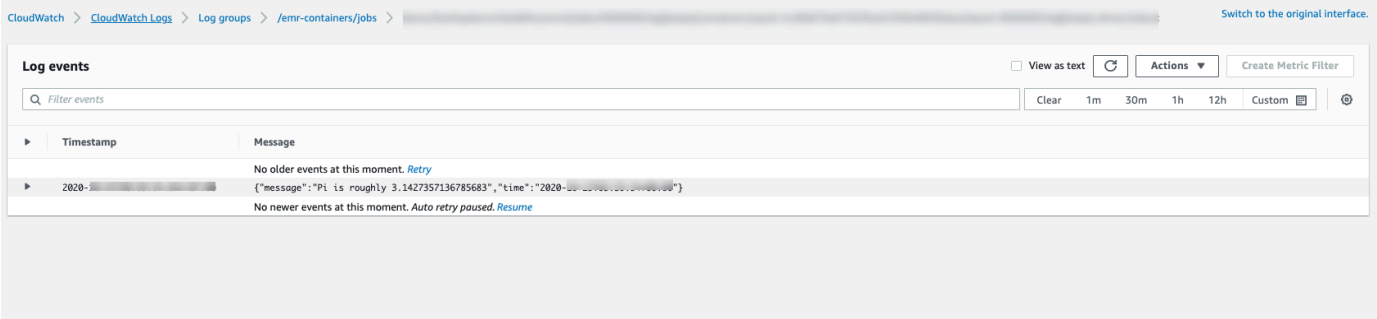
```
CREATE EXTERNAL TABLE IF NOT EXISTS demo.amazonreview( marketplace string,
 customer_id string, review_id  string, product_id  string, product_parent  string,
 product_title  string, star_rating  integer, helpful_votes  integer, total_votes
  integer, vine  string, verified_purchase  string, review_headline  string,
 review_body  string, review_date  date, year  integer) STORED AS PARQUET LOCATION
 's3://URI to parquet files';
SELECT count(*) FROM demo.amazonreview;
SELECT count(*) FROM demo.amazonreview WHERE star_rating = 3;
```

このジョブの出力は、設定されている monitoringConfiguration に応じて S3 または CloudWatch のドライバーの stdout ログに表示されます。

4. ジョブ実行に指定されたパラメータを使用して、JSON ファイルを作成することもできます。次に、JSON ファイルへのパスを指定して start-job-run コマンドを実行します。詳細については、「Submit a job run」を参照してください。ジョブ実行パラメータの設定について詳しくは、「Options for configuring a job run」を参照してください。

   ジョブの進行状況をモニタリングしたり、失敗をデバッグしたりするには、Amazon S3、CloudWatch Logs、またはその両方にアップロードされたログを検査します。Amazon S3 のログパスについては、Configure a job run to use S3 logs を参照してください。CloudWatch Logs については、Configure a job run to use CloudWatch Logs を参照してください。CloudWatch Logs でログを表示するには、以下の手順に従います。

   - CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
   - ナビゲーションペインで [ログ] を選択します。次に、[ロググループ] を選択します。
   - Amazon EMR on EKS のロググループを選択したら、アップロードされたログイベントを表示します。



> **Important**
>
> ジョブにはデフォルトで再試行ポリシーが設定されています。構成を変更または無効にする方法については、「Using job retry policies」を参照してください。

GitHub の「Amazon EMR on EKS Best Practices Guide」へのリンク

「Amazon EMR on EKS Best Practices Guide」は、オープンソースコミュニティとのコラボレーションにより作成されました。これにより、仮想クラスターの作成と実行について迅速に作業を進め、推奨事項を提示できます。これらのセクションについては、「Amazon EMR on EKS best practices guide」を使用することをお勧めします。各セクションのリンクを選択して GitHub サイトに移動します。

セキュリティ

> **Note**
>
> EKS での Amazon EMR のセキュリティの詳細については、「Amazon EMR on EKS でのセキュリティのベストプラクティス」を参照してください。

暗号化のベストプラクティス: 保管中のデータおよび転送中のデータに暗号化を使用する方法。

ネットワークセキュリティの管理では、Amazon RDS や Amazon Redshift のように AWS のサービス にホストされているデータソースに接続する際の EKS での Amazon EMR のポッドのセキュリティグループを設定する方法について説明します。

AWS Secrets Manager を使用してシークレットを保存します。

Pyspark ジョブ送信

Pyspark ジョブ送信: zip、egg、wheel、pex などのパッケージ形式を使用して、pySpark アプリケーションのさまざまなタイプのパッケージを指定します。

ストレージ

EBS ボリュームの使用: EBS ボリュームを必要とするジョブに静的プロビジョニングと動的プロビジョニングを使用する方法。

Amazon FSx for Lustre ボリュームの使用: Amazon FSx for Luster ボリュームを必要とするジョブに静的プロビジョニングと動的プロビジョニングを使用する方法。

インスタンスストアボリュームの使用: ジョブ処理にインスタンスストアボリュームを使用する方法。

メタストア統合

Hive メタストアの使用: Hive メタストアを使用するさまざまな方法を提供します。

Glue AWS の使用 : には、Glue AWS カタログを設定するさまざまな方法が用意されています。

デバッグ

Spark デバッグの使用: ログレベルの変更の方法。

ドライバーポッドの Spark UI への接続。

EKS での Amazon EMR でセルフホスト型の Spark 履歴サーバーを使用する方法。

EKS での Amazon EMR に関する問題のトラブルシューティング

トラブルシューティング。

ノードの配置

`single-az` およびその他のユースケースでの Kubernetes ノードセレクターの使用。

Fargate ノード配置の使用。

パフォーマンス

動的リソース割り当て (DRA) の使用。

Amazon VPC コンテナネットワークインターフェイスプラグイン (CNI)、Cluster Autoscaler、コア DNS の EKS ベストプラクティス。

コストの最適化

スポットインスタンスの使用: Amazon EC2 スポットインスタンスのベストプラクティスと Spark ノードの廃止機能の使用方法。

の使用 AWS Outposts

[を使用した Amazon EMR on EKS の実行 AWS Outposts]

Amazon EMR on EKS の Docker イメージのカスタマイズ

カスタマイズした Docker イメージは、Amazon EMR on EKS で使用できます。Amazon EMR on EKS ランタイムイメージをカスタマイズすると、次の利点を得られます。

- アプリケーションの依存関係とランタイム環境を単一のイミュータブルコンテナにパッケージ化し、移植性を高め、各ワークロードの依存関係管理を効率化します。
- 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。
- Amazon EMR on EKS を、ローカルでの開発やテストなど、組織内で現在確立されているビルド、テスト、デプロイの各プロセスと統合します。
- 組織内のコンプライアンスおよびガバナンス要件を満たす、確立されたセキュリティプロセス (イメージスキャンなど) を適用します。

トピック

Docker イメージをカスタマイズする方法

Amazon EMR on EKS の Docker イメージをカスタマイズするには、次の手順を行います。この手順では、ベースイメージを取得して、カスタマイズして公開し、イメージを使用することによってワークロードを送信する方法を示します。

> **Note**
>
> Docker イメージのカスタマイズ時に考慮することが必要な場合があるその他のオプションは、インタラクティブエンドポイント用にカスタマイズすることです。これは、必要な依存関係を確保する、またはマルチアーキテクチャコンテナイメージを使用するために行います。
>
> - インタラクティブエンドポイントの Docker イメージをカスタマイズする
> - マルチアーキテクチャイメージを使用する

前提条件

- Amazon EMR on EKS で Amazon EMR on EKS のセットアップ ステップを実行します。
- ご自分の環境に Docker をインストールします。詳細については、Get Docker を参照してください。

ステップ 1: Amazon Elastic Container Registry (Amazon ECR) からベースイメージを取得する

ベースイメージには、Amazon EMR ランタイムと、他の AWS サービスへのアクセスに使用されるコネクタが含まれています。Amazon EMR 6.9.0 以降の場合は、Amazon ECR Public Gallery からベースイメージを取得できます。ギャラリーを参照してイメージリンクを見つけ、そのイメージをローカルワークスペースに取り込みます。例えば、Amazon EMR 7.6.0 リリースの場合、次のdocker pullコマンドは最新の標準ベースイメージを取得します。emr-7.6.0:latest を emr-7.6.0-spark-rapids:latest に置き換えると、Nvidia RAPIDS アクセラレーターがあるイメージを取得できます。emr-7.6.0:latest を emr-7.6.0-java11:latest に置き換えて Java 11 ランタイムでイメージを取得することもできます。

```
docker pull public.ecr.aws/emr-on-eks/spark/emr-7.6.0:latest
```

Amazon EMR 6.9.0 以前のリリースのベースイメージを取得したい場合、または各リージョンの Amazon ECR レジストリアカウントから取得したい場合は、次のステップを使用してください。

1. ベースイメージ URI を選択します。次の例で示されるように、イメージ URI は *ECR-registry-account*.dkr.ecr.*Region*.amazonaws.com/spark/*container-image-tag* の形式に従います。

```
895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-6.6.0:latest
```

ご利用のリージョンのベースイメージを選択するには、ベースイメージ URI の選択の詳細 を参照してください。

2. ベースイメージが保存されている Amazon ECR リポジトリにログインします。*895885662937* と *us-west-2* を Amazon ECR レジストリアカウントと選択した AWS リージョンに置き換えます。

```
aws ecr get-login-password --region us-west-2 | docker login --username AWS --
password-stdin 895885662937.dkr.ecr.us-west-2.amazonaws.com
```

3. ベースイメージをローカルワークスペースにプルします。*emr-6.6.0:latest* を、選択したコンテナイメージタグに置き換えます。

```
docker pull 895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-6.6.0:latest
```

ステップ 2: ベースイメージをカスタマイズする

Amazon ECR からプルしたベースイメージをカスタマイズするには、次の手順を行います。

1. ローカルワークスペースで新しい Dockerfile を作成します。
2. 前の手順で作成した Dockerfile を編集し、以下のコンテンツを追加します。この Dockerfile は、895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-6.6.0:latest からプルしたコンテナイメージを使用します。

```
FROM 895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-6.6.0:latest
USER root
### Add customization commands here ####
USER hadoop:hadoop
```

3. Dockerfile にコマンドを追加して、ベースイメージをカスタマイズします。たとえば、次の Dockerfile で示すように、Python ライブラリをインストールするコマンドを追加します。

```
FROM 895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-6.6.0:latest
USER root
RUN pip3 install --upgrade boto3 pandas numpy // For python 3
USER hadoop:hadoop
```

4. Dockerfile が作成されたディレクトリと同じディレクトリから、次のコマンドを実行して Docker イメージを作成します。Docker イメージの名前を指定します (例えば、*emr6.6_custom* など)。

```
docker build -t emr6.6_custom .
```

ステップ 3: (オプション、ただし推奨) カスタムイメージを検証する

公開する前に、カスタムイメージの互換性をテストすることをお勧めします。Amazon EMR on EKS カスタムイメージ CLI を使用して、イメージに Amazon EMR on EKS で実行するために必要なファイル構造と正しい構成があるかどうかを確認できます。

> **Note**
>
> Amazon EMR on EKS カスタムイメージ CLI では、イメージにエラーがないことを確認できません。ベースイメージから依存関係を削除する際は、注意してください。

カスタムイメージを検証するには、次のステップに従います。

1. Amazon EMR on EKS カスタムイメージ CLI をダウンロードしてインストールします。詳細については、Amazon EMR on EKS カスタムイメージ CLI インストールガイドを参照してください。
2. 以下のコマンドを実行して、インストールをテストします。

```
emr-on-eks-custom-image --version
```

出力の例を以下に示します。

```
Amazon EMR on EKS Custom Image CLI
Version: x.xx
```

3. 以下のコマンドを実行して、カスタムイメージを検証します。

```
emr-on-eks-custom-image validate-image -i image_name -r release_version [-
t image_type]
```

- `-i` には、検証する必要のあるローカルイメージ URI を指定します。これには、イメージ URI や、イメージについて定義した任意の名前またはタグを指定できます。
- `-r` には、ベースイメージの正確なリリースバージョンを指定します (たとえば、`emr-6.6.0-latest` など)。
- `-t` には、イメージタイプを指定します。これが Spark イメージの場合は、`spark` を入力します。デフォルト値は `spark` です。現在の Amazon EMR on EKS カスタムイメージ CLI バージョンでは、Spark ランタイムイメージのみがサポートされています。

コマンドが正常に実行され、カスタムイメージが必須のすべての設定とファイル構造を満たしている場合、以下の例で示すように、返される出力にはすべてのテストの結果が表示されます。

```
Amazon EMR on EKS Custom Image Test
Version: x.xx
... Checking if docker cli is installed
... Checking Image Manifest
[INFO] Image ID: xxx
[INFO] Created On: 2021-05-17T20:50:07.986662904Z
[INFO] Default User Set to hadoop:hadoop : PASS
[INFO] Working Directory Set to /home/hadoop : PASS
[INFO] Entrypoint Set to /usr/bin/entrypoint.sh : PASS
[INFO] SPARK_HOME is set with value: /usr/lib/spark : PASS
[INFO] JAVA_HOME is set with value: /etc/alternatives/jre : PASS
[INFO] File Structure Test for spark-jars in /usr/lib/spark/jars: 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 bin-files in /usr/bin: PASS
... Start Running Sample Spark Job
[INFO] Sample Spark Job Test with local:///usr/lib/spark/examples/jars/spark-
examples.jar : PASS
----------------------------------------------------------------
Overall Custom Image Validation Succeeded.
----------------------------------------------------------------
```

カスタムイメージが必須の設定またはファイル構造を満たしていない場合、エラーメッセージが表示されます。返される出力では、誤った設定またはファイル構造に関する情報が提供されます。

ステップ 4: カスタムイメージを公開する

新しい Docker イメージを Amazon ECR レジストリに公開します。

1. 次のコマンドを実行して、Docker イメージを保存するための Amazon ECR リポジトリを作成します。リポジトリの名前を入力します (例えば、*emr6.6_custom_repo* など)。*us-west-2* を、ご利用のリージョンに置き換えます。

   ```
   aws ecr create-repository \
       --repository-name emr6.6_custom_repo \
       --image-scanning-configuration scanOnPush=true \
       --region us-west-2
   ```

   詳細については、Amazon ECR ユーザーガイドのリポジトリの作成を参照してください。

2. 次のコマンドを実行して、デフォルトレジストリに対して認証します。

   ```
   aws ecr get-login-password --region us-west-2 | docker login --username AWS --
   password-stdin aws_account_id.dkr.ecr.us-west-2.amazonaws.com
   ```

   詳細については、Amazon ECR ユーザーガイドのデフォルトレジストリに対して認証するを参照してください。

3. イメージにタグを付けて、作成した Amazon ECR リポジトリに公開します。

   イメージにタグを付けます。

   ```
   docker tag emr6.6_custom aws_account_id.dkr.ecr.us-
   west-2.amazonaws.com/emr6.6_custom_repo
   ```

   イメージをプッシュします。

   ```
   docker push aws_account_id.dkr.ecr.us-west-2.amazonaws.com/emr6.6_custom_repo
   ```

   詳細については、Amazon ECR ユーザーガイドのイメージを Amazon ECR にプッシュするを参照してください。

ステップ 5: カスタムイメージを使用して Amazon EMR で Spark ワークロードを送信する

カスタムイメージを作成して公開したら、カスタムイメージを使用して Amazon EMR on EKS ジョブを送信できます。

まず、次の JSON ファイルの例で示すように、start-job-run-request.json ファイルを作成して、`spark.kubernetes.container.image` パラメータを指定し、カスタムイメージを参照します。

> **Note**
>
> 以下の JSON スニペットの `entryPoint` 引数で示すように、`local://` スキームを使用してカスタムイメージで利用可能なファイルを参照できます。`local://` スキームを使用して、アプリケーションの依存関係を参照することもできます。`local://` スキームを使用して参照されるすべてのファイルと依存関係は、カスタムイメージの指定されたパスにすでに存在している必要があります。

```
{
    "name": "spark-custom-image",
    "virtualClusterId": "virtual-cluster-id",
    "executionRoleArn": "execution-role-arn",
    "releaseLabel": "emr-6.6.0-latest",
    "jobDriver": {
      "sparkSubmitJobDriver": {
        "entryPoint": "local:///usr/lib/spark/examples/jars/spark-examples.jar",
        "entryPointArguments": [
                  "10"
              ],
         "sparkSubmitParameters": "--class org.apache.spark.examples.SparkPi --conf
 spark.kubernetes.container.image=123456789012.dkr.ecr.us-west-2.amazonaws.com/
emr6.6_custom_repo"
       }
    }
}
```

次の例で示すように、applicationConfiguration プロパティを使用してカスタムイメージを参照することもできます。

```
{
    "name": "spark-custom-image",
    "virtualClusterId": "virtual-cluster-id",
    "executionRoleArn": "execution-role-arn",
    "releaseLabel": "emr-6.6.0-latest",
    "jobDriver": {
      "sparkSubmitJobDriver": {
        "entryPoint": "local:///usr/lib/spark/examples/jars/spark-examples.jar",
        "entryPointArguments": [
                  "10"
              ],
         "sparkSubmitParameters": "--class org.apache.spark.examples.SparkPi"
       }
    },
    "configurationOverrides": {
        "applicationConfiguration": [
            {
                "classification": "spark-defaults",
                "properties": {
                    "spark.kubernetes.container.image": "123456789012.dkr.ecr.us-
west-2.amazonaws.com/emr6.6_custom_repo"
                }
            }
        ]
    }
}
```

次に、start-job-run コマンドを実行してジョブを送信します。

```
aws emr-containers start-job-run --cli-input-json file://./start-job-run-request.json
```

上記の JSON の例では、*emr-6.6.0-latest* を Amazon EMR リリースバージョンに置き換えます。-latest リリースバージョンを使用して、選択したバージョンに最新のセキュリティ更新プログラムが含まれていることを確認することを強くお勧めします。Amazon EMR リリースバージョンおよびそれらのイメージタグの詳細については、「ベースイメージ URI の選択の詳細」を参照してください。

> **Note**
>
> `spark.kubernetes.driver.container.image` と `spark.kubernetes.executor.container.image` を使用して、ドライバーポッドとエグゼキュターポッドに別のイメージを指定できます。

インタラクティブエンドポイントの Docker イメージをカスタマイズする

カスタマイズされたベースカーネルイメージを実行できるように、インタラクティブエンドポイント用の Docker イメージをカスタマイズすることもできます。これは、EMR Studio からインタラクティブワークロードを実行するときに必要な依存関係を確実に確保するのに役立ちます。

1. 上記のステップ 1〜4 に従って、Docker イメージをカスタマイズします。Amazon EMR 6.9.0 リリース以降では、Amazon ECR Public Gallery からベースイメージ URI を取得できます。Amazon EMR 6.9.0 より前のリリースでは、それぞれの AWS リージョンの Amazon ECR レジストリアカウントでイメージを取得できますが、唯一の違いは Dockerfile 内のベースイメージ URI です。ベースイメージ URI は次の形式に従います。

   ```
   ECR-registry-account.dkr.ecr.Region.amazonaws.com/notebook-spark/container-image-tag
   ```

   `spark` の代わりに、ベースイメージ URI で `notebook-spark` を使用する必要があります。ベースイメージには、Spark ランタイムと、Spark ランタイムで実行されるノートブックのカーネルが含まれています。リージョンとコンテナイメージタグの選択について詳しくは、ベースイメージ URI の選択の詳細 を参照してください。

   > **Note**
   >
   > 現在、ベースイメージの上書きのみがサポートされており、ベースイメージ AWS が提供するもの以外のタイプのまったく新しいカーネルの導入はサポートされていません。

2. カスタムイメージで使用できるインタラクティブエンドポイントを作成します。

   まず、以下の内容で `custom-image-managed-endpoint.json` という JSON ファイルを作成します。

   ```
   {
   ```

```
    "name": "endpoint-name",
    "virtualClusterId": "virtual-cluster-id",
    "type": "JUPYTER_ENTERPRISE_GATEWAY",
    "releaseLabel": "emr-6.6.0-latest",
    "executionRoleArn": "execution-role-arn",
    "certificateArn": "certificate-arn",
    "configurationOverrides": {
        "applicationConfiguration": [
            {
                "classification": "jupyter-kernel-overrides",
                "configurations": [
                    {
                        "classification": "python3",
                        "properties": {
                            "container-image": "123456789012.dkr.ecr.us-
west-2.amazonaws.com/custom-notebook-python:latest"
                        }
                    },
                    {
                        "classification": "spark-python-kubernetes",
                        "properties": {
                            "container-image": "123456789012.dkr.ecr.us-
west-2.amazonaws.com/custom-notebook-spark:latest"
                        }
                    }
                ]
            }
        ]
    }
}
```

次に、次の例で示すように、JSON ファイルで指定された設定を使用してインタラクティブエンドポイントを作成します。

```
aws emr-containers create-managed-endpoint --cli-input-json custom-image-managed-
endpoint.json
```

詳細については、「Create an interactive endpoint for your virtual cluster」を参照してください。

3. EMR Studio 経由でインタラクティブエンドポイントに接続します。詳細については、Connecting from Studio を参照してください。

マルチアーキテクチャイメージを使用する

Amazon EMR on EKS は、Amazon Elastic Container Registry (Amazon ECR) のマルチアーキテクチャコンテナイメージをサポートしています。詳細については、Amazon ECR のマルチアーキテクチャコンテナイメージの紹介を参照してください。

Amazon EMR on EKS カスタムイメージは、Graviton ベースの EC2 AWS インスタンスとnon-Graviton-based EC2 インスタンスの両方をサポートします。Graviton ベースのイメージは、非 Graviton ベースのイメージと同じ Amazon ECR のイメージリポジトリに格納されています。

例えば、Docker マニフェストリストに 6.6.0 イメージがあるかどうかを確認するには、次のコマンドを実行します。

```
docker manifest inspect 895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/
emr-6.6.0:latest
```

出力は次のとおりです。arm64 アーキテクチャは Graviton インスタンス用です。amd64 は 非 Graviton インスタンス用です。

```
{
   "schemaVersion": 2,
   "mediaType": "application/vnd.docker.distribution.manifest.list.v2+json",
   "manifests": [
      {
         "mediaType": "application/vnd.docker.distribution.manifest.v2+json",
         "size": 1805,
         "digest":
 "xxx123:6b971cb47d11011ab3d45fff925e9442914b4977ae0f9fbcdcf5cfa99a7593f0",
         "platform": {
            "architecture": "arm64",
            "os": "linux"
         }
      },
      {
         "mediaType": "application/vnd.docker.distribution.manifest.v2+json",
         "size": 1805,
         "digest":
 "xxx123:6f2375582c9c57fa9838c1d3a626f1b4fc281e287d2963a72dfe0bd81117e52f",
         "platform": {
            "architecture": "amd64",
            "os": "linux"
```

```
          }
       }
    ]
}
```

マルチアーキテクチャイメージを作成するには、次のステップに従います。

1. 次の内容で Dockerfile を作成して、arm64 イメージをプルできるようにします。

```
FROM --platform=arm64 895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/
emr-6.6.0:latest
USER root

RUN pip3 install boto3 // install customizations here
USER hadoop:hadoop
```

2. マルチアーキテクチャイメージを作成するには、Amazon ECR のマルチアーキテクチャコンテナイメージの紹介の手順に従います。

> **Note**
>
> arm64 イメージは arm64 インスタンス上に作成する必要があります。同様に、amd64 インスタンス上に amd64 イメージを作成する必要があります。

   Docker buildx コマンドを使用すると、特定のインスタンスタイプごとに作成せずに、マルチアーキテクチャイメージを作成することもできます。詳細については、Leverage multi-CPU architecture support を参照してください。

3. マルチアーキテクチャイメージを作成した後、同じ spark.kubernetes.container.image パラメータを使用してイメージを参照することにより、ジョブを送信できます。Graviton ベースの EC2 インスタンスとnon-Graviton-based EC2 AWS インスタンスの両方を持つ異種クラスターでは、インスタンスはイメージをプルするインスタンスアーキテクチャに基づいて正しいアーキテクチャイメージを決定します。

ベースイメージ URI の選択の詳細

> **Note**
>
> Amazon EMR 6.9.0 以降のリリースでは、Amazon ECR Public Gallery からベースイメージを取得できるため、このページの指示に従ってベースイメージ URI を作成する必要はありません。ベースイメージのコンテナイメージタグを見つけるには、EKS での Amazon EMR の該当するリリースのリリースノートページを参照してください。

選択できるベース Docker イメージは、Amazon Elastic Container Registry (Amazon ECR) に保存されます。次の例で示すように、イメージ URI は *ECR-registry-account*.dkr.ecr.*Region*.amazonaws.com/spark/*container-image-tag* 形式に従います。

```
895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-7.6.0:latest
```

次の例で示すように、インタラクティブエンドポイントのイメージ URI は *ECR-registry-account*.dkr.ecr.*Region*.amazonaws.com/notebook-spark/*container-image-tag* 形式に従います。spark の代わりに、ベースイメージ URI で notebook-spark を使用する必要があります。

```
895885662937.dkr.ecr.us-west-2.amazonaws.com/notebook-spark/emr-7.6.0:latest
```

同様に、インタラクティブエンドポイント用の Spark python3 イメージ以外の場合、イメージ URI は *ECR-registry-account*.dkr.ecr.*Region*.amazonaws.com/notebook-python/*container-image-tag* です。次の URI の例は、正しい形式です。

```
895885662937.dkr.ecr.us-west-2.amazonaws.com/notebook-python/emr-7.6.0:latest
```

ベースイメージのコンテナイメージタグを見つけるには、EKS での Amazon EMR の該当するリリースのリリースノートページを参照してください。

リージョン別の Amazon ECR レジストリアカウント

高いネットワークレイテンシーを回避するには、最も近い からベースイメージをプルします AWS リージョン。次の表に基づいて、イメージのプル元のリージョンに対応する Amazon ECR レジストリアカウントを選択します。

Regions	Amazon ECR レジストリアカウント	
ap-east-1	736135916053	
ap-northeast-1	059004520145	
ap-northeast-2	996579266876	
ap-northeast-3	705689932349	
ap-southeast-3	946962994502	
ap-south-1	235914868574	
ap-south-2	691480105545	
ap-southeast-1	671219180197	
ap-southeast-2	038297999601	
ca-central-1	351826393999	
eu-central-1	107292555468	
eu-central-2	314408114945	
eu-north-1	830386416364	
eu-west-1	483788554619	
eu-west-2	118780647275	
eu-west-3	307523725174	
eu-south-1	238014973495	
eu-south-2	350796622945	
il-central-1	395734710648	

Regions	Amazon ECR レジストリアカウント	
me-south-1	008085056818	
me-central-1	818935616732	
sa-east-1	052806832358	
us-gov-west-1	299385240661	
us-gov-east-1	299393998622	
us-east-1	755674844232	
us-east-2	711395599931	
us-west-1	608033475327	
us-west-2	895885662937	
af-south-1	358491847878	
cn-north-1	068337069695	
cn-northwest-1	068420816659	

イメージをカスタマイズするための考慮事項

Docker イメージをカスタマイズする場合、ジョブの正確なランタイムをきめ細かなレベルで選択できます。この機能を使用する場合は、次のベストプラクティスを考慮してください。これらには、イメージのセキュリティ、設定、マウントに関する考慮事項が含まれます。

- セキュリティは、 AWS とお客様の間で共有される責任です。イメージに追加するバイナリのセキュリティパッチの適用は、お客様が行います。Amazon EMR on EKS でのセキュリティのベストプラクティス (特にカスタムイメージの最新のセキュリティ更新プログラムを入手する および最小特権の原則を適用する) に従います。
- ベースイメージをカスタマイズするときは、Docker ユーザーを hadoop:hadoop に変更して、ルートユーザーを使用してジョブが実行されないようにする必要があります。

- EKS での Amazon EMR は、実行時に、spark-defaults.conf などのイメージの設定の上にファイルをマウントします。これらの設定ファイルを上書きするには、カスタムイメージでファイルを直接変更するのではなく、ジョブの送信中に applicationOverrides パラメータを使用することをお勧めします。
- EKS での Amazon EMR は、実行時に特定のフォルダをマウントします。これらのフォルダに加えられた変更は、コンテナでは使用できません。カスタムイメージにアプリケーションまたはその依存関係を追加する場合は、次の事前定義パスの一部ではないディレクトリを選択することをお勧めします。
  - /var/log/fluentd
  - /var/log/spark/user
  - /var/log/spark/apps
  - /mnt
  - /tmp
  - /home/hadoop
- カスタマイズしたイメージは、Amazon ECR、Docker Hub、プライベートエンタープライズリポジトリなど、Docker と互換性があるリポジトリにアップロードできます。選択した Docker リポジトリを使用した Amazon EKS クラスター認証の設定について詳しくは、「Pull an Image from a Private Registry」を参照してください。

EKS での Amazon EMR を使用した Flink ジョブの実行

Amazon EMR リリース 6.13.0 以降では、EKS での Amazon EMR のジョブ送信モデルとして、Apache Flink または Flink Kubernetes オペレータを使用する EKS での Amazon EMR をサポートしています。Apache Flink で EKS での Amazon EMR を使用すると、Amazon EMR リリースランタイムを使用して Flink アプリケーションを独自の Amazon EKS クラスターにデプロイおよび管理できます。Amazon EKS クラスターに Flink Kubernetes オペレーターをデプロイすると、そのオペレータを使用して Flink アプリケーションを直接送信できます。オペレータは Flink アプリケーションのライフサイクルを管理します。

トピック

Flink Kubernetes オペレータのセットアップと使用

以下のページでは、Amazon EMR on EKS で Flink ジョブを実行できるよう、Flink Kubernetes オペレータをセットアップして使用する方法について説明します。トピックとして、必要な前提条件、環境のセットアップ方法、Amazon EMR on EKS での Flink アプリケーションの実行方法などが紹介されています。

トピック

Amazon EMR on EKS での Flink Kubernetes オペレータのセットアップ

Flink Kubernetes オペレータを Amazon EKS にインストールする前に、以下のタスクを完了してセットアップを行います。Amazon Web Services (AWS) に既にサインアップしていて、Amazon EKS を既に使用している場合、Amazon EMR on EKS を使用する準備はほぼ整っています。Amazon EKS で Flink Kubernetes オペレータのセットアップを行うには、以下のタスクを完了します。前提条件のいずれかを既に完了している場合は、その前提条件をスキップして、次の前提条件に進むことができます。

- の最新バージョンをインストールまたは更新する AWS CLI – をすでにインストールしている場合は AWS CLI、最新バージョンがあることを確認します。
- kubectl と eksctl の設定 — eksctl は、Amazon EKS との通信に使用するコマンドラインツールです。
- Helm のインストール – Kubernetes 用の Helm パッケージマネージャーを使用すると、Kubernetes クラスターにアプリケーションをインストールして管理できます。
- Amazon EKS – eksctl の使用開始 – Amazon EKS にノードを持つ新しい Kubernetes クラスターを作成する手順に従います。
- Amazon EMR リリースラベルの選択 (リリース 6.13.0 以降) – Flink Kubernetes オペレータは、Amazon EMR リリース 6.13.0 以降でサポートされています。
- Amazon EKS クラスターでサービスアカウント (IRSA) の IAM ロールを有効にします。
- ジョブ実行ロールを作成します。
- ジョブ実行ロールの信頼ポリシーを更新する。
- オペレータ実行ロールを作成します。この手順はオプションです。Flink ジョブとオペレータに同じロールを使用できます。オペレータに異なる IAM ロールを使用する場合は、別のロールを作成できます。
- オペレータ実行ロールの信頼ポリシーを更新します。Amazon EMR Flink Kubernetes オペレータサービスアカウントにロールを使用する場合は、そのロールの信頼ポリシーエントリを 1 つ明示的に追加する必要があります。次の例に示した形式に従うことができます。

```
{
   "Version": "2012-10-17",
   "Statement": [
       {
           "Effect": "Allow",
           "Principal": {
               "Federated": "arn:aws:iam::ACCOUNT_ID:oidc-provider/OIDC_PROVIDER"
```

```
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringLike": {
                    "OIDC_PROVIDER:sub": "system:serviceaccount:NAMESPACE:emr-
containers-sa-flink-operator"
                }
            }
        }
    ]
}
```

Amazon EMR on EKS での Flink Kubernetes オペレータのインストール

このトピックでは、Flink のデプロイを準備することで、Amazon EKS で Flink Kubernetes オペレータを使用開始する際に役立ちます。

Kubernetes オペレータをインストールする

Apache Flink 用の Kubernetes オペレータをインストールするには、次の手順を実行します。

1. 「the section called “設定”」のステップをまだ完了していない場合は完了します。
2. *cert-manager* を (Amazon EKS クラスターごとに 1 回) インストールして、ウェブフックコンポーネントを追加できるようにします。

   ```
   kubectl apply -f https://github.com/cert-manager/cert-manager/releases/download/
   v1.12.0/cert-manager.yaml
   ```

3. Helm チャートをインストールします。

   ```
   export VERSION=7.6.0 # The Amazon EMR release version
   export NAMESPACE=The Kubernetes namespace to deploy the operator

   helm install flink-kubernetes-operator \
   oci://public.ecr.aws/emr-on-eks/flink-kubernetes-operator \
   --version $VERSION \
   --namespace $NAMESPACE
   ```

   出力例:

```
NAME: flink-kubernetes-operator
LAST DEPLOYED: Tue May 31 17:38:56 2022
NAMESPACE: $NAMESPACE
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

4. デプロイが完了するまで待ち、Helm チャートのインストールを検証します。

```
kubectl wait deployment flink-kubernetes-operator --namespace $NAMESPACE --for
 condition=Available=True --timeout=30s
```

5. デプロイが完了すると、次のメッセージが表示されます。

```
deployment.apps/flink-kubernetes-operator condition met
```

6. 次のコマンドを使用して、デプロイしたオペレータを表示します。

```
helm list --namespace $NAMESPACE
```

次に、出力の例を示します。この例では、アプリのバージョン `x.y.z-amzn-n` が Amazon EMR on EKS リリースの Flink オペレータのバージョンと一致しています。詳細については、「Apache Flink をサポートしている Amazon EMR on EKS のリリース」を参照してください。

```
NAME                            NAMESPACE    REVISION    UPDATED
               STATUS      CHART                                    APP VERSION

flink-kubernetes-operator    $NAMESPACE   1           2023-02-22 16:43:45.24148
 -0500 EST    deployed    flink-kubernetes-operator-emr-7.6.0    x.y.z-amzn-n
```

Flink アプリケーションを実行する

Amazon EMR 6.13.0 以降では、EKS 上の Amazon EMR のアプリケーションモードで Flink Kubernetes オペレーターを使用して Flink アプリケーションを実行できます。Amazon EMR 6.15.0 以降では、Flink アプリケーションをセッションモードで実行することもできます。このページでは、EKS 上の Amazon EMR で Flink アプリケーションを実行するために使用できる両方の方法について説明します。

> **Note**
>
> Flink ジョブを送信する際に高可用性メタデータを保存するには、Amazon S3 バケットを作成する必要があります。この機能を使用しない場合には無効にできます。これは、デフォルトでは有効になっています。

前提条件 - Flink Kubernetes オペレータを使用して Flink アプリケーションを実行する前に、the section called “設定” と the section called “Kubernetes オペレータをインストールする” のステップを完了してください。

Application mode

Amazon EMR 6.13.0 以降では、EKS 上の Amazon EMR のアプリケーションモードで Flink Kubernetes オペレーターを使用して Flink アプリケーションを実行できます。

1. 以下の例のように FlinkDeployment 定義ファイル basic-example-app-cluster.yaml を作成します。いずれかのオプトイン AWS リージョンを有効にして使用する場合は、必ずコメントを解除して設定してくださいfs.s3a.endpoint.region。

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: basic-example-app-cluster
spec:
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "2"
    #fs.s3a.endpoint.region: OPT_IN_AWS_REGION_NAME
    state.checkpoints.dir: CHECKPOINT_S3_STORAGE_PATH
    state.savepoints.dir: SAVEPOINT_S3_STORAGE_PATH
  flinkVersion: v1_17
  executionRoleArn: JOB_EXECUTION_ROLE_ARN
  emrReleaseLabel: "emr-6.13.0-flink-latest" # 6.13 or higher
  jobManager:
    storageDir: HIGH_AVAILABILITY_STORAGE_PATH
    resource:
      memory: "2048m"
      cpu: 1
  taskManager:
    resource:
      memory: "2048m"
```

```
      cpu: 1
  job:
    # if you have your job jar in S3 bucket you can use that path as well
    jarURI: local:///opt/flink/examples/streaming/StateMachineExample.jar
    parallelism: 2
    upgradeMode: savepoint
    savepointTriggerNonce: 0
  monitoringConfiguration:
    cloudWatchMonitoringConfiguration:
       logGroupName: LOG_GROUP_NAME
```

2. 次のコマンドで Flink デプロイを送信します。これにより、basic-example-app-cluster という名前で FlinkDeployment オブジェクトも作成されます。

```
kubectl create -f basic-example-app-cluster.yaml -n <NAMESPACE>
```

3. Flink UI にアクセスします。

```
kubectl port-forward deployments/basic-example-app-cluster 8081 -n NAMESPACE
```

4. localhost:8081 を開いて、Flink ジョブをローカルに表示します。
5. ジョブをクリーンアップします。チェックポイント、高可用性、セーブポイントメタデータ、CloudWatch ログなど、このジョブ用に作成された S3 アーティファクトを忘れずにクリーンアップしてください。

Flink Kubernetes オペレータを利用して Flink にアプリケーションを送信する方法の詳細については、GitHub の apache/flink-kubernetes-operator フォルダにある「Flink Kubernetes operator examples」を参照してください。

Session mode

Amazon EMR 6.15.0 以降では、EKS 上の Amazon EMR のセッションモードで Flink Kubernetes オペレーターを使用して Flink アプリケーションを実行できます。

1. 以下の例のように basic-example-app-cluster.yaml という名前の FlinkDeployment 定義ファイルを作成します。いずれかのオプトイン AWS リージョンを有効にして使用する場合は、必ずコメントを解除して設定してください fs.s3a.endpoint.region。

```
apiVersion: flink.apache.org/v1beta1
```

```
kind: FlinkDeployment
metadata:
  name: basic-example-session-cluster
spec:
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "2"
    #fs.s3a.endpoint.region: OPT_IN_AWS_REGION_NAME
    state.checkpoints.dir: CHECKPOINT_S3_STORAGE_PATH
    state.savepoints.dir: SAVEPOINT_S3_STORAGE_PATH
  flinkVersion: v1_17
  executionRoleArn: JOB_EXECUTION_ROLE_ARN
  emrReleaseLabel: "emr-6.15.0-flink-latest"
  jobManager:
    storageDir: HIGH_AVAILABILITY_S3_STORAGE_PATH
    resource:
      memory: "2048m"
      cpu: 1
  taskManager:
    resource:
      memory: "2048m"
      cpu: 1
  monitoringConfiguration:
    s3MonitoringConfiguration:
       logUri:
    cloudWatchMonitoringConfiguration:
       logGroupName: LOG_GROUP_NAME
```

2. 次のコマンドで Flink デプロイを送信します。これにより、basic-example-session-cluster という名前で FlinkDeployment オブジェクトも作成されます。

```
kubectl create -f basic-example-app-cluster.yaml -n NAMESPACE
```

3. 次のコマンドを使用して、セッションクラスター LIFECYCLE が STABLE であることを確認します。

```
kubectl get flinkdeployments.flink.apache.org basic-example-session-cluster -
n NAMESPACE
```

出力は次の例と類似したものになります。

```
NAME                              JOB STATUS   LIFECYCLE STATE
basic-example-session-cluster                          STABLE
```

4. 次のサンプルコンテンツを含む FlinkSessionJob カスタム定義リソースファイル basic-session-job.yaml を作成します。

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkSessionJob
metadata:
  name: basic-session-job
spec:
  deploymentName: basic-session-deployment
  job:
    # If you have your job jar in an S3 bucket you can use that path.
    # To use jar in S3 bucket, set
    # OPERATOR_EXECUTION_ROLE_ARN (--set emrContainers.operatorExecutionRoleArn=
$OPERATOR_EXECUTION_ROLE_ARN)
    # when you install Spark operator
    jarURI: https://repo1.maven.org/maven2/org/apache/flink/flink-examples-
streaming_2.12/1.16.1/flink-examples-streaming_2.12-1.16.1-TopSpeedWindowing.jar
    parallelism: 2
    upgradeMode: stateless
```

5. 次のコマンドで Flink セッションジョブを送信します。これにより、FlinkSessionJob オブジェクト basic-session-job が作成されます。

```
kubectl apply -f basic-session-job.yaml -n $NAMESPACE
```

6. 次のコマンドを使用して、セッションクラスター LIFECYCLE が STABLE、JOB STATUS が RUNNING であることを確認します。

```
kubectl get flinkdeployments.flink.apache.org basic-example-session-cluster -
n NAMESPACE
```

出力は次の例と類似したものになります。

```
NAME                              JOB STATUS   LIFECYCLE STATE
basic-example-session-cluster     RUNNING      STABLE
```

7. Flink UI にアクセスします。

```
kubectl port-forward deployments/basic-example-session-cluster 8081 -n NAMESPACE
```

8. localhost:8081 を開いて、Flink ジョブをローカルに表示します。

9. ジョブをクリーンアップします。チェックポイント、高可用性、セーブポイントメタデータ、CloudWatch ログなど、このジョブ用に作成された S3 アーティファクトを忘れずにクリーンアップしてください。

Flink アプリケーションを実行するためのセキュリティロールのアクセス権限

このトピックでは、Flink アプリケーションをデプロイして実行するためのセキュリティロールについて説明します。デプロイを管理し、ジョブを作成および管理するには、オペレーターロールとジョブロールの 2 つのロールが必要です。このトピックでは、それらを紹介し、そのアクセス権限を一覧表示します。

ロールベースのアクセスコントロール

オペレータをデプロイし、Flink ジョブを実行するには、オペレータロールとジョブロールという 2 つの Kubernetes ロールを作成する必要があります。Amazon EMR は、オペレータをインストールするときに、デフォルトでロールを 2 つ作成します。

オペレータロール

オペレータロールを使用すると、`flinkdeployments` を管理して Flink ジョブやその他のリソース (サービスなど) ごとに JobManager を作成および管理できます。

オペレータロールのデフォルト名は `emr-containers-sa-flink-operator` で、使用するには以下の権限が必要です。

```
rules:
- apiGroups:
  - ""
  resources:
  - pods
  - services
  - events
  - configmaps
  - secrets
  - serviceaccounts
  verbs:
  - '*'
- apiGroups:
  - rbac.authorization.k8s.io
  resources:
```

```
  - roles
  - rolebindings
  verbs:
  - '*'
- apiGroups:
  - apps
  resources:
  - deployments
  - deployments/finalizers
  - replicasets
  verbs:
  - '*'
- apiGroups:
  - extensions
  resources:
  - deployments
  - ingresses
  verbs:
  - '*'
- apiGroups:
  - flink.apache.org
  resources:
  - flinkdeployments
  - flinkdeployments/status
  - flinksessionjobs
  - flinksessionjobs/status
  verbs:
  - '*'
- apiGroups:
  - networking.k8s.io
  resources:
  - ingresses
  verbs:
  - '*'
- apiGroups:
  - coordination.k8s.io
  resources:
  - leases
  verbs:
  - '*'
```

ジョブロール

JobManager は、ジョブロールを使用して、ジョブごとに TaskManagers と ConfigMaps を作成および管理します。

```
rules:
- apiGroups:
  - ""
  resources:
  - pods
  - configmaps
  verbs:
  - '*'
- apiGroups:
  - apps
  resources:
  - deployments
  - deployments/finalizers
  verbs:
  - '*'
```

Amazon EMR on EKS での Flink Kubernetes オペレータのアンインストール

次の手順に従って、Flink Kubernetes オペレータをアンインストールします。

1. オペレータを削除します。

   ```
   helm uninstall flink-kubernetes-operator -n <NAMESPACE>
   ```

2. Helm がアンインストールしない Kubernetes リソースを削除します。

   ```
   kubectl delete serviceaccounts, roles, rolebindings -l emr-
   containers.amazonaws.com/component=flink.operator --namespace <namespace>
   kubectl delete crd flinkdeployments.flink.apache.org
    flinksessionjobs.flink.apache.org
   ```

3. (オプション) cert-manager を削除します。

   ```
   kubectl delete -f https://github.com/jetstack/cert-manager/releases/download/
   v1.12.0/cert-manager.yaml
   ```

Flink Native Kubernetes の使用

Amazon EMR リリース 6.13.0 以降では、Flink アプリケーションを Amazon EMR on EKS クラスターに送信して実行するために使用できるコマンドラインツールとして Flink Native Kubernetes がサポートされています。

トピック

Amazon EMR on EKS での Flink Native Kubernetes のセットアップ

Amazon EMR on EKS で Flink CLI を使用してアプリケーションを実行する前に、次のタスクを完了してセットアップを行います。Amazon Web Services (AWS) に既にサインアップしていて、Amazon EKS を既に使用している場合、Amazon EMR on EKS を使用する準備はほぼ整っています。前提条件のいずれかを既に完了している場合は、その前提条件をスキップして、次の前提条件に進むことができます。

- の最新バージョンをインストールまたは更新する AWS CLI – を既にインストールしている場合は AWS CLI、最新バージョンがあることを確認します。
- Amazon EKS – eksctl の使用開始 – Amazon EKS にノードを持つ新しい Kubernetes クラスターを作成する手順に従います。
- Amazon EMR ベースイメージ URI (リリース 6.13.0 以上) を選択する – Flink Kubernetes コマンドは、Amazon EMR リリース 6.13.0 以降でサポートされています。
- JobManager サービスアカウントに TaskManager ポッドを作成および監視するための適切な権限があることを確認します。詳細については、「Native Kubernetes の Flink JobManager サービスアカウントセキュリティ要件」を参照してください。
- ローカルの AWS 認証情報プロファイルをセットアップします。
- Flink アプリケーションの実行対象とする Amazon EKS クラスターの kubeconfig ファイルを作成または更新します。

Amazon EMR on EKS の Flink Native Kubernetes を使ってみる

これらの手順では、Flink アプリケーションの設定、サービスアカウントの設定、および実行の方法を示します。Flink Native Kubernetes は、稼働中の Kubernetes クラスターに Flink をデプロイするために使用されます。

Flink アプリケーションを設定して実行する

Amazon EMR 6.13.0 以降では、Amazon EKS クラスターで Flink アプリケーションを実行できるよう、Flink Native Kubernetes をサポートしています。Flink アプリケーションを実行するには、次の手順に従います。

1. Flink Native Kubernetes コマンドを使用して Flink アプリケーションを実行する前に、「the section called “設定”」のステップを完了してください。
2. Flink をダウンロードおよびインストールします。
3. 次の環境変数の値を設定します。

```
#Export the FLINK_HOME environment variable to your local installation of Flink
export FLINK_HOME=/usr/local/bin/flink #Will vary depending on your installation
export NAMESPACE=flink
export CLUSTER_ID=flink-application-cluster
export IMAGE=<123456789012.dkr.ecr.sample-AWS #####-.amazonaws.com/flink/
emr-6.13.0-flink:latest>
export FLINK_SERVICE_ACCOUNT=emr-containers-sa-flink
export FLINK_CLUSTER_ROLE_BINDING=emr-containers-crb-flink
```

4. Kubernetes リソースを管理するためのサービスアカウントを作成します。

```
kubectl create serviceaccount $FLINK_SERVICE_ACCOUNT -n $NAMESPACE
kubectl create clusterrolebinding $FLINK_CLUSTER_ROLE_BINDING --clusterrole=edit --
serviceaccount=$NAMESPACE:$FLINK_SERVICE_ACCOUNT
```

5. `run-application` CLI コマンドを実行します。

```
$FLINK_HOME/bin/flink run-application \
    --target kubernetes-application \
    -Dkubernetes.namespace=$NAMESPACE \
    -Dkubernetes.cluster-id=$CLUSTER_ID \
    -Dkubernetes.container.image.ref=$IMAGE \
    -Dkubernetes.service-account=$FLINK_SERVICE_ACCOUNT \
    local:///opt/flink/examples/streaming/Iteration.jar
```

```
2022-12-29 21:13:06,947 INFO  org.apache.flink.kubernetes.utils.KubernetesUtils
            [] - Kubernetes deployment requires a fixed port. Configuration
 blob.server.port will be set to 6124
2022-12-29 21:13:06,948 INFO  org.apache.flink.kubernetes.utils.KubernetesUtils
            [] - Kubernetes deployment requires a fixed port. Configuration
 taskmanager.rpc.port will be set to 6122
2022-12-29 21:13:07,861 WARN
 org.apache.flink.kubernetes.KubernetesClusterDescriptor      [] - Please note that
 Flink client operations(e.g. cancel, list, stop, savepoint, etc.) won't work from
 outside the Kubernetes cluster since 'kubernetes.rest-service.exposed.type' has
 been set to ClusterIP.
2022-12-29 21:13:07,868 INFO
 org.apache.flink.kubernetes.KubernetesClusterDescriptor      [] - Create flink
 application cluster flink-application-cluster successfully, JobManager Web
 Interface: http://flink-application-cluster-rest.flink:8081
```

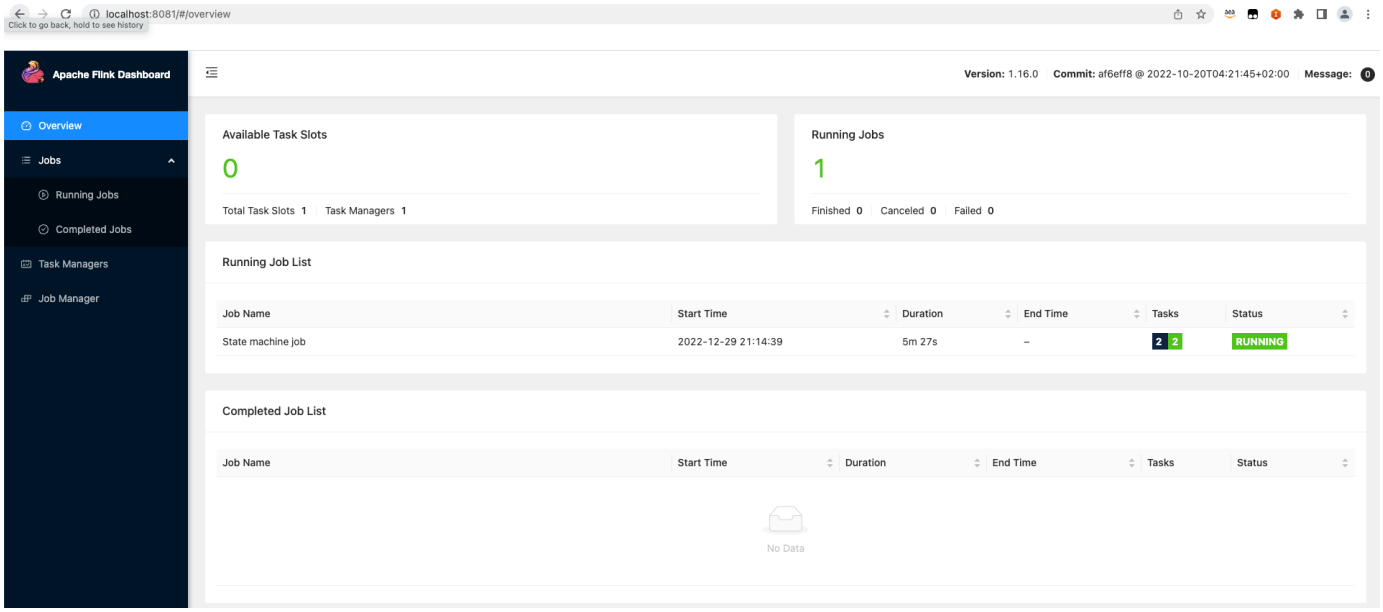
6. 作成した Kubernetes リソースを確認します。

```
kubectl get all -n <namespace>
NAME READY STATUS RESTARTS AGE
pod/flink-application-cluster-546687cb47-w2p2z 1/1 Running 0 3m37s
pod/flink-application-cluster-taskmanager-1-1 1/1 Running 0 3m24s

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
service/flink-application-cluster ClusterIP None <none> 6123/TCP,6124/TCP 3m38s
service/flink-application-cluster-rest ClusterIP 10.100.132.158 <none> 8081/TCP
 3m38s

NAME READY UP-TO-DATE AVAILABLE AGE
deployment.apps/flink-application-cluster 1/1 1 1 3m38s

NAME DESIRED CURRENT READY AGE
replicaset.apps/flink-application-cluster-546687cb47 1 1 1 3m38s
```

7. 8081 にポート転送します。

```
kubectl port-forward service/flink-application-cluster-rest 8081 -n <namespace>
Forwarding from 127.0.0.1:8081 -> 8081
```

8. Flink UI にローカルにアクセスします。



9. Flink アプリケーションを削除します。

```
kubectl delete deployment.apps/flink-application-cluster -n <namespace>
deployment.apps "flink-application-cluster" deleted
```

Flink にアプリケーションを送信する方法の詳細については、Apache Flink ドキュメントの「Native Kubernetes」を参照してください。

Native Kubernetes の Flink JobManager サービスアカウントセキュリティ要件

Flink JobManager ポッドは、Kubernetes サービスアカウントを使用して Kubernetes API サーバーにアクセスし、TaskManager ポッドを作成および監視します。JobManager サービスアカウントには、TaskManager ポッドを作成/削除するための適切な権限が必要です。また、TaskManager からリーダーの ConfigMaps を監視して、クラスター内の JobManager と ResourceManager のアドレスを取得できるようにする必要があります。

このサービスアカウントには、次のルールが適用されます。

```
rules:
- apiGroups:
  - ""
  resources:
  - pods
```

```
  verbs:
  - "*"
- apiGroups:
  - ""
  resources:
  - services
  verbs:
  - "*"
- apiGroups:
  - ""
  resources:
  - configmaps
  verbs:
  - "*"
- apiGroups:
  - "apps"
  resources:
  - deployments
  verbs:
  - "*"
```

Flink と FluentD の Docker イメージのカスタマイズ

Amazon EMR on EKS の Docker イメージを Apache Flink または FluentD のイメージを使用してカスタマイズするには、次の手順に従います。これには、ベースイメージの取得、カスタマイズ、公開、ワークロードの送信に関する技術ガイダンスが含まれます。

トピック

前提条件

Docker イメージをカスタマイズする前に、以下の前提条件を満たしていることを確認してください。

- Amazon EMR on EKS での Flink Kubernetes オペレータのセットアップ手順が完了している。
- ご自分の環境に Docker をインストールしている。詳細については、Get Docker を参照してください。

ステップ 1: Amazon Elastic Container Registry からベースイメージを取得する

ベースイメージには、Amazon EMR ランタイムと、他の AWS のサービスへのアクセスに必要なコネクタが含まれています。Flink バージョン 6.14.0 以降で Amazon EMR on EKS を使用している場合は、Amazon ECR Public Gallery からベースイメージを取得できます。ギャラリーを参照してイメージリンクを見つけ、そのイメージをローカルワークスペースに取り込みます。例えば、Amazon EMR 6.14.0 リリースの場合、次の `docker pull` コマンドは最新の標準ベースイメージを返します。`emr-6.14.0:latest` を必要なリリースバージョンに置き換えます。

```
docker pull public.ecr.aws/emr-on-eks/flink/emr-6.14.0-flink:latest
```

以下は、Flink ギャラリーイメージと Fluentd ギャラリーイメージへのリンクです。

- emr-on-eks/flink/emr-6.14.0-flink
- emr-on-eks/fluentd/emr-6.14.0(

ステップ 2: ベースイメージをカスタマイズする

以下の手順では、Amazon ECR からプルしたベースイメージをカスタマイズする方法を説明します。

1. ローカルワークスペースで新しい `Dockerfile` を作成します。
2. `Dockerfile` を編集し、以下の内容を追加します。この `Dockerfile` は、`public.ecr.aws/emr-on-eks/flink/emr-7.7.0-flink:latest` からプルしたコンテナイメージを使用します。

   ```
   FROM public.ecr.aws/emr-on-eks/flink/emr-7.7.0-flink:latest
   USER root
   ### Add customization commands here ####
   USER hadoop:hadoop
   ```

   `Fluentd` を使用している場合は、次の設定を使用します。

```
FROM public.ecr.aws/emr-on-eks/fluentd/emr-7.7.0:latest
USER root
### Add customization commands here ####
USER hadoop:hadoop
```

3. Dockerfile にコマンドを追加して、ベースイメージをカスタマイズします。次のコマンドは、Python ライブラリをインストールする方法を示しています。

```
FROM public.ecr.aws/emr-on-eks/flink/emr-7.7.0-flink:latest
USER root
RUN pip3 install --upgrade boto3 pandas numpy // For python 3
USER hadoop:hadoop
```

4. DockerFile が作成されたディレクトリと同じディレクトリで、次のコマンドを実行して Docker イメージを作成します。-t フラグの後に入力するフィールドは、イメージのカスタム名です。

```
docker build -t <YOUR_ACCOUNT_ID>.dkr.ecr.<YOUR_ECR_REGION>.amazonaws.com/
<ECR_REPO>:<ECR_TAG>
```

ステップ 3: カスタムイメージを公開する

新しい Docker イメージを Amazon ECR レジストリに公開できるようになりました。

1. 次のコマンドを実行して、Docker イメージを保存するための Amazon ECR リポジトリを作成します。リポジトリの名前 (例: emr_custom_repo.) を指定します。詳細については、「Amazon Elastic Container Registry ユーザーガイド」の「リポジトリの作成」を参照してください。

```
aws ecr create-repository \
       --repository-name emr_custom_repo \
       --image-scanning-configuration scanOnPush=true \
       --region <AWS_REGION>
```

2. 次のコマンドを実行して、デフォルトレジストリに対して認証します。詳細については、「Amazon Elastic Container Registry ユーザーガイド」の「デフォルトレジストリの認証」を参照してください。

```
aws ecr get-login-password --region <AWS_REGION> | docker login --username AWS --
password-stdin <AWS_ACCOUNT_ID>.dkr.ecr.<YOUR_ECR_REGION>.amazonaws.com
```

3. イメージをプッシュします。詳細については、「Amazon Elastic Container Registry ユーザーガイド」の「イメージを Amazon ECR にプッシュする」を参照してください。

```
docker push <YOUR_ACCOUNT_ID>.dkr.ecr.<YOUR_ECR_REGION>.amazonaws.com/
<ECR_REPO>:<ECR_TAG>
```

ステップ 4: カスタムイメージを使用して Amazon EMR で Flink ワークロードを送信する

カスタムイメージを使用するには、FlinkDeployment 仕様に以下の変更を加えます。そのためには、デプロイ仕様の spec.image 行に独自のイメージを入力します。

```
apiVersion: flink.apache.org/v1beta1
   kind: FlinkDeployment
   metadata:
     name: basic-example
   spec:
     flinkVersion: v1_18
     image: <YOUR_ACCOUNT_ID>.dkr.ecr.<YOUR_ECR_REGION>.amazonaws.com/
<ECR_REPO>:<ECR_TAG>
     imagePullPolicy: Always
     flinkConfiguration:
           taskmanager.numberOfTaskSlots: "1"
```

Fluentd ジョブにカスタムイメージを使用するには、デプロイ仕様の monitoringConfiguration.image 行に独自のイメージを入力します。

```
  monitoringConfiguration:
       image: <YOUR_ACCOUNT_ID>.dkr.ecr.<YOUR_ECR_REGION>.amazonaws.com/
<ECR_REPO>:<ECR_TAG>
       cloudWatchMonitoringConfiguration:
         logGroupName: flink-log-group
         logStreamNamePrefix: custom-fluentd
```

Flink Kubernetes オペレータと Flink ジョブのモニタリング

このセクションでは、Amazon EMR on EKS で Flink ジョブをモニタリングする方法をいくつか紹介します。これには、Flink と Amazon Managed Service for Prometheus の統合、ジョブのステータスとメトリクスを提供する Flink Web Dashboard の使用、Amazon S3 と Amazon CloudWatchにログデータを送信するためのモニタリング設定の使用が含まれます。

トピック

- Amazon Managed Service for Prometheus を使用した Flink ジョブのモニタリング
- Flink UI を使用した Flink ジョブのモニタリング
- モニタリング設定を使用した、Flink Kubernetes オペレータと Flink ジョブのモニタリング

Amazon Managed Service for Prometheus を使用した Flink ジョブのモニタリング

Apache Flink を Amazon Managed Service for Prometheus (管理ポータル) とインテグレーションできます。Amazon Managed Service for Prometheus では、Amazon Managed Service for Prometheus サーバーからメトリクスを Amazon EKS で実行されているクラスターに取り込むことができます。Amazon Managed Service for Prometheus は、Amazon EKS クラスターで既に実行されている Prometheus サーバーと連携して動作します。Amazon Managed Service for Prometheus と Amazon EMR Flink オペレータとのインテグレーションを実行すると、Prometheus サーバーが自動的にデプロイされ、Amazon Managed Service for Prometheus とインテグレーションするように設定されます。

1. 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。後でそのリモート書き込み URL が必要になります。
2. サービスアカウントの IAM ロールをセットアップします。

   このオンボーディング方法では、Prometheus サーバーが稼働している Amazon EKS クラスターのサービスアカウントの IAM ロールを使用します。こうしたロールは、サービスロールとも呼ばれます。

   まだサービスロールがない場合は、Amazon EKS クラスターからメトリクスを取り込めるようにサービスロールをセットアップします。

   続行する前に、IAM ロールを `amp-iamproxy-ingest-role` という名前で作成します。

3. Amazon EMR Flink オペレータを Amazon Managed Service for Prometheus と共にインストールします。

Amazon Managed Service for Prometheus ワークスペース、Amazon Managed Service for Prometheus 専用の IAM ロール、および必要な権限が用意できたので、Amazon EMR Flink オペレータをインストールできます。

enable-amp.yaml ファイルを作成します。このファイルを使用すると、カスタム設定で Amazon Managed Service for Prometheus の設定を上書きできます。必ず独自のロールを使用してください。

```
kube-prometheus-stack:
    prometheus:
    serviceAccount:
        create: true
        name: "amp-iamproxy-ingest-service-account"
        annotations:
            eks.amazonaws.com/role-arn: "arn:aws:iam::<AWS_ACCOUNT_ID>:role/amp-
iamproxy-ingest-role"
    remoteWrite:
        - url: <AMAZON_MANAGED_PROMETHEUS_REMOTE_WRITE_URL>
        sigv4:
            region: <AWS_REGION>
        queueConfig:
            maxSamplesPerSend: 1000
            maxShards: 200
            capacity: 2500
```

Helm Install --set コマンドを使用して、flink-kubernetes-operator チャートにオーバーライドを渡します。

```
helm upgrade -n <namespace> flink-kubernetes-operator \
   oci://public.ecr.aws/emr-on-eks/flink-kubernetes-operator \
   --set prometheus.enabled=true
   -f enable-amp.yaml
```

このコマンドにより、オペレータ内の Prometheus レポーターがポート 9999 に自動的にインストールされます。また、以後 FlinkDeployment を指定すると、metrics ポートが 9249 に公開されます。

- Flink オペレータメトリクスは、Prometheus のラベル `flink_k8soperator_` の下に表示されます。
- Flink タスクマネージャーメトリクスは、Prometheus のラベル `flink_taskmanager_` の下に表示されます。
- Flink ジョブマネージャーメトリクスは、Prometheus のラベル `flink_jobmanager_` の下に表示されます。

Flink UI を使用した Flink ジョブのモニタリング

実行中の Flink アプリケーションの正常性とパフォーマンスをモニタリングするには、Flink Web Dashboard を使用します。ジョブのステータス、TaskManagers の数、ジョブのメトリクスとログといった情報を確認できます。また、Flink ジョブの設定を表示して変更できるほか、Flink クラスターとやり取りしてジョブの送信やキャンセルができます。

Kubernetes で実行中の Flink アプリケーションの Flink Web Dashboard にアクセスするには

1. `kubectl port-forward` コマンドを使用して、Flink アプリケーションの TaskManager ポッドで Flink Web Dashboard が実行されているポートにローカルポートを転送します。デフォルトでは、このポートは 8081 です。*`deployment-name`* を上記の Flink アプリケーションデプロイの名前に置き換えます。

   ```
   kubectl get deployments -n namespace
   ```

   出力例:

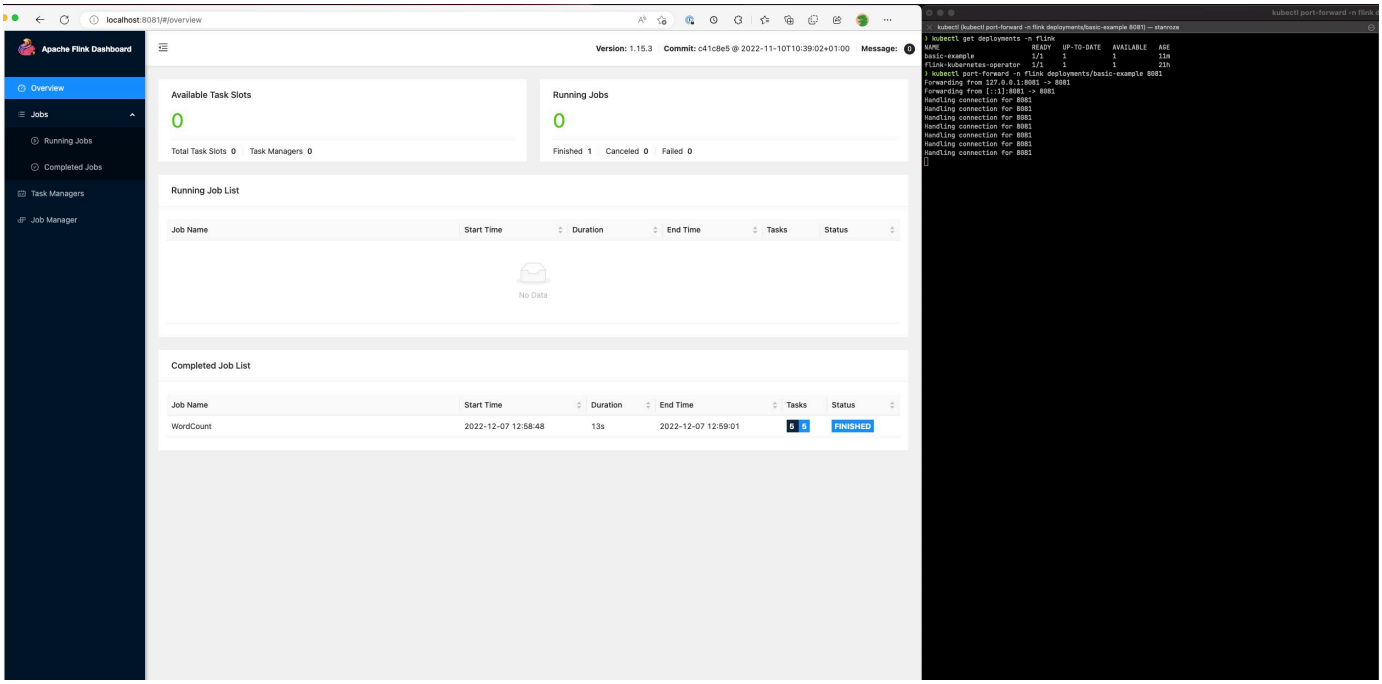
   ```
   kubectl get deployments -n flink-namespace
   NAME                        READY   UP-TO-DATE   AVAILABLE   AGE
   basic-example               1/1       1            1           11m
   flink-kubernetes-operator   1/1       1            1           21h
   ```

   ```
   kubectl port-forward deployments/deployment-name 8081 -n namespace
   ```

2. 別のポートをローカルに使用する場合は、*`local-port`*:8081 パラメータを使用します。

   ```
   kubectl port-forward -n flink deployments/basic-example 8080:8081
   ```

3. ウェブブラウザで、`http://localhost:8081` (カスタムローカルポートを使用した場合は `http://localhost:local-port`) に移動して、Flink Web Dashboard にアクセスします。ジョブのステータス、TaskManagers の数、ジョブのメトリクスとログといった実行中の Flink アプリケーションの情報を確認できます。



モニタリング設定を使用した、Flink Kubernetes オペレータと Flink ジョブのモニタリング

モニタリング設定を使用すると、Flink アプリケーションログとオペレータログを S3 や CloudWatch (どちらか一方または両方を選択可能) にアーカイブするように簡単にセットアップできます。このようにすると、FluentD サイドカーが JobManager ポッドと TaskManager ポッドに追加され、その後、設定しておいたシンクにこれらのコンポーネントのログが転送されます。

> **Note**
>
> この機能は他の AWS のサービスとやり取りする必要があるため、この機能を使用できるようにするには、Flink オペレータと Flink ジョブ (サービスアカウント) のサービスアカウントに IAM ロールをセットアップする必要があります。このセットアップを行うには、「Amazon EMR on EKS での Flink Kubernetes オペレータのセットアップ」で IRSA を使用必要があります。

Flink アプリケーションログ

この設定は、次の方法で定義できます。

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: basic-example
spec:
  image: FLINK IMAGE TAG
  imagePullPolicy: Always
  flinkVersion: v1_17
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "2"
  executionRoleArn: JOB EXECUTION ROLE
  jobManager:
    resource:
      memory: "2048m"
      cpu: 1
  taskManager:
    resource:
      memory: "2048m"
      cpu: 1
  job:
    jarURI: local:///opt/flink/examples/streaming/StateMachineExample.jar
  monitoringConfiguration:
    s3MonitoringConfiguration:
      logUri: S3 BUCKET
    cloudWatchMonitoringConfiguration:
      logGroupName: LOG GROUP NAME
      logStreamNamePrefix: LOG GROUP STREAM PREFIX
    sideCarResources:
      limits:
        cpuLimit: 500m
        memoryLimit: 250Mi
    containerLogRotationConfiguration:
        rotationSize: 2GB
        maxFilesToKeep: 10
```

次に、設定オプションを示します。

- `s3MonitoringConfiguration` — S3 への転送をセットアップするための設定キー
  - `logUri` (必須) – ログの保存先への S3 バケットパス。

- ログがアップロードされると、S3 上のパスは次のようになります。
  - ログローテーションは有効になっていません。

```
s3://${logUri}/${POD NAME}/STDOUT or STDERR.gz
```

  - ログローテーションは有効になっています。ローテーションするファイルと現在のファイル (日付スタンプがないファイル) の両方を使用できます。

```
s3://${logUri}/${POD NAME}/STDOUT or STDERR.gz
```

    次の形式にすると、数値が増分されます。

```
s3://${logUri}/${POD NAME}/stdout_YYYYMMDD_index.gz
```

- このフォワーダーを使用するには、次の IAM 権限が必要です。

```
{
    "Effect": "Allow",
    "Action": [
        "s3:PutObject"
    ],
    "Resource": [
       "S3_BUCKET_URI/*",
       "S3_BUCKET_URI"
    ]
}
```

- `cloudWatchMonitoringConfiguration` — CloudWatch への転送をセットアップするための設定キー。
  - `logGroupName` (必須) – ログの送信先となる CloudWatch ロググループの名前 (ロググループが存在しない場合は自動的に作成されます)。
  - `logStreamNamePrefix` (オプション) – ログの送信先となるログストリームの名前。デフォルト値は空の文字列です。形式は次のとおりです。

```
${logStreamNamePrefix}/${POD NAME}/STDOUT or STDERR
```

  - このフォワーダーを使用するには、次の IAM 権限が必要です。

```
{
```

```
    "Effect": "Allow",
    "Action": [
        "logs:CreateLogStream",
        "logs:CreateLogGroup",
        "logs:PutLogEvents"
    ],
    "Resource": [
        "arn:aws:logs:REGION:ACCOUNT-ID:log-group:{YOUR_LOG_GROUP_NAME}:*",
        "arn:aws:logs:REGION:ACCOUNT-ID:log-group:{YOUR_LOG_GROUP_NAME}"
    ]
}
```

- `sideCarResources` (オプション) – 起動した Fluentbit サイドカーコンテナにリソース制限を設定するための設定キー。
  - `memoryLimit` (オプション) - デフォルト値は 512Mi です。必要に応じて調整してください。
  - `cpuLimit` (オプション) — このオプションにはデフォルトはありません。必要に応じて調整してください。
- `containerLogRotationConfiguration` (オプション) – コンテナログローテーションの動作を制御します。このエージェントは、デフォルトでは有効になっています。
  - `rotationSize` (必須) - ログローテーションのファイルサイズを指定します。指定できる値の範囲は 2 KB ~ 2 GB です。rotationSize パラメータの数値単位部分は整数として渡されます。小数値はサポートされていないため、値 1500 MB などに 1.5 GB のローテーションサイズを指定できます。デフォルトは 2 GB です。
  - `maxFilesToKeep` (必須) — ローテーションが行われた後にコンテナに保持するファイルの最大数を指定します。最小値は 1 で、最大値は 50 です。デフォルトは 10 です。

Flink オペレータログ

また、オペレータのログアーカイブを有効にすることもできます。そのためには、Helm チャートのインストール先にある `values.yaml` ファイルで次のオプションを使用します。S3 と CloudWatch のいずれか、またはその両方を有効にできます。

```
monitoringConfiguration:
  s3MonitoringConfiguration:
    logUri: "S3-BUCKET"
    totalFileSize: "1G"
    uploadTimeout: "1m"
  cloudWatchMonitoringConfiguration:
    logGroupName: "flink-log-group"
```

```
    logStreamNamePrefix: "example-job-prefix-test-2"
  sideCarResources:
    limits:
      cpuLimit: 1
      memoryLimit: 800Mi
  memoryBufferLimit: 700M
```

次に、monitoringConfiguration で使用できる設定オプションを示します。

- s3MonitoringConfiguration — S3 にアーカイブするようにこのオプションを設定します。
- logUri (必須) – ログの保存先への S3 バケットパス。
- ログがアップロードされると、S3 バケットパスは次のような形式になります。
  - ログローテーションは有効になっていません。

    ```
    s3://${logUri}/${POD NAME}/OPERATOR or WEBHOOK/STDOUT or STDERR.gz
    ```

  - ログローテーションは有効になっています。ローテーションするファイルと現在のファイル (日付スタンプがないファイル) の両方を使用できます。

    ```
    s3://${logUri}/${POD NAME}/OPERATOR or WEBHOOK/STDOUT or STDERR.gz
    ```

    次の形式のインデックスにすると、数値が増分されます。

    ```
    s3://${logUri}/${POD NAME}/OPERATOR or WEBHOOK/stdout_YYYYMMDD_index.gz
    ```

- cloudWatchMonitoringConfiguration — CloudWatch への転送をセットアップするための設定キー。
  - logGroupName (必須) - ログの送信となる CloudWatch ロググループの名前。このグループが存在しない場合は自動的に作成されます。
  - logStreamNamePrefix (オプション) – ログの送信先となるログストリームの名前。デフォルト値は空の文字列です。CloudWatch の形式は次のとおりです。

    ```
    ${logStreamNamePrefix}/${POD NAME}/STDOUT or STDERR
    ```

- sideCarResources (オプション) – 起動した Fluentbit サイドカーコンテナにリソース制限を設定するための設定キー。
  - memoryLimit (オプション) — メモリの上限。必要に応じて調整してください。デフォルトは 512Mi です。

  - `cpuLimit` — CPU の制限。必要に応じて調整してください。デフォルト値はありません。
- `containerLogRotationConfiguration` (オプション) – コンテナログローテーションの動作を制御します。このエージェントは、デフォルトでは有効になっています。
  - `rotationSize` (必須) - ログローテーションのファイルサイズを指定します。指定できる値の範囲は 2 KB ~ 2 GB です。rotationSize パラメータの数値単位部分は整数として渡されます。小数値はサポートされていないため、値 1500 MB などに 1.5 GB のローテーションサイズを指定できます。デフォルトは 2 GB です。
  - `maxFilesToKeep` (必須) — ローテーションが行われた後にコンテナに保持するファイルの最大数を指定します。最小値は 1 で、最大値は 50 です。デフォルトは 10 です。

Flink が高可用性とジョブの耐障害性をサポートする方法

以下のセクションでは、Flink ジョブの信頼性を向上し、高可用性を実現する方法について説明します。これは、Flink の高可用性や障害発生時のさまざまな復旧機能などの組み込み機能によって実現されます。

トピック

Flink 演算子と Flink アプリケーションでの高可用性 (HA、High Availability) の使用

このトピックでは、高可用性を設定する方法を示し、いくつかの異なるユースケースでどのように機能するかについて説明します。これには、ジョブマネージャーを使用している場合や Flink ネイティブ kubernetes を使用している場合が含まれます。

Flink 演算子の高可用性

Flink 演算子の高可用性を有効にすることで、障害発生時にスタンバイの Flink 演算子にフェイルオーバーして、演算子制御ループのダウンタイムを最小限に抑えることができます。高可用性はデフォルトで有効になっており、デフォルトの開始演算子レプリカ数は 2 です。`values.yaml` ファイルのレプリカフィールドは Helm チャート用に設定できます。

以下のフィールドがカスタマイズ可能です。

- replicas (オプション、デフォルトは 2): この数を 1 より大きく設定すると、他のスタンバイ演算子が作成され、ジョブの復旧が早くなります。
- highAvailabilityEnabled (オプション、デフォルトは true): HA を有効にするかどうかを制御します。このパラメータを true に指定すると、マルチ AZ 配置サポートが有効になり、正しい flink-conf.yaml パラメータが設定されます。

values.yaml ファイルに以下の設定を行うことで、演算子の HA を無効にできます。

```
...
imagePullSecrets: []

replicas: 1

# set this to false if you don't want HA
highAvailabilityEnabled: false
...
```

マルチ AZ 配置

複数のアベイラビリティーゾーンに演算子ポッドを作成します。これはソフト制約であり、別の AZ に十分なリソースがない場合、演算子ポッドは同じ AZ にスケジュールされます。

リーダーレプリカの決定

HA が有効になっている場合、レプリカはリースを使用してどの JM がリーダーかを判断し、リーダーの選定には K8s リースを使用します。リースを記述し、.Spec.Holder Identity フィールドを確認することで、現在のリーダーを判断できます

```
kubectl describe lease <Helm Install Release Name>-<NAMESPACE>-lease -n <NAMESPACE> |
 grep "Holder Identity"
```

Flink-S3 インタラクション

アクセス認証情報の設定

S3 バケットにアクセスするための適切な IAM 権限を IRSA に設定していることを確認してください。

S3 アプリケーションモードからジョブジャーを取得

Flink 演算子は S3 からのアプリケーションジャーの取得もサポートしています。FlinkDeployment 仕様で jarURI の S3 ロケーションを指定するだけです。

この機能は、PyFlink スクリプトのような他のアーティファクトのダウンロードにも使用できます。結果として生じる Python スクリプトはパス `/opt/flink/usrlib/` の下にドロップされます。

以下の例は、PyFlink ジョブにこの機能を使用する方法を示しています。jarURI フィールドと args フィールドに注意してください。

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: python-example
spec:
  image: <YOUR CUSTOM PYFLINK IMAGE>
  emrReleaseLabel: "emr-6.12.0-flink-latest"
  flinkVersion: v1_16
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "1"
  serviceAccount: flink
  jobManager:
    highAvailabilityEnabled: false
    replicas: 1
    resource:
      memory: "2048m"
      cpu: 1
  taskManager:
    resource:
      memory: "2048m"
      cpu: 1
  job:
    jarURI: "s3://<S3-BUCKET>/scripts/pyflink.py" # Note, this will trigger the
 artifact download process
    entryClass: "org.apache.flink.client.python.PythonDriver"
    args: ["-pyclientexec", "/usr/local/bin/python3", "-py", "/opt/flink/usrlib/
pyflink.py"]
    parallelism: 1
    upgradeMode: stateless
```

Flink S3 コネクタ

Flink には 2 つの S3 コネクタ (下記参照) が同梱されています。以下のセクションでは、どのコネクタをどのような場合に使用するかについて説明します。

チェックポイント: Presto S3 コネクタ

- S3 スキームを s3p:// に設定します。
- S3 へのチェックポイントに使用することをお勧めするコネクタ。詳細については、「Apache Flink ドキュメント」の「S3 固有」を参照してください。

FlinkDeployment 仕様の例:

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: basic-example
spec:
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "2"
    state.checkpoints.dir: s3p://<BUCKET-NAME>/flink-checkpoint/
```

S3 への読み取りと書き込み: Hadoop S3 コネクタ

- S3 スキームを s3:// または (s3a://) に設定する
- S3 からのファイルの読み書きに推奨されるコネクタ (Flinks Filesystem インターフェイスを実装する S3 コネクタのみ)。
- デフォルトでは、fs.s3a.aws.credentials.provider が flink-conf.yaml ファイルに設定されます。このファイルは com.amazonaws.auth.WebIdentityTokenCredentialsProvider です。デフォルト flink-conf を完全にオーバーライドして S3 とやり取りする場合は、必ずこのプロバイダーを使用してください。

FlinkDeployment 仕様の例

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: basic-example
spec:
  job:
    jarURI: local:///opt/flink/examples/streaming/WordCount.jar
    args: [ "--input", "s3a://<INPUT BUCKET>/PATH", "--output", "s3a://<OUTPUT BUCKET>/
PATH" ]
```

```
    parallelism: 2
    upgradeMode: stateless
```

Flink ジョブマネージャー

Flink デプロイの高可用性 (HA、High Availability) により、一時的なエラーが発生して JobManager がクラッシュした場合でも、ジョブを継続して進行させることができます。ジョブは HA が有効になっている正常な最後のチェックポイントから再開されます。HA が有効になっていないと、Kubernetes は JobManager を再起動しますが、ジョブは新しいジョブとして開始され、進行状況は失われます。HA を設定したら、JobManager で一時的な障害が発生した場合に参照できるように HA メタデータを永続ストレージに保存し、正常な最後のチェックポイントからジョブを再開するように Kubernetes に指示できます。

Flink ジョブでは HA がデフォルトで有効になっています (レプリカ数は 2 に設定されているため、HA メタデータを保持するための S3 ストレージロケーションを指定する必要があります)。

HA 設定

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: basic-example
spec:
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "2"
  executionRoleArn: "<JOB EXECUTION ROLE ARN>"
  emrReleaseLabel: "emr-6.13.0-flink-latest"
  jobManager:
    resource:
      memory: "2048m"
      cpu: 1
    replicas: 2
    highAvailabilityEnabled: true
    storageDir: "s3://<S3 PERSISTENT STORAGE DIR>"
  taskManager:
    resource:
      memory: "2048m"
      cpu: 1
```

以下は、Job Manager (.spec.jobManager で定義されている) における上記の HA 設定の説明です。

- `highAvailabilityEnabled` (オプション、デフォルトは true): HA を有効にせず、提供されている HA 設定を使用しない場合は、これを `false` に設定してください。引き続き、「replicas」フィールドを操作して HA を手動で設定することはできます。
- `replicas` (オプション、デフォルトは 2): この数を 1 より大きく設定すると、他のスタンバイ JobManagers が作成され、ジョブのリカバリが速くなります。HA を無効にする場合は、レプリカ数を 1 に設定する必要があります。そうしないと、検証エラーが発生し続けます (HA が有効になっていない場合、サポートされるレプリカは 1 つのみです)。
- `storageDir` (必須): デフォルトではレプリカ数を 2 に設定しているため、永続的な storageDir を用意する必要があります。現在、このフィールドはストレージロケーションとして S3 パスのみを受け付けます。

ポッドのローカリティ

HA を有効にすると、同じ AZ 内のポッドのコロケーションも試みられるため、パフォーマンスが向上します (同じ AZ にポッドを配置することでネットワークレイテンシーが減少します)。これはベストエフォート型のプロセスです。つまり、ポッドの大部分がスケジュールされている AZ に十分なリソースがない場合でも、残りのポッドは引き続きスケジュールされますが、最終的にはこの AZ 外のノードに配置される可能性があります。

リーダーレプリカの決定

HA が有効になっている場合、レプリカはリースを使用してどの JM がリーダーかを判断し、このメタデータを保存するデータストアとして K8s Configmap を使用します。リーダーを特定する場合は、Configmap の内容とデータ内のキー `org.apache.flink.k8s.leader.restserver` を確認し、IP アドレスを持つ K8s ポッドを見つけてください。以下の bash コマンドを使用することもできます。

```
ip=$(kubectl get configmap -n <NAMESPACE> <JOB-NAME>-cluster-config-map -o json | jq -
r ".data[\"org.apache.flink.k8s.leader.restserver\"]" | awk -F: '{print $2}' | awk -F
 '/' '{print $3}')
kubectl get pods -n NAMESPACE  -o json | jq -r ".items[] | select(.status.podIP ==
 \"$ip\") | .metadata.name"
```

Flink ジョブ – ネイティブ Kubernetes

Amazon EMR 6.13.0 以降では、Amazon EKS クラスター上で高可用性モードで Flink アプリケーションを実行できるよう、Flink ネイティブの Kubernetes をサポートしています。

> **Note**
>
> Flink ジョブを送信する際に高可用性メタデータを保存するには、Amazon S3 バケットを作成する必要があります。この機能を使用しない場合には無効にできます。これは、デフォルトでは有効になっています。

Flink の高可用性機能を有効にするには、run-application CLI コマンドを実行するときに次の Flink パラメータを指定します。パラメータは例の下に定義されています。

```
-Dhigh-availability.type=kubernetes \
-Dhigh-availability.storageDir=S3://DOC-EXAMPLE-STORAGE-BUCKET \
-
Dfs.s3a.aws.credentials.provider="com.amazonaws.auth.WebIdentityTokenCredentialsProvider"
 \
-Dkubernetes.jobmanager.replicas=3 \
-Dkubernetes.cluster-id=example-cluster
```

- **Dhigh-availability.storageDir** – ジョブの高可用性メタデータを保存するための Amazon S3 バケット。

  Dkubernetes.jobmanager.replicas – 作成するジョブマネージャーポッドの数を、1 より大きい整数で指定します。

  Dkubernetes.cluster-id – Flink クラスターを識別する固有の ID です。

EKS 上の Amazon EMR によるタスクリカバリとスケーリング操作のための Flink ジョブの再起動時間の最適化

タスクが失敗したり、スケーリング操作が発生したりすると、Flink は最後に完了したチェックポイントからタスクを再実行しようとします。チェックポイントの状態のサイズと並列タスクの数によっては、再起動プロセスの実行に 1 分以上かかる場合があります。再起動中は、ジョブのバックログタスクが蓄積されることがあります。ただし、Flink が実行グラフの回復と再開の速度を最適化してジョブの安定性を高める方法はいくつかあります。

このページでは、Amazon EMR Flink がスポットインスタンスでタスクリカバリまたはスケーリング操作中のジョブの再起動時間を改善できるいくつかの方法について説明します。スポットインスタンスとは、割引価格で利用可能な未使用のコンピューティング能力のことです。これには、不定期に中

断するなど独自の動作があるため、EKS 上の Amazon EMR が廃止やジョブの再起動を実行する方法を含めて、Amazon EMR on EKS がこれらをどのように処理するかを理解することが重要です。

トピック

タスクローカルリカバリ

> **Note**
>
> EKS 6.14.0 以降では、Amazon EMR 上の Flink によるタスクローカルリカバリをサポートしています。

Flink チェックポイントでは、各タスクが Flink が Amazon S3 などの分散ストレージに書き込む状態のスナップショットを作成します。復旧時には、タスクは分散ストレージから状態を復元します。分散ストレージはすべてのノードからアクセスできるため、耐障害性があり、再スケーリング中に状態を再分散できます。

ただし、リモート分散ストアには欠点もあります。すべてのタスクはネットワーク経由でリモートロケーションから状態を読み取る必要があります。そのため、タスクの回復やスケーリング操作中に大きな状態の回復時間が長くなる可能性があります。

リカバリ時間が長いというこの問題は、タスクローカルリカバリによって解決できます。タスクはチェックポイントの状態を、ローカルディスクなど、タスクのローカルにあるセカンダリストレージに書き込みます。また、プライマリストレージ (この場合は Amazon S3) に状態が保存されます。復元中、スケジューラーは、タスクが以前に実行されていたのと同じタスクマネージャー上でタスクをスケジュールし、リモート状態ストアから読み取るのではなく、ローカル状態ストアから復元できるようにします。詳細については、「Apache Flink ドキュメント」の「タスクローカルリカバリ」を参照してください。

サンプルジョブを使ったベンチマークテストでは、タスクローカルリカバリを有効にすると、リカバリ時間が数分から数秒に短縮されたことがわかりました。

タスクローカルリカバリを有効にするには、flink-conf.yaml ファイルで次の設定を行います。チェックポイント間隔の値をミリ秒単位で指定します。

```
state.backend.local-recovery: true
state.backend: hasmap or rocksdb
state.checkpoints.dir: s3://STORAGE-BUCKET-PATH/checkpoint
execution.checkpointing.interval: 15000
```

Amazon EBS ボリュームマウントによるタスクローカルリカバリ

> **Note**
>
> EKS 6.15.0 以降では、Amazon EMR 上の Flink を用いた Amazon EBS でのタスクローカルリカバリをサポートしています。

EKS 上の Amazon EMR で Flink を使用すると、Amazon EBS ボリュームを TaskManager ポッドに自動的にプロビジョニングして、タスクローカルリカバリを行うことができます。デフォルトのオーバーレイマウントには 10 GB のボリュームが付属しており、状態の低いジョブには十分です。状態が大きいジョブでは、EBS ボリュームの自動マウントオプションを有効にできます。TaskManager ポッドはポッド作成時に自動的に作成およびマウントされ、ポッドの削除時に削除されます。

以下のステップを使用して、EKS 上の Amazon EMR 内の Flink 用に自動 EBS ボリュームマウントを有効にします。

1. 次のステップで使用する以下の変数の値をエクスポートします。

   ```
   export AWS_REGION=aa-example-1
   export FLINK_EKS_CLUSTER_NAME=my-cluster
   export AWS_ACCOUNT_ID=111122223333
   ```

2. クラスター用の kubeconfig YAML ファイルを作成または更新します。

   ```
   aws eks update-kubeconfig --name $FLINK_EKS_CLUSTER_NAME --region $AWS_REGION
   ```

3. お使いの Amazon EKS クラスター上の Amazon EBS Container Storage Interface (CSI) ドライバー 用 IAM サービスアカウントを作成します。

   ```
   eksctl create iamserviceaccount \
   ```

```
   --name ebs-csi-controller-sa \
   --namespace kube-system \
   --region $AWS_REGION \
   --cluster $FLINK_EKS_CLUSTER_NAME\
   --role-name TLR_${AWS_REGION}_${FLINK_EKS_CLUSTER_NAME} \
   --role-only \
   --attach-policy-arn arn:aws:iam::aws:policy/service-role/
AmazonEBSCSIDriverPolicy \
   --approve
```

4. 以下のコマンドで、Amazon EBS CSI ドライバーを作成します。

```
eksctl create addon \
   --name aws-ebs-csi-driver \
   --region $AWS_REGION \
   --cluster $FLINK_EKS_CLUSTER_NAME \
   --service-account-role-arn arn:aws:iam::${AWS_ACCOUNT_ID}:role/TLR_
${AWS_REGION}_${FLINK_EKS_CLUSTER_NAME}
```

5. 以下のコマンドで、Amazon EBS ストレージクラスを作成します。

```
cat # EOF # storage-class.yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ebs-sc
provisioner: ebs.csi.aws.com
volumeBindingMode: WaitForFirstConsumer
EOF
```

次に、クラスを適用します。

```
kubectl apply -f storage-class.yaml
```

6. サービスアカウントを作成するオプションを使用して、Amazon EMR Flink Kubernetes オペレーターを Helm インストールします。これにより、Flink デプロイで使用する `emr-containers-sa-flink` を作成します。

```
helm install flink-kubernetes-operator flink-kubernetes-operator/ \
   --set jobServiceAccount.create=true \
   --set rbac.jobRole.create=true \
   --set rbac.jobRoleBinding.create=true
```

7. Flink ジョブを送信してタスクローカルリカバリ用 EBS ボリュームの自動プロビジョニングを有効にするには、お使いの `flink-conf.yaml` ファイルに以下の構成を設定します。ジョブの状態サイズに合わせてサイズ制限を調整します。`serviceAccount` を `emr-containers-sa-flink` に設定します。チェックポイント間隔の値をミリ秒単位で指定します。`executionRoleArn` は省略してください。

```
flinkConfiguration:
    task.local-recovery.ebs.enable: true
    kubernetes.taskmanager.local-recovery.persistentVolumeClaim.sizeLimit: 10Gi
    state.checkpoints.dir: s3://BUCKET-PATH/checkpoint
    state.backend.local-recovery: true
    state.backend: hasmap or rocksdb
    state.backend.incremental: "true"
    execution.checkpointing.interval: 15000
  serviceAccount: emr-containers-sa-flink
```

Amazon EBS CSI ドライバープラグインを削除する準備ができたら、以下のコマンドを使用します。

```
  # Detach Attached Policy
  aws iam detach-role-policy --role-name TLR_${$AWS_REGION}_${FLINK_EKS_CLUSTER_NAME}
 --policy-arn arn:aws:iam::aws:policy/service-role/AmazonEBSCSIDriverPolicy
  # Delete the created Role
  aws iam delete-role --role-name TLR_${$AWS_REGION}_${FLINK_EKS_CLUSTER_NAME}
  # Delete the created service account
  eksctl delete iamserviceaccount --name ebs-csi-controller-sa --namespace kube-system
 --cluster $FLINK_EKS_CLUSTER_NAME --region $AWS_REGION
  # Delete Addon
  eksctl delete addon --name aws-ebs-csi-driver --cluster $FLINK_EKS_CLUSTER_NAME --
region $AWS_REGION
  # Delete the EBS storage class
  kubectl delete -f storage-class.yaml
```

汎用ログベースのインクリメンタルチェックポイント

> **Note**
>
> EKS 6.14.0 以降では、Amazon EMR 上の Flink による汎用ログベースのインクリメンタルチェックポイントをサポートしています。

チェックポイントの速度を向上させるため、汎用ログベースのインクリメンタルチェックポイントが Flink 1.16 に追加されました。チェックポイント間隔を短くすると、回復後に再処理する必要のあるイベントが少なくなるため、多くの場合回復作業が削減されます。詳細については、「Apache Flink ブログ」の「汎用ログベースのインクリメンタルチェックポイントによるチェックポイントの速度と安定性の向上」を参照してください。

サンプルジョブを使用したベンチマークテストでは、汎用ログベースのインクリメンタルチェックポイントを使用すると、チェックポイントにかかる時間が数分から数秒に短縮されたことがわかりました。

汎用ログベースのインクリメンタルチェックポイントを有効にするには、`flink-conf.yaml` ファイルで次の設定を行います。チェックポイント間隔の値をミリ秒単位で指定します。

```
state.backend.changelog.enabled: true
state.backend.changelog.storage: filesystem
dstl.dfs.base-path: s3://bucket-path/changelog
state.backend.local-recovery: true
state.backend: rocksdb
state.checkpoints.dir: s3://bucket-path/checkpoint
execution.checkpointing.interval: 15000
```

きめ細かなリカバリ

> **Note**
>
> EKS 6.14.0 以降では、Amazon EMR 上の Flink を用いたデフォルトスケジューラーでのきめ細かなリカバリをサポートしています。EKS 6.15.0 以降では、Amazon EMR 上の Flink を用いたアダプティブスケジューラー内のきめ細かなリカバリをサポートしています。

実行中にタスクが失敗した場合、Flink は実行グラフ全体をリセットし、最後に完了したチェックポイントから完全な再実行をトリガーします。これは、失敗したタスクを単に再実行するよりもコストがかかります。きめ細かい復元では、失敗したタスクのパイプラインに接続されたコンポーネントのみを再起動します。次の例では、ジョブグラフには 5 つの頂点 (A から E) があります。頂点間のすべての接続はポイントごとの分散でパイプライン化され、ジョブの `parallelism.default` は 2 に設定されます。

```
A # B # C # D # E
```

この例では、合計 10 個のタスクが実行されています。最初のパイプライン (a1 から e1) は TaskManager (TM1) で実行され、2 番目のパイプライン (a2 から e2) は別の TaskManager (TM2) 上で実行されます。

```
a1 # b1 # c1 # d1 # e1
a2 # b2 # c2 # d2 # e2
```

パイプライン接続されたコンポーネントには、a1 # e1 と a2 # e2 の 2 つがあります。TM1 または TM2 のどちらか一方で障害が発生しても、影響を受けるのは、その TaskManager が実行されていたパイプライン内の 5 つのタスクのみです。再起動戦略では、影響を受けるパイプラインコンポーネントのみを起動します。

きめ細かいリカバリは、完全に並列した Flink ジョブでのみ機能します。keyBy() または redistribute() オペレーションではサポートされていません。詳細については、「Flink 改善提案」Jira プロジェクトの「FLIP-1: タスク障害からのきめ細かな回復」を参照してください。

きめ細かい復元を有効にするには、flink-conf.yaml ファイルで次の設定を行います。

```
jobmanager.execution.failover-strategy: region
restart-strategy: exponential-delay or fixed-delay
```

アダプティブスケジューラーに組み込まれた再起動メカニズム

> **Note**
>
> EKS 6.15.0 以降では、Amazon EMR 上の Flink を用いたアダプティブスケジューラー内の複合再起動メカニズムをサポートしています。

アダプティブスケジューラーは、使用可能なスロットに基づいてジョブの並列処理を調整できます。設定したジョブの並列処理を満たすだけの十分なスロットがない場合は、自動的に並列処理を減らします 新しいスロットが使用可能になると、ジョブは設定されたジョブの並列処理に合わせて再びスケールアップされます。適応型スケジューラーは、利用可能なリソースが十分になくなってもジョブのダウンタイムを回避します。これは Flink Autoscaler でサポートされているスケジューラーです。これらの理由から、Amazon EMR Flink を使用するアダプティブスケジューラーをお勧めします。ただし、アダプティブスケジューラーは、新しいリソースが追加されるたびに 1 回再起動するなど、短期間に複数の再起動を行う場合があります。これにより、ジョブのパフォーマンスが低下する可能性があります。

Amazon EMR 6.15.0 以降では、Flink のアダプティブスケジューラーに再起動メカニズムが組み合わされています。このメカニズムは、最初のリソースが追加されると再起動ウィンドウを開き、設定したウィンドウ間隔 (デフォルトの 1 分) まで待機します。並列処理を設定してジョブを実行するのに十分なリソースがあるとき、または間隔がタイムアウトになったときに、1 回再起動します。

サンプルジョブを使用したベンチマークテストでは、アダプティブスケジューラーと Flink オートスケーラーを使用すると、この機能はデフォルトの動作よりも 10% 多くのレコードを処理することがわかりました。

複合再起動メカニズムを有効にするには、`flink-conf.yaml` ファイルで以下の設定を行います。

```
jobmanager.adaptive-scheduler.combined-restart.enabled: true
jobmanager.adaptive-scheduler.combined-restart.window-interval: 1m
```

EKS 上での Amazon EMR の Flink を用いたスポットインスタンスの適切な廃止

EKS 上の Amazon EMR で Flink を使用すると、タスクリカバリまたはスケーリング操作中のジョブの再起動時間を改善できます。

概要

EKS リリース 6.15.0 以降上の Amazon EMR では、EKS 上の Amazon EMR スポットインスタンスのタスクマネージャーの、Apache Flink を用いることによる適切な廃止をサポートしています。この機能の一部として、EKS 上の Flink を搭載した Amazon EMR には以下の機能が備わっています。

- ジャストインタイムチェックポイント - Flink のストリーミングジョブは、スポットインスタンスの中断に対応したり、実行中のジョブのジャストインタイム (JIT) チェックポイントを実行したり、これらのスポットインスタンスで追加のタスクをスケジュールしないようにできます。JIT チェックポイントは、デフォルトおよびアダプティブスケジューラーでサポートされています。
- 複合再起動メカニズム - 複合再起動メカニズムは、ターゲットリソースの並列処理、または現在設定されているウィンドウの終わりに到達した場合に、ベストエフォートベースでジョブの再起動を試みます。これにより、複数のスポットインスタンス終了によりジョブが連続して再起動されるのを防ぐこともできます。複合再起動メカニズムはアダプティブスケジューラーでのみ使用できます。

これらの機能には次のメリットがあります。

- スポットインスタンスを利用してタスクマネージャーを実行することで、クラスター費用を削減できます。
- スポットインスタンスタスクマネージャーの活性が向上することで耐障害性が強化され、ジョブスケジューリング効率が向上します。
- スポットインスタンス終了後の再起動回数が減るため、Flink ジョブのアップタイムが増えます。

適切な廃止の仕組み

以下の例を考えてみましょう。Apache Flink を実行している EKS クラスターに Amazon EMR をプロビジョニングし、ジョブマネージャーにはオンデマンドノードを、タスクマネージャーにはスポットインスタンスノードを指定するとします。終了の 2 分前に、タスクマネージャーは中断通知を受け取ります。

このシナリオでは、ジョブマネージャーはスポットインスタンスの中断信号を処理し、スポットインスタンス上の追加タスクのスケジューリングをブロックし、ストリーミングジョブの JIT チェックポイントを開始します。

次にジョブマネージャーは、現在の再起動間隔ウィンドウで現在のジョブの並列処理を満たす新しいリソースが十分に利用可能になった場合にのみ、ジョブグラフを再起動します。再起動ウィンドウ間隔は、スポットインスタンスの交換期間、新しいタスクマネージャーポッドの作成、およびジョブマネージャーへの登録に基づいて決定されます。

前提条件

適切な廃止を使用するには、Apache Flink を実行している EKS クラスター上の Amazon EMR にストリーミングジョブを作成して実行します。次の例のように、少なくとも 1 つのスポットインスタンスでスケジュールされたアダプティブスケジューラーとタスクマネージャーを有効にします。ジョブマネージャーにはオンデマンドノードを使用する必要があります。また、スポットインスタンスが少なくとも 1 つあれば、タスクマネージャーにもオンデマンドノードを使用できます。

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: deployment_name
spec:
  flinkVersion: v1_17
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "2"
    cluster.taskmanager.graceful-decommission.enabled: "true"
```

```
    execution.checkpointing.interval: "240s"
    jobmanager.adaptive-scheduler.combined-restart.enabled: "true"
    jobmanager.adaptive-scheduler.combined-restart.window-interval : "1m"
  serviceAccount: flink
  jobManager:
    resource:
      memory: "2048m"
      cpu: 1
    nodeSelector:
      'eks.amazonaws.com/capacityType': 'ON_DEMAND'
  taskManager:
    resource:
      memory: "2048m"
      cpu: 1
    nodeSelector:
      'eks.amazonaws.com/capacityType': 'SPOT'
  job:
    jarURI: flink_job_jar_path
```

設定

このセクションでは、廃止のニーズに合わせて指定できる大部分の構成について説明します。

キー	説明	デフォルト値	許容値
cluster.taskmanager.graceful-decommission.enabled	タスクマネージャーの適切な廃止を有効にします。	true	true, false
jobmanager.adaptive-scheduler.combined-restart.enabled	アダプティブスケジューラー内の複合、再起動メカニズムを有効にします。	false	true, false

キー	説明	デフォルト値	許容値
jobmanager.adaptive-scheduler.combined-restart.window-interval	ジョブのマージされた再起動を実行するための複合再起動ウィンドウ間隔。単位のない整数はミリ秒として解釈されます。	1m	例: 30、 60s、 3m、 1h

Flink アプリケーションでの Autoscaler の使用

演算子 autoscaler を使用すると、Flink ジョブからメトリクスを収集してジョブの頂点レベルで並列処理を自動的に調整することで、バックプレッシャーを緩和できます。次に、設定がどのようになるか、その一例を示します。

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  ...
spec:
  ...
  flinkVersion: v1_18
  flinkConfiguration:
    job.autoscaler.enabled: "true"
    job.autoscaler.stabilization.interval: 1m
    job.autoscaler.metrics.window: 5m
    job.autoscaler.target.utilization: "0.6"
    job.autoscaler.target.utilization.boundary: "0.2"
    job.autoscaler.restart.time: 2m
    job.autoscaler.catch-up.duration: 5m
    pipeline.max-parallelism: "720"
  ...
```

この設定では、Amazon EMR の最新リリースのデフォルト値を使用します。他のバージョンを使用する場合、値が異なる場合があります。

> **Note**
>
> Amazon EMR 7.2.0 以降、設定にプレフィックス kubernetes.operator を含める必要はありません。7.1.0 以前を使用する場合は、各設定の前にプレフィックスを使用する必要があります。例えば、kubernetes.operator.job.autoscaler.scaling.enabled と指定する必要があります。

次に、autoscaler の設定オプションを示します。

- job.autoscaler.scaling.enabled – autoscaler による頂点スケーリングの実行を有効にするかどうかを指定します。デフォルト: true。この設定を無効にすると、autoscaler はメトリクスのみを収集し、頂点ごとに推奨される並列処理を評価しますが、ジョブはアップグレードされません。
- job.autoscaler.stabilization.interval — 新しいスケーリングが実行されない安定化期間。デフォルトは 5 分です。
- job.autoscaler.metrics.window — スケーリングメトリクスの集計期間ウィンドウサイズ。ウィンドウサイズが大きいほど、スムーズで安定性が増しますが、負荷の突然の変化に対して autoscaler の対応が遅くなる可能性があります。デフォルトは 15 分です。3 ~ 60 分の値を試してみることをお勧めします。
- job.autoscaler.target.utilization — 頂点の目標使用率であり、ジョブのパフォーマンスを安定させ、負荷の変動をある程度和らげることができます。デフォルトは 0.7 で、ジョブ頂点の目標使用率/負荷は 70% です。
- job.autoscaler.target.utilization.boundary — 頂点の目標使用率の境界であり、負荷が変動してもすぐにはスケールしないよう、バッファとしての役割を果たします。デフォルトは 0.3 です。つまり、目標使用率からの偏差を 30% まで許容します。この値を超えると、スケーリングアクションをトリガーします。
- ob.autoscaler.restart.time — アプリケーションを再起動するまでの想定時間。デフォルトは 5 分です。
- job.autoscaler.catch-up.duration — キャッチアップするまでの想定時間。キャッチアップとは、スケーリング操作が完了した後でバックログを完全に処理することです。デフォルトは 5 分です。このキャッチアップ期間を短くすると、autoscaler がスケーリングアクションのために予約しなければならないキャパシティが増えます。
- pipeline.max-parallelism — autoscaler が使用できる最大並列処理。この値が Flink 設定か各オペレータに直接設定されている最大並列処理よりも高い場合、autoscaler はこの制限を無視し

ます。デフォルトは -1 です。autoscaler は、最大並列処理数の除数として並列処理を計算することに注意してください。このため、Flink が提供するデフォルトを利用するのではなく、除数が多い最大並列処理設定を選択することをお勧めします。この設定には、120、180、240、360、720 といった 60 の倍数を使用することをお勧めします。

詳細な設定リファレンスページについては、「Autoscaler configuration」を参照してください。

autoscaler パラメータの自動調整

このセクションでは、さまざまな Amazon EMR バージョンの自動調整の動作について説明します。また、さまざまな自動スケーリング設定についても詳しく説明します。

> **Note**
>
> Amazon EMR 7.2.0 以降では、オープンソース設定 `job.autoscaler.restart.time-tracking.enabled` を使用して、再スケール時間推定を有効にします。再スケール時間推定には Amazon EMR 自動調整と同じ機能があるため、再起動時間に経験値を手動で割り当てる必要はありません。
> Amazon EMR 7.1.0 以前を使用している場合でも、Amazon EMR 自動調整を使用できます。

7.2.0 and higher

Amazon EMR 7.2.0 以降では、自動スケーリングの決定を適用するために必要な実際の再起動時間を測定します。リリース 7.1.0 以前では、`job.autoscaler.restart.time` 設定を使用して推定最大再起動時間を手動で設定する必要がありました。設定 `job.autoscaler.restart.time-tracking.enabled` を使用すると、最初のスケーリングの再起動時間を入力するだけで済みます。その後、オペレータは実際の再起動時間を記録してその後のスケーリングに使用します。

この追跡を有効にするには、次のコマンドを使用します。

```
job.autoscaler.restart.time-tracking.enabled: true
```

以下は、再スケール時間推定の関連設定です。

設定	必須	デフォルト	説明
job.autoscaler.restart.time-tracking.enabled	いいえ	False	Flink Autoscaler が時間の経過とともに設定を自動的に調整してスケーリングの決定を最適化するかどうかを示します。Autoscaler は Autoscaler パラメータ `restart.time` のみを自動チューニングできることに注意してください。
job.autoscaler.restart.time	いいえ	5m	オペレータが以前のスケーリングから実際の再起動時間を決定するまで、Amazon EMR on EKS が使用する予想される再起動時間です。
job.autoscaler.restart.time-tracking.limit	いいえ	15m	`job.autoscaler.restart.time-tracking.enabled` が `true` に設定されている場合の最大観察再起動時間。

以下は、再スケール時間推定を試すために使用できるデプロイ仕様の例です。

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: autoscaling-example
spec:
  flinkVersion: v1_18
  flinkConfiguration:

    # Autoscaler parameters
    job.autoscaler.enabled: "true"
    job.autoscaler.scaling.enabled: "true"
    job.autoscaler.stabilization.interval: "5s"
    job.autoscaler.metrics.window: "1m"
```

```
    job.autoscaler.restart.time-tracking.enabled: "true"
    job.autoscaler.restart.time: "2m"
    job.autoscaler.restart.time-tracking.limit: "10m"

    jobmanager.scheduler: adaptive
    taskmanager.numberOfTaskSlots: "1"
    pipeline.max-parallelism: "12"

  executionRoleArn: <JOB ARN>
  emrReleaseLabel: emr-7.7.0-flink-latest
  jobManager:
    highAvailabilityEnabled: false
    storageDir: s3://<s3_bucket>/flink/autoscaling/ha/
    replicas: 1
    resource:
      memory: "1024m"
      cpu: 0.5
  taskManager:
    resource:
      memory: "1024m"
      cpu: 0.5
  job:
    jarURI: s3://<s3_bucket>/some-job-with-back-pressure
    parallelism: 1
    upgradeMode: stateless
```

バックプレッシャーをシミュレートするには、次のデプロイ仕様を使用します。

```
job:
    jarURI: s3://<s3_bucket>/pyflink-script.py
    entryClass: "org.apache.flink.client.python.PythonDriver"
    args: ["-py", "/opt/flink/usrlib/pyflink-script.py"]
    parallelism: 1
    upgradeMode: stateless
```

次の Python スクリプトを S3 バケットにアップロードします。

```
import logging
import sys
import time
import random
```

```
from pyflink.datastream import StreamExecutionEnvironment
from pyflink.table import StreamTableEnvironment

TABLE_NAME="orders"
QUERY=f"""
CREATE TABLE {TABLE_NAME} (
  id INT,
  order_time AS CURRENT_TIMESTAMP,
  WATERMARK FOR order_time AS order_time - INTERVAL '5' SECONDS
)
WITH (
  'connector' = 'datagen',
  'rows-per-second'='10',
  'fields.id.kind'='random',
  'fields.id.min'='1',
  'fields.id.max'='100'
);
"""

def create_backpressure(i):
    time.sleep(2)
    return i

def autoscaling_demo():
    env = StreamExecutionEnvironment.get_execution_environment()
    t_env = StreamTableEnvironment.create(env)
    t_env.execute_sql(QUERY)
    res_table = t_env.from_path(TABLE_NAME)

    stream =  t_env.to_data_stream(res_table) \
      .shuffle().map(lambda x: create_backpressure(x))\
      .print()
    env.execute("Autoscaling demo")

if __name__ == '__main__':
    logging.basicConfig(stream=sys.stdout, level=logging.INFO, format="%(message)s")
    autoscaling_demo()
```

再スケール時間推定が機能していることを確認するには、Flink オペレータの DEBUG レベルの口グ記録が有効になるようにします。以下の例は、Helm チャートファイル `values.yaml` を更新する方法を示しています。次に、更新された helm チャートを再インストールし、Flink ジョブを再度実行します。

```
log4j-operator.properties: |+
  # Flink Operator Logging Overrides
  rootLogger.level = DEBUG
```

リーダーポッドの名前を取得します。

```
ip=$(kubectl get configmap -n $NAMESPACE <job-name>-cluster-config-map -o json | jq
 -r ".data[\"org.apache.flink.k8s.leader.restserver\"]" | awk -F: '{print $2}' | awk
 -F '/' '{print $3}')

kubectl get pods -n $NAMESPACE -o json | jq -r ".items[] | select(.status.podIP ==
 \"$ip\") | .metadata.name"
```

次のコマンドを実行して、メトリクス評価で使用される実際の再起動時間を取得します。

```
kubectl logs <FLINK-OPERATOR-POD-NAME> -c flink-kubernetes-operator -n <OPERATOR-
NAMESPACE> -f | grep "Restart time used in scaling summary computation"
```

次のような ログが表示されます。最初のスケーリングのみが job.autoscaler.restart.time を使用することに注意してください。その後のスケーリングでは、観察された再起動時間が使用されます。

```
2024-05-16 17:17:32,590 o.a.f.a.ScalingExecutor        [DEBUG][default/autoscaler-
example] Restart time used in scaling summary computation: PT2M
2024-05-16 17:19:03,787 o.a.f.a.ScalingExecutor        [DEBUG][default/autoscaler-
example] Restart time used in scaling summary computation: PT14S
2024-05-16 17:19:18,976 o.a.f.a.ScalingExecutor        [DEBUG][default/autoscaler-
example] Restart time used in scaling summary computation: PT14S
2024-05-16 17:20:50,283 o.a.f.a.ScalingExecutor        [DEBUG][default/autoscaler-
example] Restart time used in scaling summary computation: PT14S
2024-05-16 17:22:21,691 o.a.f.a.ScalingExecutor        [DEBUG][default/autoscaler-
example] Restart time used in scaling summary computation: PT14S
```

7.0.0 and 7.1.0

オープンソースの組み込み Flink Autoscaler は、多くのメトリクスを使用して最適なスケーリング決定を行います。ただし、計算に使用するデフォルト値は、ほとんどのワークロードに適用できることを目的としており、特定のジョブには最適ではない場合があります。Flink オペレータの Amazon EMR on EKS バージョンに追加された自動調整機能は、特定のキャプチャされたメトリ

クスで観察された過去の傾向を調べ、それに応じて特定のジョブに合わせて調整された最適な値を計算しようとします。

設定	必須	デフォルト	説明
kubernetes.operator.job.autoscaler.autotune.enable	いいえ	False	Flink Autoscaler が時間の経過とともに設定を自動的に調整して autoscaler のスケーリングの決定を最適化するかどうかを示します。現在、Autoscaler は Autoscaler `restart.time` パラメータのみを自動チューニングできます。
kubernetes.operator.job.autoscaler.autotune.metrics.history.max.count	いいえ	3	Autoscaler が Amazon EMR on EKS メトリクス設定マップに保持する Amazon EMR on EKS メトリクスの履歴数を示します。
kubernetes.operator.job.autoscaler.autotune.metrics.restart.count	いいえ	3	Autoscaler が特定のジョブの平均再起動時間の計算を開始する前に実行する再起動の数を示します。

自動調整を有効にするには、以下を完了している必要があります。

- `kubernetes.operator.job.autoscaler.autotune.enable:` を `true` に設定します。
- `metrics.job.status.enable:` を `TOTAL_TIME` に設定します。
- Autoscaler for Flink アプリケーションの使用のセットアップに従って Autoscaling を有効にしている。

以下は、自動調整を試すために使用できるデプロイ仕様の例です。

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
```

```
metadata:
  name: autoscaling-example
spec:
  flinkVersion: v1_18
  flinkConfiguration:

    # Autotuning parameters
    kubernetes.operator.job.autoscaler.autotune.enable: "true"
    kubernetes.operator.job.autoscaler.autotune.metrics.history.max.count: "2"
    kubernetes.operator.job.autoscaler.autotune.metrics.restart.count: "1"
    metrics.job.status.enable: TOTAL_TIME

    # Autoscaler parameters
    kubernetes.operator.job.autoscaler.enabled: "true"
    kubernetes.operator.job.autoscaler.scaling.enabled: "true"
    kubernetes.operator.job.autoscaler.stabilization.interval: "5s"
    kubernetes.operator.job.autoscaler.metrics.window: "1m"

    jobmanager.scheduler: adaptive

    taskmanager.numberOfTaskSlots: "1"
    state.savepoints.dir: s3://<S3_bucket>/autoscaling/savepoint/
    state.checkpoints.dir: s3://<S3_bucket>/flink/autoscaling/checkpoint/
    pipeline.max-parallelism: "4"

  executionRoleArn: <JOB ARN>
  emrReleaseLabel: emr-6.14.0-flink-latest
  jobManager:
    highAvailabilityEnabled: true
    storageDir: s3://<S3_bucket>/flink/autoscaling/ha/
    replicas: 1
    resource:
      memory: "1024m"
      cpu: 0.5
  taskManager:
    resource:
      memory: "1024m"
      cpu: 0.5
  job:
    jarURI: s3://<S3_bucket>/some-job-with-back-pressure
    parallelism: 1
    upgradeMode: last-state
```

バックプレッシャーをシミュレートするには、次のデプロイ仕様を使用します。

```
  job:
    jarURI: s3://<S3_bucket>/pyflink-script.py
    entryClass: "org.apache.flink.client.python.PythonDriver"
    args: ["-py", "/opt/flink/usrlib/pyflink-script.py"]
    parallelism: 1
    upgradeMode: last-state
```

次の Python スクリプトを S3 バケットにアップロードします。

```
import logging
import sys
import time
import random

from pyflink.datastream import StreamExecutionEnvironment
from pyflink.table import StreamTableEnvironment

TABLE_NAME="orders"
QUERY=f"""
CREATE TABLE {TABLE_NAME} (
  id INT,
  order_time AS CURRENT_TIMESTAMP,
  WATERMARK FOR order_time AS order_time - INTERVAL '5' SECONDS
)
WITH (
  'connector' = 'datagen',
  'rows-per-second'='10',
  'fields.id.kind'='random',
  'fields.id.min'='1',
  'fields.id.max'='100'
);
"""

def create_backpressure(i):
    time.sleep(2)
    return i

def autoscaling_demo():
    env = StreamExecutionEnvironment.get_execution_environment()
    t_env = StreamTableEnvironment.create(env)
    t_env.execute_sql(QUERY)
```

```
    res_table = t_env.from_path(TABLE_NAME)

    stream =  t_env.to_data_stream(res_table) \
      .shuffle().map(lambda x: create_backpressure(x))\
      .print()
    env.execute("Autoscaling demo")

if __name__ == '__main__':
    logging.basicConfig(stream=sys.stdout, level=logging.INFO, format="%(message)s")
    autoscaling_demo()
```

自動チューナーが動作していることを確認するには、次のコマンドを使用します。Flink オペレータには独自のリーダーポッド情報を使用する必要があります。

まず、リーダーポッドの名前を取得します。

```
ip=$(kubectl get configmap -n $NAMESPACE <job-name>-cluster-config-map -o json | jq
 -r ".data[\"org.apache.flink.k8s.leader.restserver\"]" | awk -F: '{print $2}' | awk
 -F '/' '{print $3}')

kubectl get pods -n $NAMESPACE -o json | jq -r ".items[] | select(.status.podIP ==
 \"$ip\") | .metadata.name"
```

リーダーポッドの名前を取得したら、次のコマンドを実行できます。

```
kubectl logs -n $NAMESPACE  -c flink-kubernetes-operator --follow <YOUR-FLINK-
OPERATOR-POD-NAME>  | grep -E 'EmrEks|autotun|calculating|restart|autoscaler'
```

次のような口グが表示されます。

```
[m[33m2023-09-13 20:10:35,941[m [36mc.a.c.f.k.o.a.EmrEksMetricsAutotuner[m
 [36m[DEBUG][flink/autoscaling-example] Using the latest
 Emr Eks Metric for calculating restart.time for autotuning:
 EmrEksMetrics(restartMetric=RestartMetric(restartingTime=65, numRestarts=1))

[m[33m2023-09-13 20:10:35,941[m [36mc.a.c.f.k.o.a.EmrEksMetricsAutotuner[m
 [32m[INFO ][flink/autoscaling-example] Calculated average restart.time metric via
 autotuning to be: PT0.065S
```

Amazon EMR on EKS での Flink ジョブのメンテナンスとトラブルシューティング

以下のセクションでは、実行時間が長い Flink ジョブを維持する方法と、Flink ジョブのいくつかの一般的な問題のトラブルシューティング方法についてのガイダンスを提供します。

Flink アプリケーションのメンテナンス

トピック

Flink アプリケーションは通常、数週間、数か月、さらには数年といった長期間の実行を想定して設計されています。長時間実行されるすべてのサービスと同様に、Flink ストリーミングアプリケーションも保守する必要があります。これには、バグ修正、改善、および新しいバージョンの Flink クラスターへの移行が含まれます。

`FlinkDeployment` および `FlinkSessionJob` リソースの仕様が変更された場合は、実行中のアプリケーションをアップグレードする必要があります。そのためには、オペレーターは実行中のジョブを停止し (すでに中断されている場合を除く)、最新のスペックと、ステートフルアプリケーションの場合は前回の実行時の状態で再デプロイします。

ユーザーは、ステートフル アプリケーションの停止および復元時の状態の管理方法を、`JobSpec` の `upgradeMode` の設定で制御します。

アップグレードモード

必要に応じて導入

ステートレス

ステートレスアプリケーションは空の状態からアップグレードします。

最後の状態

アプリケーションの状態に関係なく (ジョブが失敗した場合でも) クイックアップグレードでは、常に最新の成功したチェックポイントを使用するため、正常なジョブは必要ありません。HA メタデータが失われた場合は、手動によるリカバリが必要な場合があります。最新のチェックポイントを取得する際にジョブがフォールバックする時間を制限するように `kubernetes.operator.job.upgrade.last-state.max.allowed.checkpoint.age` を

設定できます。チェックポイントが設定された値より古い場合は、ジョブを正常にするためにセーブポイントが使用されます。これはセッションモードではサポートされていません。

セーブポイント

アップグレードにセーブポイントを使うと、最大限の安全性とバックアップ/フォークポイントとしての役割を果たすことができます。セーブポイントは、アップグレード処理中に作成されます。セーブポイントを作成するには Flink ジョブが実行中である必要があることに注意してください。ジョブが異常な状態にある場合は、最後のチェックポイントが使用されます (kubernetes.operator.job.upgrade.last-state-fallback.enabled が false に設定されている場合を除く)。最後のチェックポイントが利用できない場合、ジョブのアップグレードは失敗します。

トラブルシューティング

このセクションでは、Amazon EMR on EKS に関する問題をトラブルシューティングする方法について説明します。Amazon EMR に関する一般的な問題をトラブルシューティングする方法については、「Amazon EMR 管理ガイド」の「Troubleshoot a cluster」を参照してください。

- PersistentVolumeClaims (PVC) を使用するジョブのトラブルシューティング
- Amazon EMR on EKS 垂直自動スケーリングのトラブルシューティング
- Amazon EMR on EKS Spark オペレータのトラブルシューティング

Amazon EMR on EKS での Apache Flink のトラブルシューティング

Helm チャートのインストール時にリソースマッピングが見つからない

Helm チャートをインストールするときに、次のエラーメッセージが表示されることがあります。

```
Error: INSTALLATION FAILED: pulling from host 1234567890.dkr.ecr.us-
west-2.amazonaws.com failed with status code [manifests 6.13.0]: 403 Forbidden Error:
 INSTALLATION FAILED: unable to build kubernetes objects from release manifest:
 [resource mapping not found for name: "flink-operator-serving-cert" namespace: "<the
 namespace to install your operator>" from "": no matches for kind "Certificate" in
 version "cert-manager.io/v1"

ensure CRDs are installed first, resource mapping not found for name: "flink-operator-
selfsigned-issuer" namespace: "<the namespace to install your operator>" " from "": no
 matches for kind "Issuer" in version "cert-manager.io/v1"
```

```
ensure CRDs are installed first].
```

このエラーを解決するには、cert-manager をインストールして、ウェブフックコンポーネントを追加できるようにします。使用する Amazon EKS クラスターごとに cert-manager をインストールする必要があります。

```
kubectl apply -f https://github.com/cert-manager/cert-manager/releases/download/v1.12.0
```

AWS のサービス アクセス拒否エラー

access denied エラーが表示された場合は、Helm チャート `values.yaml` ファイル内の `operatorExecutionRoleArn` の IAM ロールに適切な権限が付与されていることを確認します。また、`FlinkDeployment` 仕様の `executionRoleArn` に記載されている IAM ロールに適切な権限が付与されていることを確認します。

`FlinkDeployment` がスタックしている

`FlinkDeployment` がずっと停止した状態になる場合は、次の手順を使用して、問題のデプロイを強制的に削除します。

1. デプロイ実行を編集します。

   ```
   kubectl edit -n Flink Namespace flinkdeployments/App Name
   ```

2. このファイナライザーを削除します。

   ```
   finalizers:
     - flinkdeployments.flink.apache.org/finalizer
   ```

3. デプロイを削除します。

   ```
   kubectl delete -n Flink Namespace flinkdeployments/App Name
   ```

オプトインで Flink アプリケーションを実行するときの s3a AWSBadRequestException の問題 AWS リージョン

オプトイン AWS リージョンで Flink アプリケーションを実行すると、次のエラーが表示されることがあります。

```
Caused by: org.apache.hadoop.fs.s3a.AWSBadRequestException: getFileStatus on
```

```
s3://flink.txt: com.amazonaws.services.s3.model.AmazonS3Exception: Bad Request
(Service: Amazon S3; Status Code: 400; Error Code: 400 Bad Request; Request ID:
 ABCDEFGHIJKL; S3 Extended Request ID:
ABCDEFGHIJKLMNOP=; Proxy: null), S3 Extended Request ID: ABCDEFGHIJKLMNOP=:400 Bad
 Request: Bad Request
(Service: Amazon S3; Status Code: 400; Error Code: 400 Bad Request; Request ID:
 ABCDEFGHIJKL; S3 Extended Request ID: ABCDEFGHIJKLMNOP=; Proxy: null)
```

```
Caused by: org.apache.hadoop.fs.s3a.AWSBadRequestException: getS3Region on flink-
application: software.amazon.awssdk.services.s3.model.S3Exception: null
(Service: S3, Status Code: 400, Request ID: ABCDEFGHIJKLMNOP, Extended Request ID:
 ABCDEFGHIJKLMNOPQRST==):null: null
(Service: S3, Status Code: 400, Request ID: ABCDEFGHIJKLMNOP, Extended Request ID:
 AHl42uDNaTUFOus/5IIVNvSakBcMjMCH7dd37ky0vE6jhABCDEFGHIJKLMNOPQRST==)
```

これらのエラーを修正するには、FlinkDeployment 定義ファイルで次の設定を使用します。

```
spec:
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "2"
    fs.s3a.endpoint.region: OPT_IN_AWS_REGION_NAME
```

また、SDKv2 認証情報プロバイダーを使用することをお勧めします。

```
fs.s3a.aws.credentials.provider:
 software.amazon.awssdk.auth.credentials.WebIdentityTokenFileCredentialsProvider
```

SDKv1 認証情報プロバイダーを使用する場合は、SDK がオプトインリージョンをサポートしていることを確認してください。詳細については、「aws-sdk-java GitHub リポジトリ」を参照してください。

オプトインリージョンで Flink SQL ステートメントを実行するときに S3 AWSBadRequestException が表示される場合は、必ず flink 設定仕様で設定 fs.s3a.endpoint.region: *OPT_IN_AWS_REGION_NAME* を設定してください。

CN リージョンで Flink セッションジョブを実行するときの S3A AWSBadRequestException

Amazon EMR リリース 6.15.0 ~ 7.2.0 では、CN リージョンで Flink セッションジョブを実行すると、次のエラーメッセージが表示されることがあります。これには中国 (北京) および中国 (寧夏) が含まれます。

```
Error:
 {"type":"org.apache.flink.kubernetes.operator.exception.ReconciliationException","message":"or
                    getFileStatus on s3://ABCDPath:
 software.amazon.awssdk.services.s3.model.S3Exception: null (Service: S3, Status Code:
 400, Request ID: ABCDEFGH, Extended Request ID:
                    ABCDEFGH:null: null (Service: S3, Status Code: 400, Request ID:
 ABCDEFGH, Extended Request ID: ABCDEFGH","additionalMetadata":{},"throwableList":

 [{"type":"org.apache.hadoop.fs.s3a.AWSBadRequestException","message":"getFileStatus on
 s3://ABCDPath: software.amazon.awssdk.services.s3.model.S3Exception:
                    null (Service: S3, Status Code: 400, Request ID: ABCDEFGH, Extended
 Request ID: ABCDEFGH:null: null (Service: S3, Status Code: 400, Request ID: ABCDEFGH,
                    Extended Request ID: ABCDEFGH","additionalMetadata":{}},
{"type":"software.amazon.awssdk.services.s3.model.S3Exception","message":"null
 (Service: S3, Status Code: 400,
                    Request ID: ABCDEFGH, Extended Request ID:
 ABCDEFGH","additionalMetadata":{}}]}
```

この問題は認識されています。チームは、これらのすべてのリリースバージョンで flink オペレータのパッチ適用に取り組んでいます。ただし、パッチが完了する前にこのエラーを修正する場合は、flink オペレータの helm チャートをダウンロードして解凍 (圧縮されたファイルを抽出) し、helm チャートの設定を変更する必要があります。

具体的な手順は次のとおりです。

1. 特にディレクトリを helm チャートのローカルフォルダに変更し、次のコマンドラインを実行して helm チャートをプルして解凍 (抽出) します。

   ```
   helm pull oci://public.ecr.aws/emr-on-eks/flink-kubernetes-operator \
   --version $VERSION \
   --namespace $NAMESPACE
   ```

   ```
   tar -zxvf flink-kubernetes-operator-$VERSION.tgz
   ```

2. helm チャートフォルダに移動し、`templates/flink-operator.yaml` ファイルを見つけます。
3. ConfigMap `flink-operator-config` を検索し、次の `fs.s3a.endpoint.region` 設定を `flink-conf.yaml` に追加します。以下に例を示します。

   ```
   {{- if .Values.defaultConfiguration.create }}
   ```

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: flink-operator-config
  namespace: {{ .Release.Namespace }}
  labels:
    {{- include "flink-operator.labels" . | nindent 4 }}
data:
  flink-conf.yaml: |+
fs.s3a.endpoint.region: {{ .Values.emrContainers.awsRegion }}
```

4. ローカル helm チャートをインストールし、ジョブを実行します。

Apache Flink をサポートしている Amazon EMR on EKS のリリース

Apache Flink は、次の Amazon EMR on EKS リリースで使用できます。使用可能なすべてのリリースについては、「Amazon EMR on EKS リリース」を参照してください。

リリースラベル	Java	Flink	Flink オペレータ
emr-7.2.0-flink-latest	17	1.18.1	-
emr-7.2.0-flink-k8s-operator-latest	11	-	1.8.0
emr-7.1.0-flink-latest	17	1.18.1	-
emr-7.1.0-flink-k8s-operator-latest	11	-	1.6.1
emr-7.0.0-flink-latest	11	1.18.0	-
emr-7.0.0-flink-k8s-operator-latest	11	-	1.6.1
emr-6.15.0-flink-latest	11	1.17.1	-
emr-6.15.0-flink-k8s-operator-latest	11	-	1.6.0
emr-6.14.0-flink-latest	11	1.17.1	-
emr-6.14.0-flink-k8s-operator-latest	11	-	1.6.0

リリースラベル	Java	Flink	Flink オペレータ
emr-6.13.0-flink-latest	11	1.17.0	-
emr-6.13.0-flink-k8s-operator-latest	11	-	1.5.0

Amazon EMR on EKS での Spark ジョブの実行

ジョブ実行とは、Amazon EMR on EKS に送信する Spark jar、PySpark スクリプト、SparkSQL クエリなどの作業単位です。このトピックでは、 を使用したジョブ実行の管理 AWS CLI、Amazon EMR コンソールを使用したジョブ実行の表示、一般的なジョブ実行エラーのトラブルシューティングの概要について説明します。

Amazon EMR on EKS では IPv6 Spark ジョブを実行できないことに注意してください。

> **Note**
>
> Amazon EMR on EKS でのジョブ実行を送信する前に、「Amazon EMR on EKS のセットアップ」のステップを完了する必要があります。

トピック

`StartJobRun` で Spark ジョブを実行する

このセクションでは、Spark ジョブを実行する環境を整えるための詳細なセットアップ手順と、指定されたパラメータでジョブ実行を送信するためのステップバイステップの手順について説明します。

トピック

- Amazon EMR on EKS のセットアップ
- StartJobRun でジョブ実行を送信する
- ジョブ送信者分類の使用
- Amazon EMR コンテナのデフォルト分類の使用

Amazon EMR on EKS のセットアップ

Amazon EMR on EKS のセットアップを行うには、以下のタスクを完了します。Amazon Web Services (AWS) に既にサインアップしていて、Amazon EKS を既に使用している場合、Amazon EMR on EKS を使用する準備はほぼ整っています。既に完了しているタスクはスキップしてください。

> **Note**
>
> また、Amazon EMR on EKS Workshop に従って、Amazon EMR on EKS で Spark ジョブを実行するのに必要なすべてのリソースをセットアップすることもできます。このワークショップには、CloudFormation テンプレートを使用して、開始するのに必要なリソースを作成する自動化機能も用意されています。その他のテンプレートとベストプラクティスについては、GitHub の「EMR Containers Best Practices Guide」を参照してください。

1. の最新バージョンをインストールまたは更新する AWS CLI
2. kubectl と eksctl のセットアップ
3. Amazon EKS – eksctl の使用開始
4. Amazon EMR on EKS のクラスターアクセスを有効にする
5. EKS クラスターでサービスアカウント (IRSA) の IAM ロールを有効にする
6. ジョブ実行ロールを作成する
7. ジョブ実行ロールの信頼ポリシーを更新する
8. Amazon EMR on EKS へのアクセス許可をユーザーに付与する
9. Amazon EKS クラスターをAmazon EMR に登録する

Amazon EMR on EKS のクラスターアクセスを有効にする

以下の各セクションでは、クラスターアクセスを有効にする方法をいくつか示します。1 つ目の方法は、Amazon EKS クラスターアクセス管理 (CAM) を使用することによるものです。もう 1 つの方法は、クラスターアクセスを有効にするための手動手順を実施する方法を示すものです。

EKS アクセスエントリを使用してクラスターアクセスを有効にする (推奨)

> **Note**
>
> `aws-auth` ConfigMap は非推奨です。Kubernetes API へのアクセスを管理する際は、アクセスエントリを使用することをお勧めします。

Amazon EMR は Amazon EKS クラスターアクセス管理 (CAM) と統合されているため、Amazon EKS クラスターの名前空間で Amazon EMR Spark ジョブを実行するために必要な AuthN ポリシーと AuthZ ポリシーの設定を自動化できます。Amazon EKS クラスター名前空間から仮想クラスターを作成すると、Amazon EMR は必要なすべてのアクセス許可を自動的に設定するため、現在のワークフローにその他の手順を追加する必要はありません。

> **Note**
>
> Amazon EMR と Amazon EKS CAM の統合は、新しい Amazon EMR on EKS 仮想クラスターでのみサポートされています。既存の仮想クラスターを移行してこの統合を使用することはできません。

前提条件

- のバージョン 2.15.3 以降を実行していることを確認します。 AWS CLI
- Amazon EKS クラスターでは、1.23 以降のバージョンを使用している必要があります。

セットアップ

Amazon EMR と Amazon EKS の AccessEntry API オペレーションの統合を設定するには、次の項目を完了していることを確認してください。

- Amazon EKS クラスターの `authenticationMode` が `API_AND_CONFIG_MAP` に設定されていることを確認します。

```
aws eks describe-cluster --name <eks-cluster-name>
```

まだ設定されていない場合は、authenticationMode を API_AND_CONFIG_MAP に設定します。

```
aws eks update-cluster-config
    --name <eks-cluster-name>
    --access-config authenticationMode=API_AND_CONFIG_MAP
```

認証モードの詳細については、「クラスター認証モード」を参照してください。

- CreateVirtualCluster および DeleteVirtualCluster の API オペレーションの実行に使用する IAM ロールにも、次のアクセス許可があることを確認します。

```
{
  "Effect": "Allow",
  "Action": [
    "eks:CreateAccessEntry"
  ],
  "Resource":
 "arn:<AWS_PARTITION>:eks:<AWS_REGION>:<AWS_ACCOUNT_ID>:cluster/<EKS_CLUSTER_NAME>"
},
{
  "Effect": "Allow",
  "Action": [
    "eks:DescribeAccessEntry",
    "eks:DeleteAccessEntry",
    "eks:ListAssociatedAccessPolicies",
    "eks:AssociateAccessPolicy",
    "eks:DisassociateAccessPolicy"
  ],
  "Resource": "arn:<AWS_PARTITION>:eks:<AWS_REGION>:<AWS_ACCOUNT_ID>:access-entry/
<EKS_CLUSTER_NAME>/role/<AWS_ACCOUNT_ID>/AWSServiceRoleForAmazonEMRContainers/*"
}
```

概念と用語

以下に示すのは、Amazon EKS CAM に関連する用語と概念のリストです。

- 仮想クラスター (VC) – Amazon EKS で作成された名前空間の論理表現。これは、Amazon EKS クラスター名前空間への 1:1 リンクです。これを使用して、指定された名前空間内の Amazon EKS クラスターで Amazon EMR ワークロードを実行できます。
- 名前空間 – 単一の EKS クラスター内のリソースグループを分離するためのメカニズム。
- アクセスポリシー – EKS クラスター内の IAM ロールにアクセス権とアクションを付与するアクセス許可。
- アクセスエントリ – ロール arn で作成されたエントリ。アクセスエントリをアクセスポリシーにリンクして、Amazon EKS クラスターに特定のアクセス許可を割り当てることができます。
- EKS アクセスエントリ統合仮想クラスター – Amazon EKS のアクセスエントリ API オペレーションを使用して作成された仮想クラスター。

`aws-auth` を使用してクラスターアクセスを有効にする

Amazon EMR on EKS がクラスターで特定の名前空間にアクセスできるようにする必要があります。これを行うには、Kubernetes ロールを作成し、そのロールを Kubernetes ユーザーにバインドして、Kubernetes ユーザーをサービスにリンクされたロール `AWSServiceRoleForAmazonEMRContainers` にマッピングします。これらのアクションは、IAM ID マッピングコマンドがサービス名として `emr-containers` と共に使用されると、`eksctl` で自動的に行われます。これらの操作は、次のコマンドを使用して簡単に実行できます。

```
eksctl create iamidentitymapping \
    --cluster my_eks_cluster \
    --namespace kubernetes_namespace \
    --service-name "emr-containers"
```

`my_eks_cluster` は使用する Amazon EKS クラスターの名前に置き換え、*`kubernetes_namespace`* は Amazon EMR ワークロードを実行するために作成された Kubernetes 名前空間に置き換えます。

> **Important**
>
> この機能を使用するには、前のステップである kubectl と eksctl の設定を使用して最新の eksctl をダウンロードする必要があります。

Amazon EMR on EKS のクラスターアクセスを有効にするための手動ステップ

次の手動ステップを使用して、Amazon EMR on EKS のクラスターアクセスを有効にすることもできます。

1. 特定の名前空間に Kubernetes ロールを作成する

   Amazon EKS 1.22 - 1.29

   Amazon EKS 1.22 ~ 1.29 で、次のコマンドを実行して特定の名前空間に Kubernetes ロールを作成します。このロールは、Amazon EMR on EKS に必要な RBAC アクセス許可を付与します。

```
namespace=my-namespace
cat - >>EOF | kubectl apply -f - >>namespace "${namespace}"
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: emr-containers
  namespace: ${namespace}
rules:
  - apiGroups: [""]
    resources: ["namespaces"]
    verbs: ["get"]
  - apiGroups: [""]
    resources: ["serviceaccounts", "services", "configmaps", "events", "pods",
 "pods/log"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
 "deletecollection", "annotate", "patch", "label"]
  - apiGroups: [""]
    resources: ["secrets"]
    verbs: ["create", "patch", "delete", "watch"]
  - apiGroups: ["apps"]
    resources: ["statefulsets", "deployments"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
 "annotate", "patch", "label"]
  - apiGroups: ["batch"]
    resources: ["jobs"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
 "annotate", "patch", "label"]
  - apiGroups: ["extensions", "networking.k8s.io"]
    resources: ["ingresses"]
```

```
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
 "annotate", "patch", "label"]
  - apiGroups: ["rbac.authorization.k8s.io"]
    resources: ["roles", "rolebindings"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
 "deletecollection", "annotate", "patch", "label"]
  - apiGroups: [""]
    resources: ["persistentvolumeclaims"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
 "deletecollection", "annotate", "patch", "label"]
EOF
```

Amazon EKS 1.21 and below

Amazon EKS 1.21 以前で、次のコマンドを実行して特定の名前空間に Kubernetes ロールを作成します。このロールは、Amazon EMR on EKS に必要な RBAC アクセス許可を付与します。

```
namespace=my-namespace
cat - >>EOF | kubectl apply -f - >>namespace "${namespace}"
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: emr-containers
  namespace: ${namespace}
rules:
  - apiGroups: [""]
    resources: ["namespaces"]
    verbs: ["get"]
  - apiGroups: [""]
    resources: ["serviceaccounts", "services", "configmaps", "events", "pods",
 "pods/log"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
 "deletecollection", "annotate", "patch", "label"]
  - apiGroups: [""]
    resources: ["secrets"]
    verbs: ["create", "patch", "delete", "watch"]
  - apiGroups: ["apps"]
    resources: ["statefulsets", "deployments"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
 "annotate", "patch", "label"]
  - apiGroups: ["batch"]
```

```
    resources: ["jobs"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
 "annotate", "patch", "label"]
  - apiGroups: ["extensions"]
    resources: ["ingresses"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
 "annotate", "patch", "label"]
  - apiGroups: ["rbac.authorization.k8s.io"]
    resources: ["roles", "rolebindings"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
 "deletecollection", "annotate", "patch", "label"]
  - apiGroups: [""]
    resources: ["persistentvolumeclaims"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
 "deletecollection", "annotate", "patch", "label"]
EOF
```

2. 名前空間にスコープが設定された Kubernetes ロールバインディングを作成する

次のコマンドを実行して、特定の名前空間に Kubernetes ロールバインディングを作成します。このロールバインディングは、前のステップで作成したロールに定義されたアクセス許可を、emr-containers という名前のユーザーに付与します。このユーザーにより、Amazon EMR on EKS のサービスにリンクされたロールが特定されるため、Amazon EMR on EKS は作成したロールによって定義されているとおりにアクションを実行できます。

```
namespace=my-namespace

cat - <<EOF | kubectl apply -f - --namespace "${namespace}"
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: emr-containers
  namespace: ${namespace}
subjects:
- kind: User
  name: emr-containers
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: emr-containers
  apiGroup: rbac.authorization.k8s.io
```

```
EOF
```

3. Kubernetes **aws-auth** 設定マップを更新する

   次のいずれかのオプションを使用して、Amazon EMR on EKS のサービスにリンクされたロールを、前のステップで Kubernetes ロールにバインドされた emr-containers ユーザーにマップできます。

   オプション 1: **eksctl** を使用する

   次の eksctl コマンドを実行して、Amazon EMR on EKS のサービスにリンクされたロールを emr-containers ユーザーにマップします。

```
eksctl create iamidentitymapping \
    --cluster my-cluster-name \
    --arn "arn:aws:iam::my-account-id:role/AWSServiceRoleForAmazonEMRContainers" \
    --username emr-containers
```

   オプション 2: eksctl を使用しない

   1. 次のコマンドを実行して、テキストエディタで aws-auth 設定マップを開きます。

```
kubectl edit -n kube-system configmap/aws-auth
```

      > **Note**
      >
      > Error from server (NotFound): configmaps "aws-auth" not found というエラーを受け取った場合、「Amazon EKS ユーザーガイド」のユーザーロールの追加の手順を参照して、ストック ConfigMap を適用してください。

   2. Amazon EMR on EKS のサービスにリンクされたロールの詳細を、data の下の ConfigMap の mapRoles セクションに追加します。ファイルに存在しない場合は、このセクションを追加します。データの下で更新された mapRoles セクションは、次の例のようになります。

```
apiVersion: v1
data:
  mapRoles: |
    - rolearn: arn:aws:iam::<your-account-id>:role/
AWSServiceRoleForAmazonEMRContainers
      username: emr-containers
```

```
    - ... <other previously existing role entries, if there's any>.
```

3. ファイルを保存し、テキストエディタを終了します。

EKS クラスターでサービスアカウント (IRSA) の IAM ロールを有効にする

サービスアカウントの IAM ロール機能は、Amazon EKS バージョン 1.14 以降、および 2019 年 9 月 3 日以降にバージョン1.13 に更新された EKS クラスターで利用できます。この機能を使用するために、既存の EKS クラスターをバージョン 1.14 以降に更新できます。詳細については、「Amazon EKS クラスターの Kubernetes バージョンの更新」を参照してください。

クラスターがサービスアカウントの IAM ロールをサポートしている場合、OpenID Connect 発行者 URL が関連付けられます。この URL は Amazon EKS コンソールで表示することも、次の AWS CLI コマンドを使用して取得することもできます。

> **Important**
>
> このコマンドから適切な出力を受け取るには AWS CLI 、 の最新バージョンを使用する必要があります。

```
aws eks describe-cluster --name cluster_name --query "cluster.identity.oidc.issuer" --
output text
```

予想される出力は次のようになります。

```
https://oidc.eks.<region-code>.amazonaws.com/id/EXAMPLED539D4633E53DE1B716D3041E
```

クラスターでサービスアカウントの IAM ロールを使用するには、eksctl または AWS Management Console のいずれかを使用して OIDC ID プロバイダーを作成する必要があります。

`eksctl` を使用してクラスターの IAM OIDC ID プロバイダーを作成するには

以下のコマンドを使用して、`eksctl` のバージョンを確認します。この手順では、`eksctl` をインストール済みで、お使いの `eksctl` のバージョンが 0.32.0 以上であることを前提としています。

```
eksctl version
```

eksctl のインストールまたはアップグレードの詳細については、「eksctl のインストールまたはアップグレード」を参照してください。

次のコマンドを使用して、クラスターの OIDC ID プロバイダーを作成します。*cluster_name* は、独自の値に置き換えてください。

```
eksctl utils associate-iam-oidc-provider --cluster cluster_name --approve
```

を使用してクラスターの IAM OIDC ID プロバイダーを作成するには AWS Management Console

クラスターの Amazon EKS コンソールの説明から OIDC 発行者 URL を取得するか、次の AWS CLI コマンドを使用します。

次のコマンドを使って、 AWS CLIから OIDC 発行者 URL を取得します。

```
aws eks describe-cluster --name <cluster_name> --query "cluster.identity.oidc.issuer"
 --output text
```

次の手順に従って、Amazon EKS コンソールから OIDC 発行者 URL を取得します。

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで、[ID プロバイダー] を選択し、[プロバイダーの作成] をクリックします。
    1. [プロバイダーのタイプ] で [Choose a provider type] を選択してから、[OpenID Connect] を選択します。
    2. Provider URL に、クラスターの OIDC 発行者 URL を貼り付けます。
    3. [対象者] に、「sts.amazonaws.com」と入力し、[次のステップ] を選択します。
3. プロバイダー情報が正しいことを確認し、[作成] を選択して ID プロバイダーを作成します。

ジョブ実行ロールを作成する

Amazon EMR on EKS でワークロードを実行するには、IAM ロールを作成する必要があります。このドキュメントでは、このロールをジョブ実行ロールと呼びます。IAM ロールの作成方法の詳細については、「IAM ユーザーガイド」の「IAM ロールの作成」を参照してください。

また、ジョブ実行ロールの権限を指定する IAM ポリシーを作成し、その IAM ポリシーをジョブ実行ロールにアタッチする必要があります。

ジョブ実行ロールの次のポリシーでは、リソースターゲット、Amazon S3、および CloudWatch へのアクセスが許可されます。これらのアクセス許可は、ジョブとアクセスログを監視するために必要です。を使用して同じプロセスを実行するには AWS CLI、「Amazon EMR on EKS Workshop」の「Create IAM Role for job execution」セクションの手順を使用してロールを設定することもできます。

> **Note**
>
> アクセス権限は、ジョブ実行ロール内のすべての S3 オブジェクトに付与するのではなく、適切にスコープする必要があります。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents",
                "logs:CreateLogStream",
                "logs:DescribeLogGroups",
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:*:*:*"
            ]
        }
    ]
}
```

詳細については、ジョブ実行ロールの使用、S3ログ使用のためのジョブ実行の構成、およびCloudWatchログ使用のためのジョブ実行の構成 ）を参照してください。

ジョブ実行ロールの信頼ポリシーを更新する

サービスアカウントの IAM ロール (IRSA) を使用して Kubernetes 名前空間上でジョブを実行する場合、管理者はジョブ実行ロールと EMR マネージドサービスアカウントの ID との間に信頼関係を作成する必要があります。信頼関係は、ジョブ実行ロールの信頼ポリシーを更新することによって作成できます。EMR マネージドサービスアカウントは、ジョブの送信時に自動的に作成され、ジョブが送信される名前空間にスコープ設定されます。

信頼ポリシーを更新するには、次のコマンドを実行します。

```
aws emr-containers update-role-trust-policy \
      --cluster-name cluster \
      --namespace namespace \
      --role-name iam_role_name_for_job_execution
```

詳細については、「Amazon EMR on EKS でのジョブ実行ロールの使用」を参照してください。

> **Important**
>
> 上記のコマンドを実行するオペレータには、eks:DescribeCluster、iam:GetRole、iam:UpdateAssumeRolePolicy のアクセス許可が必要です。

Amazon EMR on EKS へのアクセス許可をユーザーに付与する

Amazon EMR on EKS で実行するアクションについては、そのアクションに対応する IAM アクセス許可が必要です。Amazon EMR on EKS アクションを実行し、使用する IAM ユーザーまたはロールにポリシーをアタッチできる IAM ポリシーを作成する必要があります。

このトピックでは、新しいポリシーを作成し、ユーザーにアタッチする手順について説明します。Amazon EMR on EKS 環境を設定するために必要な基本的なアクセス許可についても説明します。ビジネスニーズに基づいて、可能な限り、特定のリソースに対するアクセス許可を絞り込むことをお勧めします。

新しい IAM ポリシーを作成して、IAM コンソールでユーザーにアタッチする

新規 IAM ポリシーを作成する

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/iam/ で IAM コンソールを開きます。
2. IAM コンソールの左側のナビゲーションペインで [ポリシー] を選択します。
3. [ポリシー] ページで、[ポリシーの作成] を選択します。
4. [Create Policy] (ポリシーの作成) ウィンドウで、[Edit JSON] (JSON の編集) タブに移動します。この手順の後、例に示されているように、1 つ以上の JSON ステートメントを含むポリシードキュメントを作成します。次に、[ポリシーの確認] を選択します。
5. [Review Policy] (ポリシーの確認) 画面で、[Policy Name] (ポリシー名)に `AmazonEMROnEKSPolicy` (など)を入力します。任意で説明を入力し、[ポリシーの作成] を選択します。

ポリシーをユーザーまたはロールにアタッチする

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/iam/ で IAM コンソールを開きます。
2. ナビゲーションペインで、ポリシー を選択してください。
3. ポリシーのリストで、前のセクションで作成したポリシーの横にあるチェックボックスを選択します。[Filter (フィルター)] メニューと検索ボックスを使用して、ポリシーのリストをフィルタリングできます。
4. [Policy actions] を選択して、[Attach] を選択します。
5. ポリシーをアタッチするユーザーまたはロールを選択します。[Filter] メニューと検索ボックスを使用して、プリンシパルエンティティのリストをフィルタリングできます。ポリシーをアタッチするユーザーまたはロールを選択した後、[Attach policy] (ポリシーのアタッチ) を選択します。

仮想クラスターを管理するためのアクセス許可

AWS アカウントの仮想クラスターを管理するには、次のアクセス許可を持つ IAM ポリシーを作成します。これらのアクセス許可により、 AWS アカウントで仮想クラスターを作成、一覧表示、説明、削除できます。

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreateServiceLinkedRole"
            ],
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": "emr-containers.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "emr-containers:CreateVirtualCluster",
                "emr-containers:ListVirtualClusters",
                "emr-containers:DescribeVirtualCluster",
                "emr-containers:DeleteVirtualCluster"
            ],
            "Resource": "*"
        }
    ]
}
```

Amazon EMR は Amazon EKS クラスターアクセス管理 (CAM) と統合されているため、Amazon EKS クラスターの名前空間で Amazon EMR Spark ジョブを実行するために必要な AuthN ポリシーと AuthZ ポリシーの設定を自動化できます。これを行うには、次のアクセス許可が必要です。

```
{
  "Effect": "Allow",
  "Action": [
    "eks:CreateAccessEntry"
  ],
  "Resource":
 "arn:<AWS_PARTITION>:eks:<AWS_REGION>:<AWS_ACCOUNT_ID>:cluster/<EKS_CLUSTER_NAME>"
},
{
  "Effect": "Allow",
  "Action": [
    "eks:DescribeAccessEntry",
```

```
    "eks:DeleteAccessEntry",
    "eks:ListAssociatedAccessPolicies",
    "eks:AssociateAccessPolicy",
    "eks:DisassociateAccessPolicy"
  ],
  "Resource": "arn:<AWS_PARTITION>:eks:<AWS_REGION>:<AWS_ACCOUNT_ID>:access-
entry/<EKS_CLUSTER_NAME>/role/<AWS_ACCOUNT_ID>/AWSServiceRoleForAmazonEMRContainers/*"
}
```

詳細については、「Automate enabling cluster access for Amazon EMR on EKS」を参照してください。

AWS アカウントからオペレーションが初めてCreateVirtualCluster呼び出されると、Amazon EMR on EKS のサービスにリンクされたロールを作成するためのCreateServiceLinkedRoleアクセス許可も必要です。詳細については、「Amazon EMR on EKS で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。

ジョブを送信するためのアクセス許可

AWS アカウントの仮想クラスターでジョブを送信するには、次のアクセス許可を持つ IAM ポリシーを作成します。これらのアクセス許可により、アカウント内のすべての仮想クラスターに対して、ジョブ実行の開始、一覧表示、説明、およびキャンセルを行えるようになります。仮想クラスターを一覧表示または記述するアクセス許可を追加することを検討する必要があります。これにより、ジョブを送信する前に仮想クラスターの状態を確認できます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "emr-containers:StartJobRun",
                "emr-containers:ListJobRuns",
                "emr-containers:DescribeJobRun",
                "emr-containers:CancelJobRun"
            ],
            "Resource": "*"
        }
    ]
}
```

デバッグとモニタリングを行うためのアクセス許可

Amazon S3 および CloudWatch にプッシュされたログにアクセスしたり、Amazon EMR コンソールでアプリケーションイベントログを表示したりするには、以下のアクセス許可を持つ IAM ポリシーを作成します。ビジネスニーズに基づいて、可能な限り、特定のリソースに対するアクセス許可を絞り込むことをお勧めします。

> **Important**
>
> Amazon S3 バケットを作成していない場合は、`s3:CreateBucket` アクセス許可をポリシーステートメントに追加する必要があります。ロググループを作成していない場合は、`logs:CreateLogGroup` をポリシーステートメントに追加する必要があります。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "emr-containers:DescribeJobRun",
                "elasticmapreduce:CreatePersistentAppUI",
                "elasticmapreduce:DescribePersistentAppUI",
                "elasticmapreduce:GetPersistentAppUIPresignedURL"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:Get*",
                "logs:DescribeLogGroups",
                "logs:DescribeLogStreams"
            ],
```

```
            "Resource": "*"
        }
    ]
}
```

Amazon S3 および CloudWatch にログをプッシュするようにジョブ実行を設定する方法の詳細については、S3ログ使用のための実行の構成）およびCloudWatchログ使用のためのジョブ実行の構成）を参照してください。

Amazon EKS クラスターをAmazon EMR に登録する

ワークロードを実行するように Amazon EMR on EKS をセットアップするために必要な最後の手順は、クラスターの登録です。

次のコマンドを使用して、前の手順で設定した Amazon EKS クラスターと名前空間の名前を指定して仮想クラスターを作成します。

> **Note**
>
> 各仮想クラスターには、EKS クラスター全体で一意の名前を付ける必要があります。同じ名前の仮想クラスターが 2 つあると、デプロイプロセスは失敗します。これは、その 2 つの仮想クラスターがそれぞれ異なる EKS クラスターに属している場合でも同じです。

```
aws emr-containers create-virtual-cluster \
--name virtual_cluster_name \
--container-provider '{
    "id": "cluster_name",
    "type": "EKS",
    "info": {
        "eksInfo": {
            "namespace": "namespace_name"
        }
    }
}'
```

または、仮想クラスターに必要なパラメータを含む JSON ファイルを作成し、JSON ファイルへのパスを指定して `create-virtual-cluster` コマンドを実行することもできます。詳細については、「仮想クラスターの管理」を参照してください。

> **Note**
>
> 仮想クラスターが正常に作成されたことを確認するには、`list-virtual-clusters` オペレーションを使用するか、Amazon EMR コンソールで Virtual Clusters ページに移動して、仮想クラスターのステータスを確認します。

StartJobRun でジョブ実行を送信する

指定したパラメータを持つ JSON ファイルを使用してジョブ実行を送信するには

1. `start-job-run-request.json` ファイルを作成し、次の JSON ファイルの例に示すように、ジョブ実行に必要なパラメータを指定します。これらのパラメータの詳細については、ジョブ実行を構成するためのオプション を参照してください。

```
{
  "name": "myjob",
  "virtualClusterId": "123456",
  "executionRoleArn": "iam_role_name_for_job_execution",
  "releaseLabel": "emr-6.2.0-latest",
  "jobDriver": {
    "sparkSubmitJobDriver": {
      "entryPoint": "entryPoint_location",
      "entryPointArguments": ["argument1", "argument2", ...],
       "sparkSubmitParameters": "--class <main_class> --conf
 spark.executor.instances=2 --conf spark.executor.memory=2G --conf
 spark.executor.cores=2 --conf spark.driver.cores=1"
    }
  },
  "configurationOverrides": {
    "applicationConfiguration": [
      {
        "classification": "spark-defaults",
        "properties": {
          "spark.driver.memory":"2G"
         }
      }
    ],
    "monitoringConfiguration": {
      "persistentAppUI": "ENABLED",
      "cloudWatchMonitoringConfiguration": {
```

```
        "logGroupName": "my_log_group",
        "logStreamNamePrefix": "log_stream_prefix"
      },
      "s3MonitoringConfiguration": {
        "logUri": "s3://my_s3_log_location"
      }
    }
  }
}
```

2. ローカルに保存されている start-job-run-request.json ファイルへのパスを指定して、start-job-run コマンドを使用します。

```
aws emr-containers start-job-run \
--cli-input-json file://./start-job-run-request.json
```

start-job-run コマンドを使用してジョブ実行を開始するには

1. 次の例に示すように、StartJobRun コマンドで指定したすべてのパラメータを指定します。

```
aws emr-containers start-job-run \
--virtual-cluster-id 123456 \
--name myjob \
--execution-role-arn execution-role-arn \
--release-label emr-6.2.0-latest \
--job-driver '{"sparkSubmitJobDriver": {"entryPoint": "entryPoint_location",
 "entryPointArguments": ["argument1", "argument2", ...], "sparkSubmitParameters":
 "--class <main_class> --conf spark.executor.instances=2 --conf
 spark.executor.memory=2G --conf spark.executor.cores=2 --conf
 spark.driver.cores=1"}}' \
--configuration-overrides '{"applicationConfiguration": [{"classification":
 "spark-defaults", "properties": {"spark.driver.memory": "2G"}}],
 "monitoringConfiguration": {"cloudWatchMonitoringConfiguration":
 {"logGroupName": "log_group_name", "logStreamNamePrefix": "log_stream_prefix"},
 "persistentAppUI":"ENABLED",  "s3MonitoringConfiguration": {"logUri":
 "s3://my_s3_log_location" }}}'
```

2. Spark SQL の場合、以下の例に示すように、StartJobRun コマンドで指定したすべてのパラメータを指定します。

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

```
--virtual-cluster-id 123456 \
--name myjob \
--execution-role-arn execution-role-arn \
--release-label emr-6.7.0-latest \
--job-driver '{"sparkSqlJobDriver": {"entryPoint": "entryPoint_location",
 "sparkSqlParameters": "--conf spark.executor.instances=2 --conf
 spark.executor.memory=2G --conf spark.executor.cores=2 --conf
 spark.driver.cores=1"}}' \
--configuration-overrides '{"applicationConfiguration": [{"classification":
 "spark-defaults", "properties": {"spark.driver.memory": "2G"}}],
 "monitoringConfiguration": {"cloudWatchMonitoringConfiguration":
 {"logGroupName": "log_group_name", "logStreamNamePrefix": "log_stream_prefix"},
 "persistentAppUI":"ENABLED",  "s3MonitoringConfiguration": {"logUri":
 "s3://my_s3_log_location" }}}'
```

ジョブ送信者分類の使用

概要

Amazon EMR on EKS StartJobRun リクエストは、Spark ドライバーを生成するジョブ送信者ポッド (job-runner ポッドとも呼ばれます) を作成します。emr-job-submitter 分類を使用して、ジョブ送信者ポッドのノードセレクタを設定し、ジョブ送信者ポッドのログ記録コンテナのイメージ、CPU、メモリを設定できます。

emr-job-submitter 分類では、次の設定を使用できます。

jobsubmitter.node.selector.[*labelKey*]

キー *labelKey* と値を設定の設定値として、ジョブ送信者ポッドのノードセレクターに追加します。例えば、jobsubmitter.node.selector.identifier を myIdentifier に設定すると、ジョブ送信者ポッドにはキー識別子の値が myIdentifier のノードセレクターが追加されます。これは、ジョブ送信者ポッドを配置できるノードを指定するために使用できます。複数のノードセレクターキーを追加するには、このプレフィックスを使用して複数の設定を設定します。

jobsubmitter.logging.image

ジョブ送信者ポッドのログ記録コンテナに使用するカスタムイメージを設定します。

jobsubmitter.logging.request.cores

ジョブ送信者ポッドのログ記録コンテナの CPUs 数のカスタム値を CPU ユニットで設定します。デフォルトでは、これは 100m に設定されています。

jobsubmitter.logging.request.memory

ジョブ送信者ポッドのログ記録コンテナのメモリ量のカスタム値をバイト単位で設定します。デフォルトでは、これは 200Mi に設定されています。メビバイトは、メガバイトに似た測定単位です。

オンデマンドインスタンスにジョブ送信者ポッドを配置することをお勧めします。ジョブ送信者ポッドをスポットインスタンスに配置すると、ジョブ送信者ポッドが実行されるインスタンスがスポットインスタンスの中断を受けると、ジョブが失敗する可能性があります。ジョブ送信者ポッドを 1 つのアベイラビリティーゾーンに配置することも、ノードに適用されている任意の Kubernetes ラベルを使用することもできます。

ジョブ送信者の分類例

このセクションの内容

- ジョブ送信者ポッドのオンデマンドノード配置を含む StartJobRun リクエスト
- ジョブ送信者ポッドのシングル AZ ノード配置を含む StartJobRun リクエスト
- ジョブ送信者ポッドのシングル AZ および Amazon EC2 インスタンスタイプ配置を含む StartJobRun リクエスト
- StartJobRun カスタムログ記録コンテナイメージ、CPU、メモリを使用した リクエスト

ジョブ送信者ポッドのオンデマンドノード配置を含む **StartJobRun** リクエスト

```
cat >spark-python-in-s3-nodeselector-job-submitter.json << EOF
{
  "name": "spark-python-in-s3-nodeselector",
  "virtualClusterId": "virtual-cluster-id",
  "executionRoleArn": "execution-role-arn",
  "releaseLabel": "emr-6.11.0-latest",
  "jobDriver": {
    "sparkSubmitJobDriver": {
      "entryPoint": "s3://S3-prefix/trip-count.py",
```

```
       "sparkSubmitParameters": "--conf spark.driver.cores=5  --conf
 spark.executor.memory=20G --conf spark.driver.memory=15G --conf
 spark.executor.cores=6"
    }
  },
  "configurationOverrides": {
    "applicationConfiguration": [
      {
        "classification": "spark-defaults",
        "properties": {
          "spark.dynamicAllocation.enabled":"false"
         }
      },
      {
        "classification": "emr-job-submitter",
        "properties": {
            "jobsubmitter.node.selector.eks.amazonaws.com/capacityType": "ON_DEMAND"
        }
      }
    ],
    "monitoringConfiguration": {
      "cloudWatchMonitoringConfiguration": {
        "logGroupName": "/emr-containers/jobs",
        "logStreamNamePrefix": "demo"
      },
      "s3MonitoringConfiguration": {
        "logUri": "s3://joblogs"
      }
    }
  }
}
EOF
aws emr-containers start-job-run --cli-input-json file:///spark-python-in-s3-
nodeselector-job-submitter.json
```

ジョブ送信者ポッドのシングル AZ ノード配置を含む **StartJobRun** リクエスト

```
cat >spark-python-in-s3-nodeselector-job-submitter-az.json << EOF
{
  "name": "spark-python-in-s3-nodeselector",
  "virtualClusterId": "virtual-cluster-id",
  "executionRoleArn": "execution-role-arn",
  "releaseLabel": "emr-6.11.0-latest",
```

```
  "jobDriver": {
    "sparkSubmitJobDriver": {
      "entryPoint": "s3://S3-prefix/trip-count.py",
       "sparkSubmitParameters": "--conf spark.driver.cores=5  --conf
 spark.executor.memory=20G --conf spark.driver.memory=15G --conf
 spark.executor.cores=6"
    }
  },
  "configurationOverrides": {
    "applicationConfiguration": [
      {
        "classification": "spark-defaults",
        "properties": {
          "spark.dynamicAllocation.enabled":"false"
         }
      },
      {
        "classification": "emr-job-submitter",
        "properties": {
            "jobsubmitter.node.selector.topology.kubernetes.io/zone": "Availability
 Zone"
        }
      }
    ],
    "monitoringConfiguration": {
      "cloudWatchMonitoringConfiguration": {
        "logGroupName": "/emr-containers/jobs",
        "logStreamNamePrefix": "demo"
      },
      "s3MonitoringConfiguration": {
        "logUri": "s3://joblogs"
      }
    }
  }
}
EOF
aws emr-containers start-job-run --cli-input-json file:///spark-python-in-s3-
nodeselector-job-submitter-az.json
```

ジョブ送信者ポッドのシングル AZ および Amazon EC2 インスタンスタイプ配置を含む **StartJobRun** リクエスト

```
{
```

```
  "name": "spark-python-in-s3-nodeselector",
  "virtualClusterId": "virtual-cluster-id",
  "executionRoleArn": "execution-role-arn",
  "releaseLabel": "emr-6.11.0-latest",
  "jobDriver": {
    "sparkSubmitJobDriver": {
      "entryPoint": "s3://S3-prefix/trip-count.py",
       "sparkSubmitParameters": "--conf spark.driver.cores=5  --conf
spark.kubernetes.pyspark.pythonVersion=3 --conf spark.executor.memory=20G
--conf spark.driver.memory=15G --conf spark.executor.cores=6 --conf
spark.sql.shuffle.partitions=1000"
    }
  },
  "configurationOverrides": {
    "applicationConfiguration": [
      {
        "classification": "spark-defaults",
        "properties": {
          "spark.dynamicAllocation.enabled":"false",
         }
      },
      {
        "classification": "emr-job-submitter",
        "properties": {
            "jobsubmitter.node.selector.topology.kubernetes.io/zone": "Availability
Zone",
            "jobsubmitter.node.selector.node.kubernetes.io/instance-type":"m5.4xlarge"
        }
      }
    ],
    "monitoringConfiguration": {
      "cloudWatchMonitoringConfiguration": {
        "logGroupName": "/emr-containers/jobs",
        "logStreamNamePrefix": "demo"
      },
      "s3MonitoringConfiguration": {
        "logUri": "s3://joblogs"
      }
    }
  }
}
```

StartJobRun カスタムログ記録コンテナイメージ、CPU、メモリを使用した リクエスト

```
{
  "name": "spark-python",
  "virtualClusterId": "virtual-cluster-id",
  "executionRoleArn": "execution-role-arn",
  "releaseLabel": "emr-6.11.0-latest",
  "jobDriver": {
    "sparkSubmitJobDriver": {
      "entryPoint": "s3://S3-prefix/trip-count.py"
    }
  },
  "configurationOverrides": {
    "applicationConfiguration": [
      {
        "classification": "emr-job-submitter",
        "properties": {
            "jobsubmitter.logging.image": "YOUR_ECR_IMAGE_URL",
            "jobsubmitter.logging.request.memory": "200Mi",
            "jobsubmitter.logging.request.cores": "0.5"
        }
      }
    ],
    "monitoringConfiguration": {
      "cloudWatchMonitoringConfiguration": {
        "logGroupName": "/emr-containers/jobs",
        "logStreamNamePrefix": "demo"
      },
      "s3MonitoringConfiguration": {
        "logUri": "s3://joblogs"
      }
    }
  }
}
```

Amazon EMR コンテナのデフォルト分類の使用

概要

emr-containers-defaults 分類では、次の設定を使用できます。

job-start-timeout

デフォルトでは、ジョブが開始できず、 SUBMITTED状態で 15 分間待機すると、ジョブはタイムアウトします。この設定では、ジョブがタイムアウトするまで待機する秒数を変更します。

logging.image

ドライバーポッドとエグゼキューターポッドのログ記録コンテナに使用するカスタムイメージを設定します。

logging.request.cores

ドライバーポッドとエグゼキューターポッドのログ記録コンテナの CPUs 数のカスタム値を CPU ユニットで設定します。デフォルトでは、これは設定されていません。

logging.request.memory

ドライバーポッドとエグゼキューターポッドのログ記録コンテナのメモリ量のカスタム値をバイト単位で設定します。デフォルトでは、これは 512Mi に設定されています。メビバイトは、メガバイトに似た測定単位です。

ジョブ送信者の分類例

このセクションの内容

- StartJobRun カスタムジョブタイムアウトによる リクエスト
- StartJobRun カスタムログ記録コンテナイメージ、CPU、メモリを使用した リクエスト

StartJobRun カスタムジョブタイムアウトによる リクエスト

```
{
  "name": "spark-python",
  "virtualClusterId": "virtual-cluster-id",
  "executionRoleArn": "execution-role-arn",
  "releaseLabel": "emr-6.11.0-latest",
  "jobDriver": {
    "sparkSubmitJobDriver": {
      "entryPoint": "s3://S3-prefix/trip-count.py"
    }
  },
  "configurationOverrides": {
    "applicationConfiguration": [
      {
```

```
        "classification": "emr-containers-defaults",
        "properties": {
            "job-start-timeout": "1800"
        }
      }
    ],
    "monitoringConfiguration": {
      "cloudWatchMonitoringConfiguration": {
        "logGroupName": "/emr-containers/jobs",
        "logStreamNamePrefix": "demo"
      },
      "s3MonitoringConfiguration": {
        "logUri": "s3://joblogs"
      }
    }
  }
}
```

StartJobRun カスタムログ記録コンテナイメージ、CPU、メモリを使用した リクエスト

```
{
  "name": "spark-python",
  "virtualClusterId": "virtual-cluster-id",
  "executionRoleArn": "execution-role-arn",
  "releaseLabel": "emr-6.11.0-latest",
  "jobDriver": {
    "sparkSubmitJobDriver": {
      "entryPoint": "s3://S3-prefix/trip-count.py"
    }
  },
  "configurationOverrides": {
    "applicationConfiguration": [
      {
        "classification": "emr-containers-defaults",
        "properties": {
            "logging.image": "YOUR_ECR_IMAGE_URL",
            "logging.request.memory": "200Mi",
            "logging.request.cores": "0.5"
        }
      }
    ],
    "monitoringConfiguration": {
      "cloudWatchMonitoringConfiguration": {
```

```
        "logGroupName": "/emr-containers/jobs",
        "logStreamNamePrefix": "demo"
      },
      "s3MonitoringConfiguration": {
        "logUri": "s3://joblogs"
      }
    }
  }
}
```

Spark 演算子を使用して Spark ジョブを実行する

Amazon EMR リリース 6.10.0 以降では、Amazon EMR on EKS のジョブ送信モデルとして、Apache Spark の Kubernetes 演算子、または Spark 演算子がサポートされています。Spark 演算子を使用すると、Amazon EMR リリースランタイムを使用して Spark アプリケーションを独自の Amazon EKS クラスターにデプロイおよび管理できます。Amazon EKS クラスターに Spark 演算子をデプロイすると、その演算子を使用して Spark アプリケーションを直接送信できます。演算子は Spark アプリケーションのライフサイクルを管理します。

> **Note**
>
> Amazon EMR は、vCPU とメモリ消費量に基づいて Amazon EKS の料金を算出します。この計算は、ドライバーポッドとエグゼキューターポッドに適用されます。この計算は、Amazon EMR アプリケーションイメージをダウンロードしてから Amazon EKS ポッドが終了するまでに開始され、秒単位で四捨五入されます。

トピック

Amazon EMR on EKS での Spark 演算子のセットアップ

Spark 演算子を Amazon EKS にインストールする前に、以下のタスクを完了してセットアップを行います。Amazon Web Services (AWS) に既にサインアップしていて、Amazon EKS を既に使用している場合、Amazon EMR on EKS を使用する準備はほぼ整っています。Amazon EKS で Spark 演算子のセットアップを行うには、以下のタスクを完了します。前提条件のいずれかを既に完了している場合は、その前提条件をスキップして、次の前提条件に進むことができます。

- の最新バージョンをインストールまたは更新する AWS CLI – を既にインストールしている場合は AWS CLI、最新バージョンであることを確認します。
- kubectl と eksctl の設定 — eksctl は、Amazon EKS との通信に使用するコマンドラインツールです。
- Helm のインストール – Kubernetes 用の Helm パッケージマネージャーを使用すると、Kubernetes クラスターにアプリケーションをインストールして管理できます。
- Amazon EKS – eksctl の使用開始 – Amazon EKS にノードを持つ新しい Kubernetes クラスターを作成する手順に従います。
- Amazon EMR ベースイメージ URI を選択する (リリース 6.10.0 以降) – Spark 演算子は Amazon EMR リリース 6.10.0 以降でサポートされています。

Amazon EMR on EKS で Spark 演算子の使用を開始する

このトピックは、Spark アプリケーションとスケジュール Spark アプリケーションをデプロイして、Amazon EKS で Spark 演算子の使用を開始するのに役立ちます。

Spark 演算子をインストールする

Apache Spark 用の Kubernetes 演算子をインストールするには、以下のステップを実行します。

1. 「Amazon EMR on EKS での Spark 演算子のセットアップ」のステップをまだ完了していない場合は完了します。
2. Amazon ECR レジストリに対し、Helm クライアントを認証します。以下のコマンドで、*region-id* 値を任意の AWS リージョンに置き換え、リージョン別の Amazon ECR レジストリアカウント ページに表示されているリージョンに対応する *ECR-registry-account* 値を置き換えます。

```
aws ecr get-login-password \
--region region-id | helm registry login \
```

```
--username AWS \
--password-stdin ECR-registry-account.dkr.ecr.region-id.amazonaws.com
```

3. 以下のコマンドで Spark 演算子をインストールします。

   Helm チャート --version パラメータには、emr- プレフィックスと日付サフィックスを削除した Amazon EMR リリースラベルを使用してください。例えば、emr-6.12.0-java17-latest リリースでは、6.12.0-java17 を指定します。以下のコマンドの例では emr-7.6.0-latest リリースを使用しているため、Helm チャート --version 用に 7.6.0 を指定しています。

   ```
   helm install spark-operator-demo \
     oci://895885662937.dkr.ecr.region-id.amazonaws.com/spark-operator \
     --set emrContainers.awsRegion=region-id \
     --version 7.6.0 \
     --namespace spark-operator \
     --create-namespace
   ```

   デフォルトでは、このコマンドは Spark 演算子のサービスアカウント emr-containers-sa-spark-operator を作成します。別のサービスアカウントを使用するには、引数 serviceAccounts.sparkoperator.name を指定します。以下に例を示します。

   ```
   --set serviceAccounts.sparkoperator.name my-service-account-for-spark-operator
   ```

   Spark 演算子で垂直的自動スケーリングを使用する場合は、インストールコマンドに以下の行を追加して、演算子にウェブフックを許可します。

   ```
   --set webhook.enable=true
   ```

4. helm list コマンドで Helm チャートをインストールしたことを確認します。

   ```
   helm list --namespace spark-operator -o yaml
   ```

   helm list コマンドは、新しくデプロイした Helm チャートのリリース情報を返します。

   ```
   app_version: v1beta2-1.3.8-3.1.1
   chart: spark-operator-7.6.0
   name: spark-operator-demo
   namespace: spark-operator
   revision: "1"
   ```

```
status: deployed
updated: 2023-03-14 18:20:02.721638196 +0000 UTC
```

5. 必要な追加オプションを指定してインストールを完了します。詳しくは、GitHub の「spark-on-k8s-operator」ドキュメントを参照してください。

Spark アプリケーションの実行

Spark 演算子は Amazon EMR 6.10.0 以降でサポートされています。Spark 演算子をインストールすると、Spark アプリケーションを実行するサービスアカウント `emr-containers-sa-spark` がデフォルトで作成されます。Amazon EMR on EKS 6.10.0 以降で Spark 演算子を使用して Spark アプリケーションを実行するには、以下のステップを実行します。

1. Spark 演算子を使用して Spark アプリケーションを実行する前に、Amazon EMR on EKS での Spark 演算子のセットアップ および Spark 演算子をインストールする のステップを完了してください。
2. 以下の例の内容で `SparkApplication` 定義ファイル `spark-pi.yaml` を作成します。

```
apiVersion: "sparkoperator.k8s.io/v1beta2"
kind: SparkApplication
metadata:
  name: spark-pi
  namespace: spark-operator
spec:
  type: Scala
  mode: cluster
  image: "895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-6.10.0:latest"
  imagePullPolicy: Always
  mainClass: org.apache.spark.examples.SparkPi
  mainApplicationFile: "local:///usr/lib/spark/examples/jars/spark-examples.jar"
  sparkVersion: "3.3.1"
  restartPolicy:
    type: Never
  volumes:
    - name: "test-volume"
      hostPath:
        path: "/tmp"
        type: Directory
  driver:
    cores: 1
    coreLimit: "1200m"
```

```
    memory: "512m"
    labels:
      version: 3.3.1
    serviceAccount: emr-containers-sa-spark
    volumeMounts:
      - name: "test-volume"
        mountPath: "/tmp"
  executor:
    cores: 1
    instances: 1
    memory: "512m"
    labels:
      version: 3.3.1
    volumeMounts:
      - name: "test-volume"
        mountPath: "/tmp"
```

3. 次に、以下のコマンドを使用して、Spark アプリケーションを送信します。これにより、spark-pi という名前の SparkApplication オブジェクトも作成されます。

```
kubectl apply -f spark-pi.yaml
```

4. 次のコマンドで SparkApplication オブジェクトのイベントがないか確認します。

```
kubectl describe sparkapplication spark-pi --namespace spark-operator
```

Spark 演算子を使用して Spark にアプリケーションを送信する方法の詳細については、GitHub の spark-on-k8s-operator ドキュメントの「Using a SparkApplication」を参照してください。

Amazon S3 for storage を使用する

Amazon S3 をファイルストレージオプションとして使用するには、YAML ファイルに次の設定を追加します。

```
hadoopConf:
# EMRFS filesystem
  fs.s3.customAWSCredentialsProvider:
 com.amazonaws.auth.WebIdentityTokenCredentialsProvider
  fs.s3.impl: com.amazon.ws.emr.hadoop.fs.EmrFileSystem
  fs.AbstractFileSystem.s3.impl: org.apache.hadoop.fs.s3.EMRFSDelegate
```

```
  fs.s3.buffer.dir: /mnt/s3
  fs.s3.getObject.initialSocketTimeoutMilliseconds: "2000"
  mapreduce.fileoutputcommitter.algorithm.version.emr_internal_use_only.EmrFileSystem:
 "2"
  mapreduce.fileoutputcommitter.cleanup-
failures.ignored.emr_internal_use_only.EmrFileSystem: "true"
sparkConf:
 # Required for EMR Runtime
 spark.driver.extraClassPath: /usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/hadoop-
aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/emr/emrfs/conf:/usr/share/aws/
emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/auxlib/*:/usr/share/aws/emr/security/conf:/
usr/share/aws/emr/security/lib/*:/usr/share/aws/hmclient/lib/aws-glue-datacatalog-
spark-client.jar:/usr/share/java/Hive-JSON-Serde/hive-openx-serde.jar:/usr/share/aws/
sagemaker-spark-sdk/lib/sagemaker-spark-sdk.jar:/home/hadoop/extrajars/*
 spark.driver.extraLibraryPath: /usr/lib/hadoop/lib/native:/usr/lib/hadoop-lzo/lib/
native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/lib/native
 spark.executor.extraClassPath: /usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/hadoop-
aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/emr/emrfs/conf:/usr/share/aws/
emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/auxlib/*:/usr/share/aws/emr/security/conf:/
usr/share/aws/emr/security/lib/*:/usr/share/aws/hmclient/lib/aws-glue-datacatalog-
spark-client.jar:/usr/share/java/Hive-JSON-Serde/hive-openx-serde.jar:/usr/share/aws/
sagemaker-spark-sdk/lib/sagemaker-spark-sdk.jar:/home/hadoop/extrajars/*
 spark.executor.extraLibraryPath: /usr/lib/hadoop/lib/native:/usr/lib/hadoop-lzo/lib/
native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/lib/native
```

Amazon EMR リリース 7.2.0 以降を使用している場合、この設定はデフォルトで含まれています。この場合、Spark アプリケーション YAML ファイルの `local://<file_path>` ではなく、`s3://<bucket_name>/<file_path>` にファイルパスを設定できます。

次に、Spark アプリケーションを通常どおり送信します。

Amazon EMR on EKS の Spark 演算子で垂直的自動スケーリングを使用する

Amazon EMR 7.0 以降では、Amazon EMR on EKS の垂直的自動スケーリングを使用してリソース管理を簡素化できます。Amazon EMR Spark アプリケーションに提供するワークロードのニーズに合わせて、メモリと CPU リソースを自動的に調整します。詳細については、「Amazon EMR Spark ジョブで垂直的自動スケーリングを使用する」を参照してください。

このセクションでは、垂直的自動スケーリングを使用するように Spark 演算子を設定する方法について説明します。

前提条件

モニタリングを設定する前に、次のセットアップタスクを完了してください。

- 「Amazon EMR on EKS での Spark 演算子のセットアップ」のステップを完了します。
- (オプション) 以前に古いバージョンの Spark 演算子をインストールした場合は、SparkApplication/ScheduledSparkApplication CRD を削除します。

```
kubectl delete crd sparkApplication
kubectl delete crd scheduledSparkApplication
```

- 「Spark 演算子をインストールする」のステップを完了します。ステップ 3 では、インストールコマンドに以下の行を追加して、演算子にウェブフックを許可します。

```
--set webhook.enable=true
```

- 「Amazon EMR on EKS の垂直的自動スケーリングのセットアップ」のステップを完了します。
- 次の Amazon S3 の場所にあるファイルへのアクセスを許可します。

1. S3 アクセス権限を持つ JobExecutionRole を使用して、ドライバーとオペレーターのサービスアカウントに注釈を付けます。

```
kubectl annotate serviceaccount -n spark-operator emr-containers-sa-spark
 eks.amazonaws.com/role-arn=JobExecutionRole
kubectl annotate serviceaccount -n spark-operator emr-containers-sa-spark-
operator eks.amazonaws.com/role-arn=JobExecutionRole
```

2. その名前空間でジョブ実行ロールの信頼ポリシーを更新します。

```
aws emr-containers update-role-trust-policy \
--cluster-name cluster \
--namespace ${Namespace}\
--role-name iam_role_name_for_job_execution
```

3. ジョブ実行ロールの IAM ロール信頼ポリシーを編集し、serviceaccount を emr-containers-sa-spark-*-*-xxxx から emr-containers-sa-* に更新します。

```
{
    "Effect": "Allow",
    "Principal": {
        "Federated": "OIDC-provider"
```

```
    },
    "Action": "sts:AssumeRoleWithWebIdentity",
    "Condition": {
        "StringLike": {
            "OIDC": "system:serviceaccount:${Namespace}:emr-containers-sa-*"
        }
    }
}
```

4. Amazon S3 をファイルストレージとして使用している場合は、yaml ファイルに次のデフォルトを追加します。

```
hadoopConf:
# EMRFS filesystem
  fs.s3.customAWSCredentialsProvider:
 com.amazonaws.auth.WebIdentityTokenCredentialsProvider
  fs.s3.impl: com.amazon.ws.emr.hadoop.fs.EmrFileSystem
  fs.AbstractFileSystem.s3.impl: org.apache.hadoop.fs.s3.EMRFSDelegate
  fs.s3.buffer.dir: /mnt/s3
  fs.s3.getObject.initialSocketTimeoutMilliseconds: "2000"

 mapreduce.fileoutputcommitter.algorithm.version.emr_internal_use_only.EmrFileSystem:
 "2"
  mapreduce.fileoutputcommitter.cleanup-
failures.ignored.emr_internal_use_only.EmrFileSystem: "true"
sparkConf:
 # Required for EMR Runtime
 spark.driver.extraClassPath: /usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/hadoop-
aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/emr/emrfs/conf:/usr/share/
aws/emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/auxlib/*:/usr/share/aws/emr/
security/conf:/usr/share/aws/emr/security/lib/*:/usr/share/aws/hmclient/lib/aws-
glue-datacatalog-spark-client.jar:/usr/share/java/Hive-JSON-Serde/hive-openx-
serde.jar:/usr/share/aws/sagemaker-spark-sdk/lib/sagemaker-spark-sdk.jar:/home/
hadoop/extrajars/*
 spark.driver.extraLibraryPath: /usr/lib/hadoop/lib/native:/usr/lib/hadoop-lzo/
lib/native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/lib/
native
 spark.executor.extraClassPath: /usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/
hadoop-aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/emr/emrfs/conf:/usr/
share/aws/emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/auxlib/*:/usr/share/aws/emr/
security/conf:/usr/share/aws/emr/security/lib/*:/usr/share/aws/hmclient/lib/aws-
glue-datacatalog-spark-client.jar:/usr/share/java/Hive-JSON-Serde/hive-openx-
```

```
serde.jar:/usr/share/aws/sagemaker-spark-sdk/lib/sagemaker-spark-sdk.jar:/home/
hadoop/extrajars/*
 spark.executor.extraLibraryPath: /usr/lib/hadoop/lib/native:/usr/lib/hadoop-
lzo/lib/native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/lib/
native
```

Spark 演算子で垂直的自動スケーリングを使用してジョブを実行する

Spark 演算子を使用して Spark アプリケーションを実行する前に、「前提条件」のステップを完了する必要があります。

Spark 演算子で垂直的自動スケーリングを使用するには、Spark アプリケーション仕様のドライバーに次の設定を追加して、垂直的自動スケーリングを有効にします。

```
dynamicSizing:
  mode: Off
  signature: "my-signature"
```

この設定は垂直的自動スケーリングを有効にし、ジョブの署名を選択できる必須の署名設定です。

設定とパラメータ値の詳細については、「Amazon EMR on EKS の垂直的自動スケーリングの設定」を参照してください。デフォルトでは、ジョブは垂直的自動スケーリングのモニタリング専用 [オフ] モードで送信されます。このモニタリング状態では、自動スケーリングを実行しなくてもリソースレコメンデーションを計算して表示できます。詳細については、「垂直的自動スケーリングモード」を参照してください。

以下の例は、垂直的自動スケーリングを使用するために必要な設定を含む `spark-pi.yaml` という名前の `SparkApplication` 定義ファイルです。

```
apiVersion: "sparkoperator.k8s.io/v1beta2"
kind: SparkApplication
metadata:
  name: spark-pi
  namespace: spark-operator
spec:
  type: Scala
  mode: cluster
  image: "895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-7.6.0:latest"
  imagePullPolicy: Always
  mainClass: org.apache.spark.examples.SparkPi
```

```
  mainApplicationFile: "local:///usr/lib/spark/examples/jars/spark-examples.jar"
  sparkVersion: "3.4.1"
  dynamicSizing:
    mode: Off
    signature: "my-signature"
  restartPolicy:
    type: Never
  volumes:
    - name: "test-volume"
      hostPath:
        path: "/tmp"
        type: Directory
  driver:
    cores: 1
    coreLimit: "1200m"
    memory: "512m"
    labels:
      version: 3.4.1
    serviceAccount: emr-containers-sa-spark
    volumeMounts:
      - name: "test-volume"
        mountPath: "/tmp"
  executor:
    cores: 1
    instances: 1
    memory: "512m"
    labels:
      version: 3.4.1
    volumeMounts:
      - name: "test-volume"
        mountPath: "/tmp"
```

次に、以下のコマンドを使用して、Spark アプリケーションを送信します。これにより、spark-pi という名前の SparkApplication オブジェクトも作成されます。

```
kubectl apply -f spark-pi.yaml
```

Spark 演算子を使用して Spark にアプリケーションを送信する方法の詳細については、GitHub の spark-on-k8s-operator ドキュメントの「Using a SparkApplication」を参照してください。

垂直的自動スケーリング機能の検証

垂直的自動スケーリングが送信されたジョブで正しく機能することを確認するには、kubectl を使用して verticalpodautoscaler カスタムリソースを取得し、スケーリングの推奨事項を確認してください。

```
kubectl get verticalpodautoscalers --all-namespaces \
-l=emr-containers.amazonaws.com/dynamic.sizing.signature=my-signature
```

このクエリの出力は以下のようになります。

```
NAMESPACE        NAME                                                        MODE
 CPU    MEM          PROVIDED    AGE
spark-operator   ds-p73j6mkosvc4xeb3gr7x4xol2bfcw5evqimzqojrlysvj3giozuq-vpa   Off
     580026651    True        15m
```

出力が類似していない場合やエラーコードが含まれている場合は、「Amazon EMR on EKS 垂直自動スケーリングのトラブルシューティング」を参照して、問題の解決に役立つ手順をご覧ください。

ポッドとアプリケーションを削除するには、以下のコマンドを実行します。

```
kubectl delete sparkapplication spark-pi
```

Amazon EMR on EKS での Spark 演算子のアンインストール

Spark 演算子をアンインストールするには、以下の手順を実行します。

1. 正しい名前空間を使用して Spark 演算子を削除します。この例の場合、名前空間は spark-operator-demo になります。

   ```
   helm uninstall spark-operator-demo -n spark-operator
   ```

2. Spark 演算子サービスアカウントを削除します。

   ```
   kubectl delete sa emr-containers-sa-spark-operator -n spark-operator
   ```

3. Spark 演算子 CustomResourceDefinitions (CRD) を削除します。

   ```
   kubectl delete crd sparkapplications.sparkoperator.k8s.io
   kubectl delete crd scheduledsparkapplications.sparkoperator.k8s.io
   ```

モニタリング設定を使用して Spark Kubernetes オペレータと Spark ジョブをモニタリングする

モニタリング設定を使用すると、Spark アプリケーションとオペレータログのログアーカイブを Amazon S3 または に簡単に設定できます Amazon CloudWatch。1 つまたは両方を選択できます。これにより、ログエージェントのサイドカーがSpark オペレータポッド、ドライバー、エグゼキュターポッドに追加され、その後、これらのコンポーネントのログが設定されたシンクに転送されます。

前提条件

モニタリングを設定する前に、次のセットアップタスクを完了してください。

1. (オプション) 以前に古いバージョンの Spark 演算子をインストールした場合は、SparkApplication/ScheduledSparkApplication CRD を削除します。

```
kubectl delete crd scheduledsparkapplications.sparkoperator.k8s.io
kubectl delete crd sparkapplications.sparkoperator.k8s.io
```

2. まだ持っていない場合は、IAM でオペレーター/ジョブ実行ロールを作成します。
3. 次のコマンドを実行して、先ほど作成したオペレーター/ジョブ実行ロールの信頼ポリシーを更新します。

```
aws emr-containers update-role-trust-policy \
--cluster-name cluster \
--namespace namespace \
--role-name iam_role_name_for_operator/job_execution_role
```

4. オペレーター/ジョブ実行ロールの IAM ロール信頼ポリシーを次のように編集します。

```
{
    "Effect": "Allow",
    "Principal": {
        "Federated": "${OIDC-provider}"
    },
    "Action": "sts:AssumeRoleWithWebIdentity",
    "Condition": {
        "StringLike": {
            "OIDC_PROVIDER:sub": "system:serviceaccount:${Namespace}:emr-
containers-sa-*"
```

```
        }
    }
}
```

5. 次のアクセス許可を使用して、IAM で monitoringConfiguration ポリシーを作成します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogStreams",
                "logs:CreateLogStream",
                "logs:CreateLogGroup",
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:region:account_id:log-group:log_group_name",
                "arn:aws:logs:region:account_id:log-group:log_group_name:*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "logs:DescribeLogGroups",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::bucket_name",
                "arn:aws:s3:::bucket_name/*"
            ]
        }
    ]
}
```

6. 上記のポリシーをオペレーター/ジョブ実行ロールにアタッチします。

Spark オペレーターログ

モニタリング設定は、 を実行するときに次の方法で定義できますhelm install。

```
helm install spark-operator spark-operator \
--namespace namespace \
--set emrContainers.awsRegion=aws_region \
--set emrContainers.monitoringConfiguration.image=log_agent_image_url \
--set
 emrContainers.monitoringConfiguration.s3MonitoringConfiguration.logUri=S3_bucket_uri \
--set
 emrContainers.monitoringConfiguration.cloudWatchMonitoringConfiguration.logGroupName=log_group
 \
--set
 emrContainers.monitoringConfiguration.cloudWatchMonitoringConfiguration.logStreamNamePrefix=lo
 \
--set emrContainers.monitoringConfiguration.sideCarResources.limits.cpuLimit=500m \
--set emrContainers.monitoringConfiguration.sideCarResources.limits.memoryLimit=512Mi \
--set
 emrContainers.monitoringConfiguration.containerLogRotationConfiguration.rotationSize=2GB
 \
--set
 emrContainers.monitoringConfiguration.containerLogRotationConfiguration.maxFilesToKeep=10
 \
--set webhook.enable=true \
--set emrContainers.operatorExecutionRoleArn=operator_execution_role_arn
```

モニタリング設定

monitoringConfiguration で使用可能な設定オプションを次に示します。

- イメージ (オプション) – ログエージェントのイメージ URL。指定しない場合、emrReleaseLabel で取得されます。
- s3MonitoringConfiguration – このオプションを設定して Amazon S3 にアーカイブします。
  - logUri – (必須) – ログを保存する Amazon S3 バケットパス。
  - 以下は、ログのアップロード後の Amazon S3 バケットパスのサンプル形式です。最初の例は、ログローテーションが有効になっていないことを示しています。

```
s3://${logUri}/${POD NAME}/operator/stdout.gz
s3://${logUri}/${POD NAME}/operator/stderr.gz
```

ログローテーションはデフォルトで有効になっています。ローテーションされたファイルとインクリメントインデックスの両方と、前のサンプルと同じ現在のファイルの両方を表示できます。

```
s3://${logUri}/${POD NAME}/operator/stdout_YYYYMMDD_index.gz
s3://${logUri}/${POD NAME}/operator/stderr_YYYYMMDD_index.gz
```

- cloudWatchMonitoringConfiguration – 転送を設定する設定キー Amazon CloudWatch。
  - logGroupName (必須) – Amazon CloudWatch ログの送信先となるロググループの名前。このグループが存在しない場合は自動的に作成されます。
  - logStreamNamePrefix (オプション) – ログを送信するログストリームの名前。デフォルト値は空の文字列です。の形式 Amazon CloudWatch は次のとおりです。

```
${logStreamNamePrefix}/${POD NAME}/STDOUT or STDERR
```

- sideCarResources (オプション) – 起動した Fluentd サイドカーコンテナにリソース制限を設定するための設定キー。
  - memoryLimit (オプション) – メモリ制限。必要に応じて調整してください。デフォルトは 512Mi です。
  - cpuLimit (オプション) – CPU 制限。必要に応じて調整してください。デフォルトは 500m です。
- containerLogRotationConfiguration (オプション) – コンテナログのローテーション動作を制御します。このエージェントは、デフォルトでは有効になっています。
  - rotationSize (必須) – ログローテーションのファイルサイズを指定します。指定できる値の範囲は 2 KB ~ 2 GB です。rotationSize パラメータの数値単位部分は整数として渡されます。小数値はサポートされていないため、値 1500 MB などに 1.5 GB のローテーションサイズを指定できます。デフォルトは 2 GB です。
  - maxFilesToKeep (必須) – ローテーションの実行後にコンテナに保持するファイルの最大数を指定します。最小値は 1 で、最大値は 50 です。デフォルトは 10 です。

monitoringConfiguration を設定したら、Amazon S3 バケット Amazon CloudWatch または両方で spark 演算子ポッドログを確認できるようになります。Amazon S3 バケットの場合、最初のログファイルがフラッシュされるまで 2 分待つ必要があります。

でログを検索するには Amazon CloudWatch、CloudWatch > Log Groups > ***Log group name*** > *Pod name*/operator/stderr に移動します。

または、CloudWatch > ロググループ > ******* > ####/オペレーター/stdout に移動することもできます。

Spark アプリケーションログ

この設定は、次の方法で定義できます。

```
apiVersion: "sparkoperator.k8s.io/v1beta2"
kind: SparkApplication
metadata:
  name: spark-pi
  namespace: namespace
spec:
  type: Scala
  mode: cluster
  imagePullPolicy: Always
  mainClass: org.apache.spark.examples.SparkPi
  mainApplicationFile: "local:///usr/lib/spark/examples/jars/spark-examples.jar"
  sparkVersion: "3.3.1"
  emrReleaseLabel: emr_release_label
  executionRoleArn: job_execution_role_arn
  restartPolicy:
    type: Never
  volumes:
    - name: "test-volume"
      hostPath:
        path: "/tmp"
        type: Directory
  driver:
    cores: 1
    coreLimit: "1200m"
    memory: "512m"
    labels:
      version: 3.3.1
    volumeMounts:
      - name: "test-volume"
        mountPath: "/tmp"
  executor:
    cores: 1
    instances: 1
    memory: "512m"
    labels:
      version: 3.3.1
```

```
    volumeMounts:
      - name: "test-volume"
        mountPath: "/tmp"
  monitoringConfiguration:
    image: "log_agent_image"
    s3MonitoringConfiguration:
      logUri: "S3_bucket_uri"
    cloudWatchMonitoringConfiguration:
      logGroupName: "log_group_name"
      logStreamNamePrefix: "log_stream_prefix"
    sideCarResources:
      limits:
        cpuLimit: "500m"
        memoryLimit: "250Mi"
    containerLogRotationConfiguration:
      rotationSize: "2GB"
      maxFilesToKeep: "10"
```

monitoringConfiguration で使用可能な設定オプションを次に示します。

- イメージ (オプション) – ログエージェントのイメージ URL。指定しない場合、emrReleaseLabel で取得されます。
- s3MonitoringConfiguration – このオプションを設定して Amazon S3 にアーカイブします。
  - logUri (必須) – ログを保存する Amazon S3 バケットパス。最初の例は、ログローテーションが有効になっていないことを示しています。

    ```
    s3://${logUri}/${APPLICATION NAME}-${APPLICATION UID}/${POD NAME}/stdout.gz
    s3://${logUri}/${APPLICATION NAME}-${APPLICATION UID}/${POD NAME}/stderr.gz
    ```

    ログローテーションはデフォルトで有効になっています。ローテーションされたファイル (インクリメントインデックス付き) と現在のファイル (日付スタンプなし) の両方を使用できます。

    ```
    s3://${logUri}/${APPLICATION NAME}-${APPLICATION UID}/${POD NAME}/
    stdout_YYYYMMDD_index.gz
    s3://${logUri}/${APPLICATION NAME}-${APPLICATION UID}/${POD NAME}/
    stderr_YYYYMMDD_index.gz
    ```

- cloudWatchMonitoringConfiguration – 転送を設定する設定キー Amazon CloudWatch。
  - logGroupName (必須) – ログを送信する Cloudwatch ロググループの名前。グループが存在しない場合は、自動的に作成されます。

- logStreamNamePrefix (オプション) – ログを送信するログストリームの名前。デフォルト値は空の文字列です。CloudWatch の形式は次のとおりです。

```
${logStreamNamePrefix}/${APPLICATION NAME}-${APPLICATION UID}/${POD NAME}/stdout
${logStreamNamePrefix}/${APPLICATION NAME}-${APPLICATION UID}/${POD NAME}/stderr
```

- sideCarResources (オプション) – 起動した Fluentd サイドカーコンテナにリソース制限を設定するための設定キー。
  - memoryLimit (オプション) – メモリ制限。必要に応じて調整してください。デフォルトは 250Mi です。
  - cpuLimit – CPU 制限。必要に応じて調整してください。デフォルトは 500m です。
- containerLogRotationConfiguration (オプション) – コンテナログのローテーション動作を制御します。このエージェントは、デフォルトでは有効になっています。
  - rotationSize (必須) – ログローテーションのファイルサイズを指定します。指定できる値の範囲は 2 KB ~ 2 GB です。rotationSize パラメータの数値単位部分は整数として渡されます。小数値はサポートされていないため、値 1500 MB などに 1.5 GB のローテーションサイズを指定できます。デフォルトは 2 GB です。
  - maxFilesToKeep (必須) – ローテーションの実行後にコンテナに保持するファイルの最大数を指定します。最小値は 1 です。最大値は 50 です。デフォルトは 10 です。

monitoringConfiguration を設定したら、Amazon S3 バケットまたは CloudWatch またはその両方で spark アプリケーションドライバーとエグゼキューターログを確認できます。Amazon S3 バケットの場合、最初のログファイルがフラッシュされるまで 2 分待つ必要があります。例えば、Amazon S3 では、バケットパスは次のようになります。

Amazon S3 > バケット > ***#####*** > *Spark ######### - UUID* > *####* > stderr.gz

または:

Amazon S3 > バケット > ***#####*** > *Spark ######### - UUID* > *####* > stdout.gz

CloudWatch では、パスは次のようになります。

CloudWatch > ロググループ > ***#######*** > *Spark ######### - UUID*/ *####*/stderr

または:

CloudWatch > ロググループ > ***#######*** > *Spark ######### - UUID*/ *####*/stdout

Amazon EMR on EKS でのセキュリティと Spark 演算子

Spark 演算子を使用する際にクラスターアクセス権限を設定する方法は 2 つあります。1 番目は、ロールベースのアクセスコントロールを使用する方法です。ロールベースのアクセスコントロール (RBAC) は、組織内のユーザーロールに基づいてアクセスを制限します。これはアクセス権を処理する主要な方法となっています。2 番目のアクセス方法は、 AWS Identity and Access Management ロールを引き受けることです。これにより、割り当てられた特定のアクセス許可によってリソースアクセスが提供されます。

トピック

- ロールベースアクセスコントロール (RBAC、role-based access control) を使用したクラスターのアクセス許可の設定
- サービスアカウント用 IAM ロール (IRSA、IAM roles for service accounts) でクラスターアクセス権限を設定する

ロールベースアクセスコントロール (RBAC、role-based access control) を使用したクラスターのアクセス許可の設定

Spark 演算子をデプロイするために、Amazon EMR on EKS は Spark 演算子と Spark アプリケーション用に 2 つのロールとサービスアカウントを作成します。

トピック

- 演算子サービスアカウントとロール
- Spark サービスのアカウントとロール

演算子サービスアカウントとロール

Amazon EMR on EKS は、Spark ジョブやサービスなどの他のリソース用の `SparkApplications` を管理するための演算子サービスアカウントとロールを作成します。

このサービスアカウントのデフォルト名は `emr-containers-sa-spark-operator` です。

このサービスロールには、以下の規則が適用されます。

```
 rules:
- apiGroups:
  - ""
  resources:
```

```
  - pods
  verbs:
  - "*"
- apiGroups:
  - ""
  resources:
  - services
  - configmaps
  - secrets
  verbs:
  - create
  - get
  - delete
  - update
- apiGroups:
  - extensions
  - networking.k8s.io
  resources:
  - ingresses
  verbs:
  - create
  - get
  - delete
- apiGroups:
  - ""
  resources:
  - nodes
  verbs:
  - get
- apiGroups:
  - ""
  resources:
  - events
  verbs:
  - create
  - update
  - patch
- apiGroups:
  - ""
  resources:
  - resourcequotas
  verbs:
  - get
  - list
```

```
  - watch
- apiGroups:
  - apiextensions.k8s.io
  resources:
  - customresourcedefinitions
  verbs:
  - create
  - get
  - update
  - delete
- apiGroups:
  - admissionregistration.k8s.io
  resources:
  - mutatingwebhookconfigurations
  - validatingwebhookconfigurations
  verbs:
  - create
  - get
  - update
  - delete
- apiGroups:
  - sparkoperator.k8s.io
  resources:
  - sparkapplications
  - sparkapplications/status
  - scheduledsparkapplications
  - scheduledsparkapplications/status
  verbs:
  - "*"
  {{- if .Values.batchScheduler.enable }}
  # required for the `volcano` batch scheduler
- apiGroups:
  - scheduling.incubator.k8s.io
  - scheduling.sigs.dev
  - scheduling.volcano.sh
  resources:
  - podgroups
  verbs:
  - "*"
  {{- end }}
  {{ if .Values.webhook.enable }}
- apiGroups:
  - batch
  resources:
```

```
  - jobs
  verbs:
  - delete
  {{- end }}
```

Spark サービスのアカウントとロール

Spark ドライバーポッドには、ポッドと同じ名前空間にある Kubernetes サービスアカウントが必要です。このサービスアカウントには、エグゼキューターポッドの作成、取得、一覧表示、パッチ適用、削除を行う権限と、ドライバー用の Kubernetes ヘッドレスサービスを作成する権限が必要です。ポッドの名前空間のデフォルトサービスアカウントに必要な権限がない限り、ドライバーは失敗し、サービスアカウントなしで終了します。

このサービスアカウントのデフォルト名は `emr-containers-sa-spark` です。

このサービスロールには、以下の規則が適用されます。

```
 rules:
- apiGroups:
  - ""
  resources:
  - pods
  verbs:
  - "*"
- apiGroups:
  - ""
  resources:
  - services
  verbs:
  - "*"
- apiGroups:
  - ""
  resources:
  - configmaps
  verbs:
  - "*"
- apiGroups:
  - ""
  resources:
  - persistentvolumeclaims
  verbs:
  - "*"
```

サービスアカウント用 IAM ロール (IRSA、IAM roles for service accounts) でクラスターアクセス権限を設定する

このセクションでは、例を使用して、 AWS Identity and Access Management ロールを引き受けるように Kubernetes サービスアカウントを設定する方法を示します。サービスアカウントを使用するポッドは、ロールがアクセス許可を持つ任意の AWS サービスにアクセスできます。

以下の例では、Spark アプリケーションを実行して Amazon S3 内のファイルの単語をカウントします。これを行うには、サービスアカウント用 IAM ロール (IRSA、IAM roles for service accounts) を設定して、Kubernetes サービスアカウントを認証および認可します。

> **Note**
>
> この例では、Spark 演算子と Spark アプリケーションを送信する名前空間に「spark-operator」名前空間を使用しています。

前提条件

このページの例を試す前に、以下の前提条件を完了します。

- Spark 演算子のセットアップを行う。
- Spark 演算子をインストールする.
- Amazon S3 バケットを作成する。
- お気に入りの詩を `poem.txt` という名前のテキストファイルに保存し、このファイルを S3 バケットにアップロードします。このページで作成した Spark アプリケーションは、テキストファイルの内容を読み取ります。S3 にファイルをアップロードする際の詳細については、「Amazon Simple Storage Service ユーザーガイド」の「バケットにオブジェクトをアップロードする」を参照してください。

IAM ロールを引き受けるように Kubernetes サービスアカウントを設定する

次のステップを使用して、ポッドがアクセス許可を持つサービスにアクセスするために使用できる IAM ロールを引き受けるように Kubernetes AWS サービスアカウントを設定します。

1. を完了したら前提条件、 AWS Command Line Interface を使用して、Amazon S3 にアップロードした`example-policy.json`ファイルへの読み取り専用アクセスを許可する ファイルを作成します。

```
cat >example-policy.json <<EOF
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::my-pod-bucket",
                "arn:aws:s3:::my-pod-bucket/*"
            ]
        }
    ]
}
EOF
```

2. 続いて、IAM ポリシー example-policy を作成します。

```
aws iam create-policy --policy-name example-policy --policy-document file://
example-policy.json
```

3. 次に、IAM ロール example-role を作成し、それを Spark ドライバーの Kubernetes サービスアカウントに関連付けます。

```
eksctl create iamserviceaccount --name driver-account-sa --namespace spark-operator
 \
--cluster my-cluster --role-name "example-role" \
--attach-policy-arn arn:aws:iam::111122223333:policy/example-policy --approve
```

4. Spark ドライバーサービスアカウントに必要なクラスターロールバインディングを含む yaml ファイルを作成します。

```
cat >spark-rbac.yaml <<EOF
apiVersion: v1
kind: ServiceAccount
metadata:
  name: driver-account-sa
---
apiVersion: rbac.authorization.k8s.io/v1
```

```
kind: ClusterRoleBinding
metadata:
  name: spark-role
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: edit
subjects:
  - kind: ServiceAccount
    name: driver-account-sa
    namespace: spark-operator
EOF
```

5. クラスターロールバインディング設定を適用します。

```
kubectl apply -f spark-rbac.yaml
```

kubectl コマンドを実行すると、アカウントが正常に作成されたことを確認できます。

```
serviceaccount/driver-account-sa created
clusterrolebinding.rbac.authorization.k8s.io/spark-role configured
```

Spark 演算子からアプリケーションを実行する

Kubernetes サービスアカウントを設定すると、前提条件 の一部としてアップロードしたテキストファイル内の単語数をカウントする Spark アプリケーションを実行できます。

1. ワードカウントアプリケーションの SparkApplication 定義を含む新しいファイル word-count.yaml を作成します。

```
cat >word-count.yaml <<EOF
apiVersion: "sparkoperator.k8s.io/v1beta2"
kind: SparkApplication
metadata:
  name: word-count
  namespace: spark-operator
spec:
  type: Java
  mode: cluster
  image: "895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-6.10.0:latest"
  imagePullPolicy: Always
```

```
  mainClass: org.apache.spark.examples.JavaWordCount
  mainApplicationFile: local:///usr/lib/spark/examples/jars/spark-examples.jar
  arguments:
    - s3://my-pod-bucket/poem.txt
  hadoopConf:
   # EMRFS filesystem
    fs.s3.customAWSCredentialsProvider:
 com.amazonaws.auth.WebIdentityTokenCredentialsProvider
    fs.s3.impl: com.amazon.ws.emr.hadoop.fs.EmrFileSystem
    fs.AbstractFileSystem.s3.impl: org.apache.hadoop.fs.s3.EMRFSDelegate
    fs.s3.buffer.dir: /mnt/s3
    fs.s3.getObject.initialSocketTimeoutMilliseconds: "2000"

 mapreduce.fileoutputcommitter.algorithm.version.emr_internal_use_only.EmrFileSystem:
 "2"
    mapreduce.fileoutputcommitter.cleanup-
failures.ignored.emr_internal_use_only.EmrFileSystem: "true"
  sparkConf:
    # Required for EMR Runtime
    spark.driver.extraClassPath: /usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/
hadoop-aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/emr/emrfs/conf:/usr/
share/aws/emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/auxlib/*:/usr/share/aws/emr/
security/conf:/usr/share/aws/emr/security/lib/*:/usr/share/aws/hmclient/lib/aws-
glue-datacatalog-spark-client.jar:/usr/share/java/Hive-JSON-Serde/hive-openx-
serde.jar:/usr/share/aws/sagemaker-spark-sdk/lib/sagemaker-spark-sdk.jar:/home/
hadoop/extrajars/*
    spark.driver.extraLibraryPath: /usr/lib/hadoop/lib/native:/usr/lib/hadoop-lzo/
lib/native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/lib/native
    spark.executor.extraClassPath: /usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/
hadoop-aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/emr/emrfs/conf:/usr/
share/aws/emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/auxlib/*:/usr/share/aws/emr/
security/conf:/usr/share/aws/emr/security/lib/*:/usr/share/aws/hmclient/lib/aws-
glue-datacatalog-spark-client.jar:/usr/share/java/Hive-JSON-Serde/hive-openx-
serde.jar:/usr/share/aws/sagemaker-spark-sdk/lib/sagemaker-spark-sdk.jar:/home/
hadoop/extrajars/*
    spark.executor.extraLibraryPath: /usr/lib/hadoop/lib/native:/usr/lib/hadoop-
lzo/lib/native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/lib/
native
  sparkVersion: "3.3.1"
  restartPolicy:
    type: Never
  driver:
    cores: 1
    coreLimit: "1200m"
```

```
    memory: "512m"
    labels:
      version: 3.3.1
    serviceAccount: my-spark-driver-sa
  executor:
    cores: 1
    instances: 1
    memory: "512m"
    labels:
      version: 3.3.1
EOF
```

2. Spark アプリケーションを送信します。

```
kubectl apply -f word-count.yaml
```

kubectl コマンドは、word-count という SparkApplication オブジェクトが正常に作成されたことを示す確認を返します。

```
sparkapplication.sparkoperator.k8s.io/word-count configured
```

3. SparkApplication オブジェクトのイベントを確認するには、以下のコマンドを実行します。

```
kubectl describe sparkapplication word-count -n spark-operator
```

kubectl コマンドは、SparkApplication の説明とイベントを返します。

```
Events:
  Type     Reason                                Age                    From
    Message
  ----     ------                                ----                   ----
    -------
  Normal   SparkApplicationSpecUpdateProcessed  3m2s (x2 over 17h)     spark-
operator  Successfully processed spec update for SparkApplication word-count
  Warning  SparkApplicationPendingRerun          3m2s (x2 over 17h)     spark-
operator  SparkApplication word-count is pending rerun
  Normal   SparkApplicationSubmitted            2m58s (x2 over 17h)    spark-
operator  SparkApplication word-count was submitted successfully
  Normal   SparkDriverRunning                   2m56s (x2 over 17h)    spark-
operator  Driver word-count-driver is running
  Normal   SparkExecutorPending                 2m50s                  spark-
operator  Executor [javawordcount-fdd1698807392c66-exec-1] is pending
```

```
  Normal   SparkExecutorRunning                2m48s                  spark-
operator  Executor [javawordcount-fdd1698807392c66-exec-1] is running
  Normal   SparkDriverCompleted                2m31s (x2 over 17h)    spark-
operator  Driver word-count-driver completed
  Normal   SparkApplicationCompleted           2m31s (x2 over 17h)    spark-
operator  SparkApplication word-count completed
  Normal   SparkExecutorCompleted              2m31s (x2 over 2m31s)  spark-
operator  Executor [javawordcount-fdd1698807392c66-exec-1] completed
```

アプリケーションが S3 ファイル内の単語数をカウントするようになりました。単語数を確認するには、ドライバーのログファイルを参照してください。

```
kubectl logs pod/word-count-driver -n spark-operator
```

kubectl コマンドは、ワードカウントアプリケーションの結果を含むログファイルの内容を返します。

```
INFO DAGScheduler: Job 0 finished: collect at JavaWordCount.java:53, took 5.146519 s
               Software: 1
```

Spark 演算子を使用して Spark にアプリケーションを送信する方法の詳細については、GitHub の Apache Spark 用 Kubernetes 演算子 (spark-on-k8s-operator) ドキュメントの「Using a SparkApplication」を参照してください。

spark-submit を使用して Spark ジョブを実行する

Amazon EMR リリース 6.10.0 以降では、Spark アプリケーションを Amazon EMR on EKS クラスターに送信して実行するために使用できるコマンドラインツールとして `spark-submit` がサポートされています。

> **Note**
>
> Amazon EMR は、vCPU とメモリ消費量に基づいて Amazon EKS の料金を算出します。この計算は、ドライバーポッドとエグゼキューターポッドに適用されます。この計算は、Amazon EMR アプリケーションイメージをダウンロードしてから Amazon EKS ポッドが終了するまでに開始され、秒単位で四捨五入されます。

トピック

Amazon EMR on EKS での spark-submit のセットアップ

Amazon EMR on EKS で spark-submit を使用してアプリケーションを実行する前に、以下のタスクを完了してセットアップを行います。Amazon Web Services (AWS) に既にサインアップしていて、Amazon EKS を既に使用している場合、Amazon EMR on EKS を使用する準備はほぼ整っています。前提条件のいずれかを既に完了している場合は、その前提条件をスキップして、次の前提条件に進むことができます。

- の最新バージョンをインストールまたは更新する AWS CLI – を既にインストールしている場合は AWS CLI、最新バージョンがあることを確認します。
- kubectl と eksctl の設定 — eksctl は、Amazon EKS との通信に使用するコマンドラインツールです。
- Amazon EKS – eksctl の使用開始 – Amazon EKS にノードを持つ新しい Kubernetes クラスターを作成する手順に従います。
- Amazon EMR ベースイメージ URI (リリース 6.10.0 以上) を選択する – `spark-submit` コマンドは Amazon EMR リリース 6.10.0 以降でサポートされています。
- ドライバーサービスアカウントにエグゼキューターポッドを作成および監視するための適切な権限があることを確認します。詳細については、「spark-submit の Spark ドライバーサービスアカウントセキュリティ要件を確認する」を参照してください。
- ローカルの AWS 認証情報プロファイルをセットアップします。
- Amazon EKS コンソールから EKS クラスターを選択し、[概要] [詳細]、[API サーバーエンドポイント] にある EKS クラスターエンドポイントを見つけます。

Amazon EMR on EKS で spark-submit の使用を開始する

Amazon EMR 6.10.0 以降では、Amazon EKS クラスターで Spark アプリケーションを実行するための spark-submit がサポートされています。次のセクションでは、Spark アプリケーションのコマンドを送信する方法を示します。

Spark アプリケーションの実行

Spark アプリケーションを実行するには、以下の手順に従います。

1. `spark-submit` コマンドで Spark アプリケーションを実行する前に、「Amazon EMR on EKS での spark-submit のセットアップ」のステップを完了してください。
2. Amazon EMR on EKS ベースイメージを使用してコンテナを実行します。詳細については、「ベースイメージ URI を選択する方法」を参照してください。

```
kubectl run -it containerName --image=EMRonEKSImage --command -n namespace /bin/
bash
```

3. 次の環境変数の値を設定します。

```
export SPARK_HOME=spark-home
export MASTER_URL=k8s://Amazon EKS-cluster-endpoint
```

4. 次に、以下のコマンドを使用して、Spark アプリケーションを送信します。

```
$SPARK_HOME/bin/spark-submit \
 --class org.apache.spark.examples.SparkPi \
 --master $MASTER_URL \
 --conf spark.kubernetes.container.image=895885662937.dkr.ecr.us-
west-2.amazonaws.com/spark/emr-6.10.0:latest \
 --conf spark.kubernetes.authenticate.driver.serviceAccountName=spark \
 --deploy-mode cluster \
 --conf spark.kubernetes.namespace=spark-operator \
 local:///usr/lib/spark/examples/jars/spark-examples.jar 20
```

Spark へのアプリケーションの送信の詳細については、Apache Spark ドキュメントの「Submitting applications」を参照してください。

> **Important**
>
> `spark-submit` は送信メカニズムとしてクラスターモードのみをサポートします。

spark-submit の Spark ドライバーサービスアカウントセキュリティ要件を確認する

Spark ドライバーポッドは Kubernetes サービスアカウントを使用して Kubernetes API サーバーにアクセスし、エグゼキューターポッドを作成および監視します。ドライバーサービスアカウントには、クラスター内のポッドの一覧表示、作成、編集、パッチ適用、削除を行うために適切な権限が必要です。以下のコマンドを実行して、リソースを一覧表示できることを確認できます。

```
kubectl auth can-i list|create|edit|delete|patch pods
```

各コマンドを実行して必要な権限があることを確認します。

```
kubectl auth can-i list pods
kubectl auth can-i create pods
kubectl auth can-i edit pods
kubectl auth can-i delete pods
kubectl auth can-i patch pods
```

このサービスロールには、以下の規則が適用されます。

```
 rules:
- apiGroups:
  - ""
  resources:
  - pods
  verbs:
  - "*"
- apiGroups:
  - ""
  resources:
  - services
  verbs:
  - "*"
- apiGroups:
  - ""
  resources:
  - configmaps
  verbs:
  - "*"
- apiGroups:
```

```
  - ""
  resources:
  - persistentvolumeclaims
  verbs:
  - "*"
```

spark-submit のサービスアカウント (IRSA) に IAM ロールを設定する

以下のセクションでは、Amazon S3 に保存されている Spark アプリケーションを実行できるように、サービスアカウント (IRSA) の IAM ロールを設定して Kubernetes サービスアカウントを認証および承認する方法について説明します。

前提条件

このドキュメントの例を試す前に、次の前提条件を満たしていることを確認してください。

- spark-submit の設定が完了している
- S3 バケットを作成し、Spark アプリケーション jar をアップロードしている

IAM ロールを引き受けるための Kubernetes サービスアカウントの設定

次の手順では、 AWS Identity and Access Management (IAM) ロールを引き受けるように Kubernetes サービスアカウントを設定する方法について説明します。サービスアカウントを使用するようにポッドを設定した後、ロール AWS のサービス がアクセス許可を持つ任意の にアクセスできます。

1. アップロードした Amazon S3 オブジェクトへの読み取り専用アクセスを許可するポリシーファイルを作成します。

```
cat >my-policy.json <<EOF
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::<my-spark-jar-bucket>",
                "arn:aws:s3:::<my-spark-jar-bucket>/*"
```

```
            ]
        }
    ]
}
EOF
```

2. IAM ポリシーを作成します。

```
aws iam create-policy --policy-name my-policy --policy-document file://my-
policy.json
```

3. IAM ロールを作成し、それを Spark ドライバーの Kubernetes サービスアカウントに関連付けます。

```
eksctl create iamserviceaccount --name my-spark-driver-sa --namespace spark-
operator \
--cluster my-cluster --role-name "my-role" \
--attach-policy-arn arn:aws:iam::111122223333:policy/my-policy --approve
```

4. Spark ドライバーのサービスアカウントに必要なアクセス権限を持つ YAML ファイルを作成します。

```
cat >spark-rbac.yaml <<EOF
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: emr-containers-role-spark
rules:
- apiGroups:
  - ""
  resources:
  - pods
  verbs:
  - "*"
- apiGroups:
  - ""
  resources:
  - services
  verbs:
  - "*"
- apiGroups:
  - ""
```

```
  resources:
  - configmaps
  verbs:
  - "*"
- apiGroups:
  - ""
  resources:
  - persistentvolumeclaims
  verbs:
  - "*"
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: spark-role-binding
  namespace: default
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: emr-containers-role-spark
subjects:
- kind: ServiceAccount
  name: emr-containers-sa-spark
  namespace: default
EOF
```

5. クラスターロールバインディング設定を適用します。

```
kubectl apply -f spark-rbac.yaml
```

6. kubectl コマンドは、作成されたアカウントの確認を返します。

```
serviceaccount/emr-containers-sa-spark created
clusterrolebinding.rbac.authorization.k8s.io/emr-containers-role-spark configured
```

Spark アプリケーションの実行

Amazon EMR 6.10.0 以降では、Amazon EKS クラスターで Spark アプリケーションを実行するための spark-submit がサポートされています。Spark アプリケーションを実行するには、以下の手順に従います。

1. Amazon EMR on EKS の spark-submit 設定の手順が完了していることを確認してください。

2. 次の環境変数の値を設定します。

```
export SPARK_HOME=spark-home
export MASTER_URL=k8s://Amazon EKS-cluster-endpoint
```

3. 次に、以下のコマンドを使用して、Spark アプリケーションを送信します。

```
$SPARK_HOME/bin/spark-submit \
 --class org.apache.spark.examples.SparkPi \
 --master $MASTER_URL \
 --conf spark.kubernetes.container.image=895885662937.dkr.ecr.us-
west-2.amazonaws.com/spark/emr-6.15.0:latest \
 --conf spark.kubernetes.authenticate.driver.serviceAccountName=emr-containers-sa-
spark \
 --deploy-mode cluster \
 --conf spark.kubernetes.namespace=default \
 --conf "spark.driver.extraClassPath=/usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/
hadoop-aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/emr/emrfs/conf:/usr/
share/aws/emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/auxlib/*:/usr/share/aws/emr/
security/conf:/usr/share/aws/emr/security/lib/*:/usr/share/aws/hmclient/lib/aws-
glue-datacatalog-spark-client.jar:/usr/share/java/Hive-JSON-Serde/hive-openx-
serde.jar:/usr/share/aws/sagemaker-spark-sdk/lib/sagemaker-spark-sdk.jar:/home/
hadoop/extrajars/*" \
 --conf "spark.driver.extraLibraryPath=/usr/lib/hadoop/lib/native:/usr/lib/hadoop-
lzo/lib/native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/lib/
native" \
 --conf "spark.executor.extraClassPath=/usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/
hadoop-aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/emr/emrfs/conf:/usr/
share/aws/emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/auxlib/*:/usr/share/aws/emr/
security/conf:/usr/share/aws/emr/security/lib/*:/usr/share/aws/hmclient/lib/aws-
glue-datacatalog-spark-client.jar:/usr/share/java/Hive-JSON-Serde/hive-openx-
serde.jar:/usr/share/aws/sagemaker-spark-sdk/lib/sagemaker-spark-sdk.jar:/home/
hadoop/extrajars/*" \
 --conf "spark.executor.extraLibraryPath=/usr/lib/hadoop/lib/native:/usr/lib/
hadoop-lzo/lib/native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/
lib/native" \
 --conf
 spark.hadoop.fs.s3.customAWSCredentialsProvider=com.amazonaws.auth.WebIdentityTokenCredent
 \
 --conf spark.hadoop.fs.s3.impl=com.amazon.ws.emr.hadoop.fs.EmrFileSystem \
 --conf
 spark.hadoop.fs.AbstractFileSystem.s3.impl=org.apache.hadoop.fs.s3.EMRFSDelegate \
 --conf spark.hadoop.fs.s3.buffer.dir=/mnt/s3 \
```

```
 --conf spark.hadoop.fs.s3.getObject.initialSocketTimeoutMilliseconds="2000" \
 --conf
 spark.hadoop.mapreduce.fileoutputcommitter.algorithm.version.emr_internal_use_only.EmrFile
 \
 --conf spark.hadoop.mapreduce.fileoutputcommitter.cleanup-
failures.ignored.emr_internal_use_only.EmrFileSystem="true" \
 s3://my-pod-bucket/spark-examples.jar 20
```

4. スパークドライバーが Spark ジョブを完了すると、Spark ジョブが完了したことを示すログラインが送信の最後に表示されます。

```
23/11/24 17:02:14 INFO LoggingPodStatusWatcherImpl: Application
 org.apache.spark.examples.SparkPi with submission ID default:org-apache-spark-
examples-sparkpi-4980808c03ff3115-driver finished
23/11/24 17:02:14 INFO ShutdownHookManager: Shutdown hook called
```

クリーンアップ

アプリケーションの実行が完了したら、次のコマンドを使用してクリーンアップを実行できます。

```
kubectl delete -f spark-rbac.yaml
```

Amazon EMR on EKS での Apache Livy の使用

Amazon EMR リリース 7.1.0 以降では、Apache Livy を使用して Amazon EMR on EKS でジョブを送信できます。Apache Livy を使用すると、独自の Apache Livy REST エンドポイントを設定して Amazon EKS クラスターに Spark アプリケーションをデプロイし、管理できます。Amazon EKS クラスターに Livy をインストールすると、Livy エンドポイントを使用して Spark アプリケーションを Livy サーバーに送信できます。サーバーは Spark アプリケーションのライフサイクルを管理します。

> **Note**
>
> Amazon EMR は、vCPU とメモリ消費量に基づいて Amazon EKS の料金を算出します。この計算は、ドライバーポッドとエグゼキューターポッドに適用されます。この計算は、Amazon EMR アプリケーションイメージをダウンロードしてから Amazon EKS ポッドが終了するまでに開始され、秒単位で四捨五入されます。

トピック

Amazon EMR on EKS 用の Apache Livy の設定

Amazon EKS クラスターに Apache Livy をインストールする前に、必須となるツールセットをインストールして設定する必要があります。これらには AWS CLI、 AWS リソースを操作するための基本的なコマンドラインツールである 、Amazon EKS を操作するためのコマンドラインツール、およびこのユースケースでクラスターアプリケーションをインターネットで利用可能にし、ネットワークトラフィックをルーティングするために使用されるコントローラーが含まれます。

- の最新バージョンをインストールまたは更新する AWS CLI – を既にインストールしている場合は AWS CLI、最新バージョンであることを確認します。
- kubectl と eksctl の設定 — eksctl は、Amazon EKS との通信に使用するコマンドラインツールです。
- Helm のインストール – Kubernetes 用の Helm パッケージマネージャーを使用すると、Kubernetes クラスターにアプリケーションをインストールして管理できます。
- Amazon EKS – eksctl の使用開始 – Amazon EKS にノードを持つ新しい Kubernetes クラスターを作成する手順に従います。
- Amazon EMR リリースラベルの選択 – Apache Livy は Amazon EMR リリース 7.1.0 以降でサポートされています。
- ALB コントローラーをインストールする – ALB コントローラーは AWS Elastic Load Balancing for Kubernetes クラスターを管理します。Apache Livy の設定中に Kubernetes Ingress を作成すると、 AWS Network Load Balancer (NLB) が作成されます。

Amazon EMR on EKS での Apache Livy の使用開始

Apache Livy をインストールするには、次の手順を実行します。これには、パッケージマネージャーの設定、Spark ワークロード実行用の名前空間の作成、Livy のインストール、負荷分散の設定、検証ステップが含まれます。Spark でバッチジョブを実行するには、これらの手順を実行する必要があります。

1. まだ設定していない場合は、Amazon EMR on EKS 用の Apache Livy を設定します。
2. Amazon ECR レジストリに対し、Helm クライアントを認証します。に対応するECR-registry-account値は、Amazon ECR レジストリアカウント AWS リージョン からリージョン別に確認できます。 https://docs.aws.amazon.com/emr/latest/EMR-on-EKS-DevelopmentGuide/docker-custom-images-tag.html#docker-custom-images-ECR

   ```
   aws ecr get-login-password \--region <AWS_REGION> | helm registry login \
   --username AWS \
   --password-stdin <ECR-registry-account>.dkr.ecr.<region-id>.amazonaws.com
   ```

3. Livy を設定すると、Livy サーバーのサービスアカウントと Spark アプリケーション用にもう一つのアカウントが作成されます。サービスアカウントの IRSA を設定するには、「サービスアカウント用 IAM ロール (IRSA、IAM roles for service accounts) でアクセス権限を設定する」を参照してください。
4. Spark ワークロードを実行する名前空間を作成します。

   ```
   kubectl create ns <spark-ns>
   ```

5. 次のコマンドを使用して Livy をインストールします。

   この Livy エンドポイントは、EKS クラスターの VPC でのみ内部的に使用できます。VPC 以外のアクセスを有効にするには、Helm インストールコマンドで `--set loadbalancer.internal=false` を設定します。

   > **Note**
   >
   > デフォルトでは、SSL はこの Livy エンドポイント内で有効ではなく、エンドポイントは EKS クラスターの VPC 内でのみ表示されます。`loadbalancer.internal=false` と `ssl.enabled=false` を設定すると、安全でないエンドポイントが VPC の外部に公開されます。安全な Livy エンドポイントを設定するには、「TLS/SSL を使用した安全な Apache Livy エンドポイントの設定」を参照してください。

```
helm install livy-demo \
  oci://895885662937.dkr.ecr.region-id.amazonaws.com/livy \
  --version 7.7.0 \
  --namespace livy-ns \
  --set image=ECR-registry-account.dkr.ecr.region-id.amazonaws.com/livy/
emr-7.7.0:latest \
  --set sparkNamespace=<spark-ns> \
  --create-namespace
```

次のような出力が表示されます。

```
NAME: livy-demo
LAST DEPLOYED: Mon Mar 18 09:23:23 2024
NAMESPACE: livy-ns
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
The Livy server has been installed.
Check installation status:
1. Check Livy Server pod is running
  kubectl --namespace livy-ns get pods -l "app.kubernetes.io/instance=livy-demo"
2. Verify created NLB is in Active state and it's target groups are healthy (if
 loadbalancer.enabled is true)

Access LIVY APIs:
    # Ensure your NLB is active and healthy
    # Get the Livy endpoint using command:
    LIVY_ENDPOINT=$(kubectl get svc -n livy-ns -l app.kubernetes.io/
instance=livy-demo,emr-containers.amazonaws.com/type=loadbalancer -o
 jsonpath='{.items[0].status.loadBalancer.ingress[0].hostname}' |  awk '{printf
 "%s:8998\n", $0}')
    # Access Livy APIs using http://$LIVY_ENDPOINT or https://$LIVY_ENDPOINT (if
 SSL is enabled)
    # Note: While uninstalling Livy, makes sure the ingress and NLB are deleted
 after running the helm command to avoid dangling resources
```

Livy サーバーと Spark セッションのデフォルトのサービスアカウント名は emr-containers-sa-livyと emr-containers-sa-spark-livy です。カスタム名を使用するに

は、serviceAccounts.name パラメータと sparkServiceAccount.name パラメータを使用します。

```
--set serviceAccounts.name=my-service-account-for-livy
--set sparkServiceAccount.name=my-service-account-for-spark
```

6. Helm チャートをインストールしたことを確認します。

```
helm list -n livy-ns -o yaml
```

helm list コマンドは、新しい Helm チャートに関する情報を返します。

```
app_version: 0.7.1-incubating
chart: livy-emr-7.7.0
name: livy-demo
namespace: livy-ns
revision: "1"
status: deployed
updated: 2024-02-08 22:39:53.539243 -0800 PST
```

7. Network Load Balancer がアクティブであることを確認します。

```
LIVY_NAMESPACE=<livy-ns>
LIVY_APP_NAME=<livy-app-name>
AWS_REGION=<AWS_REGION>

# Get the NLB Endpoint URL
NLB_ENDPOINT=$(kubectl --namespace $LIVY_NAMESPACE get svc -l "app.kubernetes.io/
instance=$LIVY_APP_NAME,emr-containers.amazonaws.com/type=loadbalancer" -o
 jsonpath='{.items[0].status.loadBalancer.ingress[0].hostname}')

# Get all the load balancers in the account's region
ELB_LIST=$(aws elbv2 describe-load-balancers --region $AWS_REGION)

# Get the status of the NLB that matching the endpoint from the Kubernetes service
NLB_STATUS=$(echo $ELB_LIST | grep -A 8 "\"DNSName\": \"$NLB_ENDPOINT\"" | awk '/
Code/{print $2}/}/' | tr -d '"},\n')
echo $NLB_STATUS
```

8. 次に、Network Load Balancer のターゲットグループが正常であることを確認します。

```
LIVY_NAMESPACE=<livy-ns>
```

```
LIVY_APP_NAME=<livy-app-name>
AWS_REGION=<AWS_REGION>

# Get the NLB endpoint
NLB_ENDPOINT=$(kubectl --namespace $LIVY_NAMESPACE get svc -l "app.kubernetes.io/
instance=$LIVY_APP_NAME,emr-containers.amazonaws.com/type=loadbalancer" -o
 jsonpath='{.items[0].status.loadBalancer.ingress[0].hostname}')

# Get all the load balancers in the account's region
ELB_LIST=$(aws elbv2 describe-load-balancers --region $AWS_REGION)

# Get the NLB ARN from the NLB endpoint
NLB_ARN=$(echo $ELB_LIST | grep -B 1 "\"DNSName\": \"$NLB_ENDPOINT\"" | awk
 '/"LoadBalancerArn":/,/"/'| awk '/:/{print $2}' | tr -d \",)

# Get the target group from the NLB. Livy setup only deploys 1 target group
TARGET_GROUP_ARN=$(aws elbv2 describe-target-groups --load-balancer-arn $NLB_ARN
 --region $AWS_REGION | awk '/"TargetGroupArn":/,/"/'| awk '/:/{print $2}' | tr -d
 \",)

# Get health of target group
aws elbv2 describe-target-health --target-group-arn $TARGET_GROUP_ARN
```

以下は、ターゲットグループのステータスを示す出力のサンプルです。

```
{
    "TargetHealthDescriptions": [
        {
            "Target": {
                "Id": "<target IP>",
                "Port": 8998,
                "AvailabilityZone": "us-west-2d"
            },
            "HealthCheckPort": "8998",
            "TargetHealth": {
                "State": "healthy"
            }
        }
    ]
}
```

NLB のステータスが active になり、ターゲットグループが healthy になったら、続行できます。これには数分かかることがあります。

9. Helm インストールから Livy エンドポイントを取得します。Livy エンドポイントが安全かどうかは、SSL を有効にしたかどうかで決まります。

```
LIVY_NAMESPACE=<livy-ns>
 LIVY_APP_NAME=livy-app-name
 LIVY_ENDPOINT=$(kubectl get svc -n livy-ns -l app.kubernetes.io/
instance=livy-app-name,emr-containers.amazonaws.com/type=loadbalancer -o
 jsonpath='{.items[0].status.loadBalancer.ingress[0].hostname}' |  awk '{printf
 "%s:8998\n", $0}')
 echo "$LIVY_ENDPOINT"
```

10. Helm インストールから Spark サービスアカウントを取得する

```
SPARK_NAMESPACE=spark-ns
LIVY_APP_NAME=<livy-app-name>
SPARK_SERVICE_ACCOUNT=$(kubectl --namespace $SPARK_NAMESPACE
 get sa -l "app.kubernetes.io/instance=$LIVY_APP_NAME" -o
 jsonpath='{.items[0].metadata.name}')
echo "$SPARK_SERVICE_ACCOUNT"
```

次のような出力が表示されます。

```
emr-containers-sa-spark-livy
```

11. VPC の外部からのアクセスを可能にするように internalALB=true を設定した場合は、Amazon EC2 インスタンスを作成し、Network Load Balancer が EC2 インスタンスからのネットワークトラフィックを許可していることを確認します。インスタンスが Livy エンドポイントにアクセスするには、これが必要となります。VPC の外部にエンドポイントを安全に公開する方法の詳細については、「TLS/SSL を使用した安全な Apache Livy エンドポイントの設定」を参照してください。

12. Livy をインストールすると、Spark アプリケーションを実行するサービスアカウント emr-containers-sa-spark が作成されます。Spark アプリケーションが S3 などの AWS リソースを使用している場合、または AWS API または CLI オペレーションを呼び出す場合は、IAM ロールを spark サービスアカウントに必要なアクセス許可にリンクする必要があります。詳細に

ついては、「サービスアカウント用 IAM ロール (IRSA、IAM roles for service accounts) でアクセス権限を設定する」を参照してください。

Apache Livy は、Livy のインストール時に使用できる追加の設定をサポートしています。詳細については、「Amazon EMR on EKS リリースでの Apache Livy のインストールプロパティ」を参照してください。

Amazon EMR on EKS の Apache Livy による Spark アプリケーションの実行

Apache Livy で Spark アプリケーションを実行する前に、Amazon EMR on EKS 用の Apache Livy の設定と、Amazon EMR on EKS での Apache Livy の使用開始の手順が完了していることを確認してください。

Apache Livy を使用して、次の 2 種類のアプリケーションを実行できます。

- バッチセッション – Livy ワークロードの一種で、Spark バッチジョブを送信します。
- インタラクティブセッション – Livy ワークロードの一種で、Spark クエリを実行するためのプログラムインターフェイスとビジュアルインターフェイスを提供します。

> **Note**
>
> さまざまなセッションのドライバーポッドとエグゼキューターポッドが相互に通信できます。名前空間はポッド間のセキュリティを保証しません。Kubernetes では、指定された名前空間内のポッドのサブセットに対する選択的なアクセス権限は許可されません。

バッチセッションの実行

バッチジョブを送信するには、次のコマンドを使用します。

```
curl -s -k -H 'Content-Type: application/json' -X POST \
      -d '{
            "name": "my-session",
            "file": "entryPoint_location (S3 or local)",
            "args": ["argument1", "argument2", ...],
            "conf": {
                "spark.kubernetes.namespace": "<spark-namespace>",
```

```
                "spark.kubernetes.container.image": "public.ecr.aws/emr-on-eks/spark/
emr-7.7.0:latest",
                "spark.kubernetes.authenticate.driver.serviceAccountName": "<spark-
service-account>"
            }
          }' <livy-endpoint>/batches
```

バッチジョブをモニタリングするには、次のコマンドを使用します。

```
curl -s -k -H 'Content-Type: application/json' -X GET <livy-endpoint>/batches/my-
session
```

インタラクティブセッションの実行

Apache Livy でインタラクティブセッションを実行するには、次の手順を参照してください。

1. SageMaker AI Jupyter Notebook など、セルフホスト型またはマネージド型の Jupyter Notebook にアクセスできることを確認してください。Jupyter Notebook には sparkmagic がインストールされている必要があります。
2. Spark 設定 `spark.kubernetes.file.upload.path` のバケットを作成します。Spark サービスアカウントにバケットへの読み取りおよび書き込みのアクセス権限があることを確認します。spark サービスアカウントの設定方法の詳細については、「サービスアカウントの IAM ロール (IRSA、IAM roles for service accounts) でアクセス権限を設定する」を参照してください。
3. コマンド `%load_ext sparkmagic.magics` を使用して、Jupyter Notebook に sparkmagic をロードします。
4. コマンド `%manage_spark` を実行して、Jupyter Notebook で Livy エンドポイントを設定します。[エンドポイントを追加] タブを選択して設定された認証タイプを選択し、ノートブックに Livy エンドポイントを追加して [エンドポイントを追加] を選択します。
5. `%manage_spark` を再度実行して Spark コンテキストを作成し、[セッションを作成] に移動します。Livy エンドポイントを選択し、一意のセッション名を指定して言語を選択し、次のプロパティを追加します。

   ```
   {
     "conf": {
       "spark.kubernetes.namespace": "livy-namespace",
       "spark.kubernetes.container.image": "public.ecr.aws/emr-on-eks/spark/
   emr-7.7.0:latest",
   ```

```
    "spark.kubernetes.authenticate.driver.serviceAccountName": "<spark-service-
account>",
    "spark.kubernetes.file.upload.path": "<URI_TO_S3_LOCATION_>"
  }
}
```

6. アプリケーションを送信して Spark コンテキストが作成されるのを待ちます。
7. インタラクティブセッションのステータスをモニタリングするには、次のコマンドを実行します。

```
curl -s -k -H 'Content-Type: application/json' -X GET livy-endpoint/sessions/my-
interactive-session
```

Spark アプリケーションのモニタリング

Livy UI を使用して Spark アプリケーションの進捗状況をモニタリングするには、リンク `http://<livy-endpoint>/ui` を使用します。

Amazon EMR on EKS を使用した Apache Livy のアンインストール

Apache Livy をアンインストールするには、次の手順に従ってください。

1. 名前空間名とアプリケーション名を使用して Livy の設定を削除します。この例では、アプリケーション名は `livy-demo` で、名前空間は `livy-ns` です。

```
helm uninstall livy-demo -n livy-ns
```

2. アンインストールすると、Amazon EMR on EKS は Livy の Kubernetes サービス、 AWS ロードバランサー、およびインストール中に作成したターゲットグループを削除します。リソースの削除には数分かかる場合があります。名前空間に Livy を再インストールする前に、リソースが削除されていることを確認してください。
3. Spark 名前空間を削除します。

```
kubectl delete namespace spark-ns
```

Amazon EMR on EKS を使用した Apache Livy のセキュリティ

Amazon EMR on EKS で Apache Livy のセキュリティを設定する方法の詳細については、以下のトピックを参照してください。これらのオプションには、トランスポートレイヤーセキュリティ、ロールベースのアクセス制御 (組織内のユーザーロールに基づいたアクセス)、および IAM ロールの使用 (付与されたアクセス権限に基づいたリソースへのアクセスの提供) が含まれます。

トピック

- TLS/SSL を使用した安全な Apache Livy エンドポイントの設定
- ロールベースのアクセスコントロール (RBAC) による Apache Livy および Spark のアプリケーションアクセス権限の設定
- サービスアカウント用 IAM ロール (IRSA、IAM roles for service accounts) でアクセス権限を設定する

TLS/SSL を使用した安全な Apache Livy エンドポイントの設定

エンドツーエンドの TLS および SSL 暗号化を使用した Amazon EMR on EKS 用の Apache Livy の設定について詳しくは、以下のセクションを参照してください。

TLS および SSL 暗号化の設定

Apache Livy エンドポイントで SSL 暗号化を設定するには、次の手順に従います。

- Secrets Store CSI Driver and AWS Secrets and Configuration Provider (ASCP) をインストールする – Secrets Store CSI Driver and ASCP は、Livy サーバーポッドが SSL を有効にするために必要な Livy の JKS 証明書とパスワードを安全に保存します。Secrets Store CSI Driver のみをインストールし、サポートされている他のシークレットプロバイダーを使用することもできます。
- ACM 証明書の作成 – この証明書は、クライアントと ALB エンドポイント間の接続を保護するために必要です。
- ALB エンドポイントと Livy AWS Secrets Manager サーバー間の接続を保護するために必要な JKS 証明書、キーパスワード、およびキーストアパスワードを に設定します。
- Livy サービスアカウントにアクセス許可を追加してシークレットを取得する AWS Secrets Manager – Livy サーバーには、ASCP からシークレットを取得し、Livy サーバーを保護するために Livy 設定を追加するためのアクセス許可が必要です。サービスアカウントに IAM アクセス権限を追加するには、「サービスアカウント用 IAM ロール (IRSA、IAM roles for service accounts) でアクセス権限を設定する」を参照してください。

のキーとキーストアパスワードを使用して JKS 証明書を設定する AWS Secrets Manager

キーとキーストアパスワードを使用して JKS 証明書を設定するには、次の手順に従います。

1. Livy サーバーのキーストアファイルを生成します。

```
keytool -genkey -alias <host> -keyalg RSA -keysize 2048 -dname
 CN=<host>,OU=hw,O=hw,L=<your_location>,ST=<state>,C=<country> -
keypass <keyPassword> -keystore <keystore_file> -storepass <storePassword> --
validity 3650
```

2. 証明書を作成します。

```
keytool -export -alias <host> -keystore mykeystore.jks -rfc -
file mycertificate.cert -storepass <storePassword>
```

3. トラストストアファイルを作成します。

```
keytool -import -noprompt -alias <host>-file <cert_file> -
keystore <truststore_file> -storepass <truststorePassword>
```

4. JKS 証明書を に保存します AWS Secrets Manager。`livy-jks-secret` をシークレットに置き換え、`fileb://mykeystore.jks` をキーストア JKS 証明書へのパスに置き換えます。

```
aws secretsmanager create-secret \
--name livy-jks-secret \
--description "My Livy keystore JKS secret" \
--secret-binary fileb://mykeystore.jks
```

5. Secrets Manager にキーストアとキーパスワードを保存します。必ず独自のパラメータを使用してください。

```
aws secretsmanager create-secret \
--name livy-jks-secret \
--description "My Livy key and keystore password secret" \
--secret-string "{\"keyPassword\":\"<test-key-password>\",\"keyStorePassword\":
\"<test-key-store-password>\"}"
```

6. 次のコマンドで Livy サーバーの名前空間を作成します。

```
kubectl create ns <livy-ns>
```

7. JKS 証明書とパスワードを持つ Livy サーバーの ServiceProviderClass オブジェクトを作成します。

```
cat >livy-secret-provider-class.yaml << EOF
apiVersion: secrets-store.csi.x-k8s.io/v1
kind: SecretProviderClass
metadata:
  name: aws-secrets
spec:
  provider: aws
  parameters:
    objects: |
        - objectName: "livy-jks-secret"
          objectType: "secretsmanager"
        - objectName: "livy-passwords"
          objectType: "secretsmanager"

EOF
kubectl apply -f livy-secret-provider-class.yaml -n <livy-ns>
```

SSL 対応の Apache Livy の使用開始

Livy サーバーで SSL を有効にしたら、 AWS Secrets Managerで keyStore シークレットと keyPasswords シークレットにアクセスできるように serviceAccount を設定する必要があります。

1. Livy サーバーの名前空間を作成します。

```
kubectl create namespace <livy-ns>
```

2. Secrets Manager のシークレットにアクセスできるように Livy のサービスアカウントを設定します。IRSA のセットアップの詳細については、「Apache Livy のインストール中に IRSA を設定する」を参照してください。

```
aws ecr get-login-password \--region region-id | helm registry login \
--username AWS \
--password-stdin ECR-registry-account.dkr.ecr.region-id.amazonaws.com
```

3. Livy をインストールします。Helm チャートのバージョンパラメータには、7.1.0 などの Amazon EMR リリースラベルを使用します。また、Amazon ECR レジストリのアカウント

ID とリージョン ID を独自の ID に置き換える必要があります。に対応するECR-registry-account値は、Amazon ECR レジストリアカウント AWS リージョン からリージョン別に確認できます。 https://docs.aws.amazon.com/emr/latest/EMR-on-EKS-DevelopmentGuide/docker-custom-images-tag.html#docker-custom-images-ECR

```
helm install <livy-app-name> \
  oci://895885662937.dkr.ecr.region-id.amazonaws.com/livy \
  --version 7.7.0 \
  --namespace livy-namespace-name \
  --set image=<ECR-registry-account.dkr.ecr>.<region>.amazonaws.com/livy/
emr-7.7.0:latest \
  --set sparkNamespace=spark-namespace \
  --set ssl.enabled=true
  --set ssl.CertificateArn=livy-acm-certificate-arn
  --set ssl.secretProviderClassName=aws-secrets
  --set ssl.keyStoreObjectName=livy-jks-secret
  --set ssl.keyPasswordsObjectName=livy-passwords
  --create-namespace
```

4. Amazon EMR on EKS への Apache Livy のインストールのステップ 5 から続けます。

ロールベースのアクセスコントロール (RBAC) による Apache Livy および Spark のアプリケーションアクセス権限の設定

Livy をデプロイするために、Amazon EMR on EKS はサーバーのサービスアカウントとロール、および Spark のサービスアカウントとロールを作成します。これらのロールには、Spark アプリケーションの設定と実行を完了するために必要な RBAC アクセス権限が必要です。

サーバーのサービスアカウントとロールのための RBAC アクセス権限

Amazon EMR on EKS は、Spark ジョブの Livy セッションを管理し、イングレスやその他のリソースとの間でトラフィックをルーティングするための Livy サーバーのサービスアカウントとロールを作成します。

このサービスアカウントのデフォルト名は emr-containers-sa-livy です。次のアクセス権限が必要です。

```
rules:
- apiGroups:
  - ""
  resources:
```

```
  - "namespaces"
  verbs:
  - "get"
- apiGroups:
  - ""
  resources:
  - "serviceaccounts"
    "services"
    "configmaps"
    "events"
    "pods"
    "pods/log"
  verbs:
  - "get"
    "list"
    "watch"
    "describe"
    "create"
    "edit"
    "delete"
    "deletecollection"
    "annotate"
    "patch"
    "label"
 - apiGroups:
   - ""
   resources:
   - "secrets"
   verbs:
   - "create"
     "patch"
     "delete"
     "watch"
 - apiGroups:
   - ""
   resources:
   - "persistentvolumeclaims"
   verbs:
   - "get"
     "list"
     "watch"
     "describe"
     "create"
     "edit"
```

```
    "delete"
    "annotate"
    "patch"
    "label"
```

Spark のサービスアカウントとロールのための RBAC アクセス権限

Spark ドライバーポッドには、ポッドと同じ名前空間にある Kubernetes サービスアカウントが必要です。このサービスアカウントには、エグゼキューターポッドとドライバーポッドに必要なリソースを管理するためのアクセス権限が必要です。名前空間のデフォルトサービスアカウントに必要なアクセス権限がなければ、ドライバーは失敗して終了します。以下の RBAC アクセス権限が必要です。

```
rules:
- apiGroups:
  - ""
    "batch"
    "extensions"
    "apps"
  resources:
  - "configmaps"
    "serviceaccounts"
    "events"
    "pods"
    "pods/exec"
    "pods/log"
    "pods/portforward"
    "secrets"
    "services"
    "persistentvolumeclaims"
    "statefulsets"
  verbs:
  - "create"
    "delete"
    "get"
    "list"
    "patch"
    "update"
    "watch"
    "describe"
    "edit"
    "deletecollection"
    "patch"
```

```
    "label"
```

サービスアカウント用 IAM ロール (IRSA、IAM roles for service accounts) でアクセス権限を設定する

デフォルトでは、Livy サーバーと Spark アプリケーションのドライバーとエグゼキューターは AWS リソースにアクセスできません。サーバーサービスアカウントと spark サービスアカウントは、Livy サーバーと spark アプリケーションのポッドの AWS リソースへのアクセスを制御します。アクセスを許可するには、サービスアカウントを、必要な AWS アクセス許可を持つ IAM ロールにマッピングする必要があります。

Apache Livy のインストール前、インストール中、インストール終了後のいずれにおいても IRSA マッピングを設定できます。

Apache Livy インストール中の IRSA の設定 (サーバーのサービスアカウント用)

> **Note**
>
> このマッピングは、サーバーのサービスアカウントでのみサポートされています。

1. Amazon EMR on EKS の Apache Livy の設定が終了し、Amazon EMR on EKS による Apache Livy のインストールの途中であることを確認します。
2. Livy サーバー用に、Kubernetes 名前空間を作成します。この例では、名前空間の名前は `livy-ns` です。
3. ポッドがアクセス AWS のサービス する のアクセス許可を含む IAM ポリシーを作成します。次の例では、Spark エントリポイントの Amazon S3 リソースを取得する IAM ポリシーを作成します。

```
cat >my-policy.json <<EOF{
"Version": "2012-10-17",
    "Statement": [
        {
"Effect": "Allow",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::my-spark-entrypoint-bucket"
        }
    ]
}
EOF
```

```
aws iam create-policy --policy-name my-policy --policy-document file://my-
policy.json
```

4. 次のコマンドを使用して、 AWS アカウント ID を変数に設定します。

```
account_id=$(aws sts get-caller-identity --query "Account" --output text)
```

5. クラスターの OpenID Connect (OIDC) ID プロバイダーを環境変数に設定します。

```
oidc_provider=$(aws eks describe-cluster --name my-cluster --region $AWS_REGION --
query "cluster.identity.oidc.issuer" --output text | sed -e "s/^https:\/\///")
```

6. サービスアカウントの名前空間と名前の変数を設定します。必ず独自の値を使用してください。

```
export namespace=default
export service_account=my-service-account
```

7. 次のコマンドを使用して信頼ポリシーファイルを作成します。名前空間内のすべての
サービスアカウントにロールへのアクセス権を付与する場合は、次のコマンドをコピーし
て、StringEquals を StringLike に、$service_account を * に置き換えます。

```
cat >trust-relationship.json <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::$account_id:oidc-provider/$oidc_provider"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "$oidc_provider:aud": "sts.amazonaws.com",
          "$oidc_provider:sub": "system:serviceaccount:$namespace:$service_account"
        }
      }
    }
  ]
}
EOF
```

8. ロールを作成します。

```
aws iam create-role --role-name my-role --assume-role-policy-document file://trust-
relationship.json --description "my-role-description"
```

9. 次の Helm インストールコマンドを使用して、serviceAccount.executionRoleArn を IRSA をマッピングするように設定します。Helm インストールコマンドの例を次に示します。に対応するECR-registry-account値は、Amazon ECR レジストリアカウント AWS リージョン からリージョン別に確認できます。 https://docs.aws.amazon.com/emr/latest/EMR-on-EKS-DevelopmentGuide/docker-custom-images-tag.html#docker-custom-images-ECR

```
helm install livy-demo \
  oci://895885662937.dkr.ecr.us-west-2.amazonaws.com/livy \
  --version 7.7.0 \
  --namespace livy-ns \
  --set image=ECR-registry-account.dkr.ecr.region-id.amazonaws.com/livy/
emr-7.7.0:latest \
  --set sparkNamespace=spark-ns \
  --set serviceAccount.executionRoleArn=arn:aws:iam::123456789012:role/my-role
```

IRSA の Spark サービスアカウントへのマッピング

IRSA を Spark サービスアカウントにマッピングする前に、次の項目が完了していることを確認してください。

- Amazon EMR on EKS の Apache Livy の設定が終了し、Amazon EMR on EKS による Apache Livy のインストールの途中であることを確認します。
- クラスターの既存 IAM OpenID Connect (OIDC) プロバイダーが必要です。既にあるかどうか、または作成する方法を確認するには、「クラスターの IAM OIDC プロバイダーを作成する」を参照してください。
- バージョン 0.171.0 以降の eksctl CLI または AWS CloudShellがインストールされていることを確認します。eksctl をインストールまたはアップグレードするには、eksctl ドキュメントの「インストール」を参照してください。

IRSA を Spark サービスアカウントにマッピングするには、次の手順に従います。

1. 次のコマンドを実行して Spark サービスアカウントを取得します。

```
SPARK_NAMESPACE=<spark-ns>
LIVY_APP_NAME=<livy-app-name>
kubectl --namespace $SPARK_NAMESPACE describe sa -l "app.kubernetes.io/instance=
$LIVY_APP_NAME" | awk '/^Name:/ {print $2}'
```

2. サービスアカウントの名前空間と名前の変数を設定します。

```
export namespace=default
export service_account=my-service-account
```

3. IAM ロール用の信頼ポリシーファイルを作成するには、次のコマンドを使用します。次の例では、名前空間内のすべてのサービスアカウントにロールを使用するアクセス権限を付与します。これを行うには、StringEquals を StringLike に、$service_account を * に置き換えます。

```
cat >trust-relationship.json <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::$account_id:oidc-provider/$oidc_provider"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "$oidc_provider:aud": "sts.amazonaws.com",
          "$oidc_provider:sub": "system:serviceaccount:$namespace:$service_account"
        }
      }
    }
  ]
}
EOF
```

4. ロールを作成します。

```
aws iam create-role --role-name my-role --assume-role-policy-document file://trust-
relationship.json --description "my-role-description"
```

5. 次の eksctl コマンドを使用して、サーバーまたは Spark のサービスアカウントをマッピングします。必ず独自の値を使用してください。

```
eksctl create iamserviceaccount --name spark-sa \
--namespace spark-namespace --cluster livy-eks-cluster \
--attach-role-arn arn:aws:iam::0123456789012:role/my-role \
--approve --override-existing-serviceaccounts
```

Amazon EMR on EKS リリースでの Apache Livy のインストールプロパティ

Apache Livy のインストールでは、Livy Helm チャートのバージョンを選択できます。Helm チャートには、インストールとセットアップのエクスペリエンスをカスタマイズするためにさまざまなプロパティが用意されています。これらのプロパティは、Amazon EMR on EKS リリース 7.1.0 以降でサポートされています。

トピック

- Amazon EMR 7.1.0 のインストールプロパティ

Amazon EMR 7.1.0 のインストールプロパティ

次の表は、サポートされているすべての Livy プロパティを示しています。Apache Livy をインストールするときは、Livy Helm チャートのバージョンを選択できます。インストール中にプロパティを設定するには、コマンド `--set <property>=<value>` を使用します。

プロパティ	説明	デフォルト値
画像	Livy サーバーの Amazon EMR リリース URI です。これは必須の設定です。	""
sparkNamespace	Livy Spark セッションを実行する名前空間です。例えば、「livy」と指定します。これは必須の設定です。	""

プロパティ	説明	デフォルト値
nameOverride	livy の代わりに名前を指定します。名前はすべての Livy リソースのラベルとして設定されます	「livy」
fullnameOverride	リソースのフルネームの代わりに使用する名前を指定します。	""
ssl.enabled	Livy エンドポイントから Livy サーバーへのエンドツーエンド SSL を有効にします。	誤
ssl.certificateArn	SSL が有効になっている場合、これはサービスによって作成された NLB の ACM 証明書 ARN です。	""
ssl.secretProviderClassName	SSL が有効になっている場合、これは Livy サーバー接続の NLB を SSL で保護するためのシークレットプロバイダークラス名です。	""
ssl.keyStoreObjectName	SSL が有効になっている場合、シークレットプロバイダークラスのキーストア証明書のオブジェクト名です。	""
ssl.keyPasswordsObjectName	SSL が有効になっている場合、キーストアとキーパスワードを持つシークレットのオブジェクト名です。	""
rbac.create	true の場合、RBAC リソースを作成します。	誤

プロパティ	説明	デフォルト値
serviceAccount.create	true の場合、Livy サービスアカウントを作成します。	正
serviceAccount.name	Livy に使用するサービスアカウントの名前です。このプロパティを設定せずにサービスアカウントを作成すると、Amazon EMR on EKS は `fullname` オーバーライドプロパティを使用して名前を自動的に生成します。	「emr-containers-sa-livy」
serviceAccount.executionRoleArn	Livy サービスアカウントの実行ロール ARN です。	""
sparkServiceAccount.create	true の場合、`.Release.Namespace` で Spark サービスアカウントを作成します。	正
sparkServiceAccount.name	Spark に使用するサービスアカウントの名前です。このプロパティを設定せずに Spark サービスアカウントを作成すると、Amazon EMR on EKS は `fullnameOverride` プロパティに `-spark-livy` サフィックスを付けた名前を自動的に生成します。	「emr-containers-sa-spark-livy」
service.name	Livy サービスの名前	`"emr-containers-livy"`
service.annotations	Livy サービスの注釈	`{}`

プロパティ	説明	デフォルト値
loadbalancer.enabled	Amazon EKS クラスターの外部に Livy エンドポイントを公開するために使用される Livy サービスのロードバランサーを作成するかどうか。	FALSE
loadbalancer.internal	Livy エンドポイントを VPC の内部として設定するか、外部として設定するか。 このプロパティを FALSE に設定すると、VPC の外部のソースにエンドポイントが公開されます。TLS/SSL を使用してエンドポイントを保護することをお勧めします。詳細については、「TLS および SSL 暗号化の設定」を参照してください。	FALSE
imagePullSecrets	プライベートリポジトリから Livy イメージをプルするために使用する `imagePullSecret` の名前のリストです。	[]
リソース	Livy コンテナのリソースリクエストと制限です。	{}
nodeSelector	Livy ポッドをスケジュールするノードです。	{}
tolerations	定義する Livy ポッドの許容度を含むリストです。	[]

プロパティ	説明	デフォルト値
affinity	Livy ポッドのアフィニティのルールです。	{}
persistence.enabled	true の場合、セッションディレクトリの永続性を有効にします。	誤
persistence.subPath	セッションディレクトリにマウントする PVC サブパスです。	""
persistence.existingClaim	新しい PVC を作成する代わりに使用する PVC です。	{}
persistence.storageClass	使用するストレージクラスです。このパラメータを定義するには、`storageClassName: <storageClass>` 形式を使用します。このパラメータを "-" に設定すると、動的プロビジョニングが無効になります。このパラメータを null に設定するか、何も指定しない場合、Amazon EMR on EKS は storageClassName を設定せず、デフォルトのプロビジョナーを使用します。	""
persistence.accessMode	PVC アクセスモードです。	`ReadWriteOnce`
persistence.size	PVC サイズです。	20Gi
persistence.annotations	PVC の追加の注釈です。	{}

プロパティ	説明	デフォルト値
env.*	Livy コンテナに設定する追加の env です。詳細については、「Livy のインストール中に独自の Livy と Spark の設定を入力する」を参照してください。	{}
envFrom.*	Kubernetes 設定マップまたはシークレットから Livy に設定する追加の env です。	[]
livyConf.*	マウントされた Kubernetes 設定マップまたはシークレットから設定する追加の livy.conf エントリです。	{}
sparkDefaultsConf.*	マウントされた Kubernetes 設定マップまたはシークレットから設定する追加の `spark-defaults.conf` エントリです。	{}

一般的な環境変数の形式エラーのトラブルシューティング

Livy と Spark の設定を入力すると、サポートされていない環境変数の形式があり、エラーの原因となることがあります。この手順では、正しい形式を使用するための一連の手順を説明します。

Livy のインストール中に独自の Livy と Spark の設定を入力する

`env.*` Helm プロパティを使用して、任意の Apache Livy または Apache Spark の環境変数を設定できます。以下のステップに従って、サンプル設定 `example.config.with-dash.withUppercase` をサポートされている環境変数の形式に変換します。

1. 大文字を 1 と小文字に置き換えます。例えば、`example.config.with-dash.withUppercase` は `example.config.with-dash.with1uppercase` になります。

2. ダッシュ (-) を 0 に置き換えます。例えば、`example.config.with-dash.with1uppercase` は `example.config.with0dash.with1uppercase` になります。
3. ドット (.) をアンダースコア (_) に置き換えます。例えば、`example.config.with0dash.with1uppercase` は `example_config_with0dash_with1uppercase` になります。
4. 小文字をすべて大文字に置き換えます。
5. プレフィックス `LIVY_` を変数名に追加します。
6. helm チャートから Livy をインストールする際に、--set env.*YOUR_VARIABLE_NAME*.value=*yourvalue* 形式で変数を使用します。

例えば、Livy と Spark の設定 `livy.server.recovery.state-store = filesystem` と `spark.kubernetes.executor.podNamePrefix = my-prefix` を設定するには、次の Helm プロパティを使用します。

```
–set env.LIVY_LIVY_SERVER_RECOVERY_STATE0STORE.value=filesystem
–set env.LIVY_SPARK_KUBERNETES_EXECUTOR_POD0NAME0PREFIX.value=myprefix
```

Amazon EMR on EKS ジョブ実行の管理

以下のセクションでは、Amazon EMR on EKS ジョブ実行を管理するのに役立つトピックについて説明します。これには、 を使用する際のジョブ実行パラメータの設定 AWS CLI、ログデータの保存方法の設定、クエリを実行するための Spark SQL スクリプトの実行、ジョブの実行状態の確認、ジョブのモニタリング方法の把握が含まれます。データを処理するためにジョブ実行を設定して完了する場合は、通常、これらのトピックを順番に実行します。

トピック

- を使用したジョブ実行の管理 AWS CLI
- StartJobRun API による Spark SQL スクリプトの実行
- ジョブ実行状態
- Amazon EMR コンソールでジョブを表示する
- ジョブ実行時の一般的なエラー

を使用したジョブ実行の管理 AWS CLI

このトピックでは、 AWS Command Line Interface () を使用してジョブ実行を管理する方法について説明しますAWS CLI。セキュリティパラメータ、ドライバー、さまざまなオーバーライド設定などのプロパティについて詳しく説明します。また、ログ記録を設定するさまざまな方法を扱ったサブトピックも含まれています。

トピック

ジョブ実行を構成するためのオプション

以下のオプションを使用して、ジョブ実行パラメータを設定します。

- `--execution-role-arn`: ジョブの実行に使用する IAM ロールを指定する必要があります。詳細については、「Amazon EMR on EKS でのジョブ実行ロールの使用」を参照してください。
- `--release-label`: Amazon EMR バージョン 5.32.0 および 6.2.0 以降を使用して Amazon EMR on EKS をデプロイできます。Amazon EMR on EKS は、以前の Amazon EMR リリースバージョンではサポートされていません。詳細については、「Amazon EMR on EKS リリース」を参照してください。
- `--job-driver`: Job ドライバーは、メインジョブに入力を提供するために使用されます。これは、実行するジョブタイプの値の 1 つだけを渡すことができるユニオンタイプフィールドです。サポートされるジョブタイプには次のものが含まれます。
  - Spark 送信ジョブ - Spark 送信を通じてコマンドを実行するために使用されます。このジョブタイプを使用して、Scala、PySpark、SparkR、SparkSQL およびその他のサポートされているジョブを Spark 送信を通じて 実行できます。このジョブタイプには以下のパラメータがあります。
    - エントリポイント - これは、実行するメイン jar/py ファイルへの HCFS (Hadoop 互換ファイルシステム) 参照です。

- EntryPointArguments - これはメイン jar/pyファイルに渡す引数の配列です。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。配列の各引数は、カンマで区切る必要があります。EntryPointArguments には、()、{}、[] などの角括弧や丸括弧を含めることはできません。
- 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、ドライバーメモリや —conf や —class などのエグゼキューターの数など、デフォルトの Spark プロパティを上書きします。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 SQL ジョブ - Spark SQL を使用して SQL クエリファイルを実行するために使用されます。このジョブタイプを使用して SparkSQL ジョブを実行できます。このジョブタイプには以下のパラメータがあります。
  - エントリポイント - これは、実行する SQL クエリファイルへの HCFS (Hadoop 互換ファイルシステム) 参照です。

    Spark SQL ジョブに使用できるその他の Spark パラメータのリストについては、「StartJobRun API による Spark SQL スクリプトの実行」を参照してください。

- `--configuration-overrides`: 設定オブジェクトを提供することで、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、設定を指定したり、JSON ファイルの設定オブジェクトを参照したりできます。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。単一の 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。Amazon EMR リリースバージョンによって使用可能な設定分類は異なります。Amazon EMR の各リリースバージョンで使用可能な設定分類の一覧については、Amazon EMR on EKS リリース を参照してください。

  アプリケーションの上書きと Spark 送信パラメータで同じ設定を渡すと、Spark 送信パラメータが優先されます。完全な設定優先順位リストは、優先順位の最も高いものから最も低いものの順に表示されます。

  - `SparkSession` 作成時に提供される構成。
  - `–conf` を使用して `sparkSubmitParameters` の一部として提供される構成。
  - アプリケーションの上書きの一部として提供される設定。
  - リリース用に Amazon EMR によって選択された最適化された設定。
  - アプリケーションのデフォルトのオープンソース構成。

Amazon CloudWatch または Amazon S3 を使用してジョブの実行をモニタリングするには、CloudWatch の設定の詳細を指定する必要があります。詳細については、Amazon S3 ログを使用するようにジョブ実行を設定するおよびAmazon CloudWatch Logs を使用するようにジョブ実行を設定するを参照してください。S3 バケットまたは CloudWatch Logs グループが存在しない場合、Amazon EMR はバケットにログをアップロードする前にそれを作成します。

- Kubernetes 設定オプションのその他のリストについては、Kubernetes の Spark プロパティを参照してください。

  以下の Spark 設定はサポートされていません。

  - `spark.kubernetes.authenticate.driver.serviceAccountName`
  - `spark.kubernetes.authenticate.executor.serviceAccountName`
  - `spark.kubernetes.namespace`
  - `spark.kubernetes.driver.pod.name`
  - `spark.kubernetes.container.image.pullPolicy`
  - `spark.kubernetes.container.image`

  > **Note**
  >
  > カスタマイズされた Docker イメージに `spark.kubernetes.container.image` を使用できます。詳細については、「Amazon EMR on EKS の Docker イメージのカスタマイズ」を参照してください。

Amazon S3 ログを使用するようにジョブ実行を設定する

ジョブの進行状況をモニタリングし、障害のトラブルシューティングをできるようにするには、Amazon S3、Amazon CloudWatch Logs、またはその両方にログ情報を送信するようにジョブを設定する必要があります。このトピックは、Amazon EMR on EKS で起動したジョブで Amazon S3 へのアプリケーションログの発行を開始するのに役立ちます。

S3 ログ IAM ポリシー

ジョブが Amazon S3 にログデータを送信できるようにするには、ジョブ実行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。*amzn-s3-demo-logging-bucket* をログ記録バケットの名前に置き換えます。

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

> **Note**
>
> Amazon EMR on EKS は、Amazon S3 バケットを作成することもできます。Amazon S3 バケットが利用できない場合は、IAM ポリシーに “s3:CreateBucket” アクセス許可を含めてください。

Amazon S3 にログを送信するための適切なアクセス許可を実行ロールに付与した後、「を使用したジョブ実行の管理 AWS CLI」に示すように、start-job-run リクエストの monitoringConfiguration セクションで s3MonitoringConfiguration が渡されると、ログデータは次の Amazon S3 の場所に送信されます。

- 送信者ログ -/*logUri*/*virtual-cluster-id*/jobs/*job-id*/containers/*pod-name*/(stderr.gz/stdout.gz)
- ドライバーログ - /*logUri*/*virtual-cluster-id*/jobs/*job-id*/containers/*spark-application-id*/spark-*job-id*-driver/(stderr.gz/stdout.gz)
- エグゼキューターログ - /*logUri*/*virtual-cluster-id*/jobs/*job-id*/containers/*spark-application-id*/*executor-pod-name*/(stderr.gz/stdout.gz)

Amazon CloudWatch Logs を使用するようにジョブ実行を設定する

ジョブの進行状況をモニタリングし、障害のトラブルシューティングを行うには、Amazon S3、Amazon CloudWatch Logs、またはその両方にログ情報を送信するようにジョブを設定する必要があります。このトピックは、Amazon EMR on EKS で起動されたジョブでの CloudWatch Logs の使用開始に役立ちます。CloudWatch Logs について詳しくは、Amazon CloudWatch ユーザーガイドのログファイルのモニタリングを参照してください。

CloudWatch Logs IAM ポリシー

ジョブが CloudWatch Logs にログデータを送信するには、ジョブ実行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。*my_log_group_name* と *my_log_stream_prefix* を、それぞれ CloudWatch Logs グループとログストリームの名前に置き換えます。実行ロール ARN に適切なアクセス許可がある限り、Amazon EMR on EKS は、ロググループとログストリームが存在しない場合に作成します。

```
{
     "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:DescribeLogGroups",
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:*:*:*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:*:*:log-group:my_log_group_name:log-
stream:my_log_stream_prefix/*"
            ]
        }
    ]
}
```

> **Note**
>
> Amazon EMR on EKS はログストリームを作成することもできます。ログストリームが存在しない場合は、IAM ポリシーに "logs:CreateLogGroup" アクセス許可を含める必要があります。

実行ロールに適切なアクセス許可を付与した後、「を使用したジョブ実行の管理 AWS CLI」に示すように、start-job-run リクエストの monitoringConfiguration セクションで cloudWatchMonitoringConfiguration が渡されると、アプリケーションはその口グデータを CloudWatch Logs に送信します。

StartJobRun API で、*log_group_name* は CloudWatch のロググループ名であり、*log_stream_prefix* は CloudWatch のログストリーム名のプレフィックスです。これらのログは AWS Management Consoleで表示および検索できます。

- 送信者ログ - *logGroup*/*logStreamPrefix*/*virtual-cluster-id*/jobs/*job-id*/containers/*pod-name*/(stderr/stdout)
- ドライバーログ - *logGroup*/*logStreamPrefix*/*virtual-cluster-id*/jobs/*job-id*/containers/*spark-application-id*/spark-*job-id*-driver/(stderrstdout)
- エグゼキューターログ - *logGroup*/*logStreamPrefix*/*virtual-cluster-id*/jobs/*job-id*/containers/*spark-application-id*/*executor-pod-name*/(stderr/stdout)

ジョブの実行のリスト

次の例に示すように、list-job-run を実行してジョブ実行の状態を表示できます。

```
aws emr-containers list-job-runs --virtual-cluster-id <cluster-id>
```

ジョブ実行の説明

次の例に示すように、describe-job-run を実行すると、ジョブの状態、状態の詳細、ジョブ名など、ジョブの詳細を取得できます。

```
aws emr-containers describe-job-run --virtual-cluster-id cluster-id --id job-run-id
```

ジョブ実行をキャンセルする

次の例に示すように、cancel-job-run を実行して実行中のジョブをキャンセルできます。

```
aws emr-containers cancel-job-run --virtual-cluster-id cluster-id --id job-run-id
```

StartJobRun API による Spark SQL スクリプトの実行

Amazon EMR on EKS リリース 6.7.0 以降には Spark SQL ジョブドライバーが含まれているため、StartJobRun API を使用して Spark SQL スクリプトを実行できます。SQL エントリポイントファイルを提供することで、既存の Spark SQL スクリプトを変更することなく、StartJobRun API を使用して Amazon EMR on EKS で Spark SQL クエリを直接実行できます。以下の表は、StartJobRun API を使用して Spark SQL ジョブでサポートされている Spark パラメータの一覧です。

Spark SQL ジョブに送信するパラメータは、以下の Spark パラメータから選択できます。これらのパラメータを使用して、Spark プロパティのデフォルトを上書きします。

オプション	説明
--name NAME	アプリケーション名
--jars JARS	ドライバーと実行クラスパスに含める jar のカンマ区切りリスト。
--packages	ドライバーとエグゼキュータークラスパスに含める jar の Maven 座標のカンマ区切りリスト。
--exclude-packages	–packages で提供される依存関係を解決する際に除外して、依存関係の競合を回避する groupId:artifactId のカンマ区切りリスト。
--repositories	—packages で与えられた Maven 座標を検索するための追加のリモートリポジトリのカンマ区切りリスト。
--files FILES	各エグゼキューターの作業ディレクトリに配置されるファイルのカンマ区切りリスト。

オプション	説明
--conf PROP=VALUE	Spark 設定プロパティ。
--properties-file FILE	追加プロパティを読み込むファイルへのパス。
--driver-memory MEM	ドライバー用メモリ。デフォルト 1024 MB。
--driver-java-options	ドライバーに渡す追加の Java オプション。
--driver-library-path	ドライバーに渡す追加のライブラリパスエントリ。
--driver-class-path	ドライバーに渡す追加のクラスパスエントリ。
--executor-memory MEM	エグゼキューターあたりのメモリ。デフォルト 1GB。
--driver-cores NUM	ドライバーが使用するコアの数。
--total-executor-cores NUM	すべてのエグゼキューターの合計コア。
--executor-cores NUM	各エグゼキューターが使用するコアの数。
--num-executors NUM	起動するエグゼキューターの数。
-hivevar <key=value>	Hive コマンドに適用する変数置換。例: `-hivevar A=B`
-hiveconf <property=value>	指定したプロパティに使用する値。

Spark SQL ジョブの場合、start-job-run-request.json ファイルを作成し、以下の例に示すように、ジョブ実行に必要なパラメータを指定します。

```
{
  "name": "myjob",
  "virtualClusterId": "123456",
  "executionRoleArn": "iam_role_name_for_job_execution",
  "releaseLabel": "emr-6.7.0-latest",
  "jobDriver": {
    "sparkSqlJobDriver": {
```

```
      "entryPoint": "entryPoint_location",
       "sparkSqlParameters": "--conf spark.executor.instances=2 --conf
 spark.executor.memory=2G --conf spark.executor.cores=2 --conf spark.driver.cores=1"
    }
  },
  "configurationOverrides": {
    "applicationConfiguration": [
      {
        "classification": "spark-defaults",
        "properties": {
          "spark.driver.memory":"2G"
         }
      }
    ],
    "monitoringConfiguration": {
      "persistentAppUI": "ENABLED",
      "cloudWatchMonitoringConfiguration": {
        "logGroupName": "my_log_group",
        "logStreamNamePrefix": "log_stream_prefix"
      },
      "s3MonitoringConfiguration": {
        "logUri": "s3://my_s3_log_location"
      }
    }
  }
}
```

ジョブ実行状態

Amazon EMR on EKS ジョブキューにジョブ実行を送信すると、ジョブ実行は PENDING 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。

ジョブ実行の各状態は以下のとおりです。

- PENDING - ジョブ実行が Amazon EMR on EKS に送信されたときの初期のジョブの状態。ジョブは仮想クラスターへの送信を待機しており、Amazon EMR on EKS はこのジョブの送信を行っています。
- SUBMITTED - 仮想クラスターに正常に送信されたジョブ実行。その後、クラススケジューラは、クラスターでこのジョブを実行しようとします。

- RUNNING - 仮想クラスターで実行されているジョブ実行。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。
- FAILED - 仮想クラスターへの送信に失敗したジョブ実行、または正常に完了しなかったジョブ実行。StateDetails と FailureReason を参照して、このジョブの失敗に関する追加情報を調べてください。
- COMPLETED - 正常に完了したジョブ実行。
- CANCEL_PENDING - ジョブ実行のキャンセルがリクエストされました。Amazon EMR on EKS は、仮想クラスター上のジョブをキャンセルしようとしています。
- CANCELLED - 正常にキャンセルされたジョブ実行。

Amazon EMR コンソールでジョブを表示する

ジョブ実行データが表示できるため、各ジョブが状態を通過するのをモニタリングできます。Amazon EMR コンソールでジョブを表示するには、以下のステップを実行します。

1. Amazon EMR コンソールの左側にあるメニューの Amazon EMR on EKS で、[仮想クラスター] を選択します。
2. 仮想クラスターのリストから、ジョブを表示する仮想クラスターを選択します。
3. [Job runs] (ジョブ実行) テーブルで、[View logs] (ログを表示する) を選択して、ジョブ実行の詳細を表示します。

> **Note**
>
> ワンクリックエクスペリエンスのサポートは、デフォルトで有効になっています。ジョブの送信中に monitoringConfiguration の persistentAppUI を DISABLED に設定すると、これを無効にできます。詳細については、「永続アプリケーションユーザーインターフェイスの表示」を参照してください。

ジョブ実行時の一般的なエラー

StartJobRun API を実行すると、次のエラーが発生することがあります。この表には、各エラーが一覧表示され、問題に迅速に対処できるように緩和策の手順が記載されています。

エラーメッセージ	エラー状態	推奨される次のステップ
エラー: 引数 --##は必須です	必須のパラメータが欠落しています。	欠落している引数を API リクエストに追加します。
[StartJobRun オペレーションの呼び出し時にエラー (AccessDeniedException) が発生しました: ユーザー *ARN* は emr-containers:StartJobRun の実行を承認されていません]	実行ロールが欠落しています。	Amazon EMR on EKS でのジョブ実行ロールの使用を参照してください。
[StartJobRun オペレーションの呼び出し時にエラー (AccessDeniedException) が発生しました: ユーザー *ARN* は emr-containers:StartJobRun の実行を承認されていません]	呼び出し元に、条件キーを介した実行ロール [有効/無効な形式] に対するアクセス許可がありません。	「Amazon EMR on EKS でのジョブ実行ロールの使用」を参照してください。
[StartJobRun オペレーションの呼び出し時にエラー (AccessDeniedException) が発生しました: ユーザー *ARN* は emr-containers:StartJobRun の実行を承認されていません]	ジョブの送信者と実行ロールの ARN が、異なるアカウントのものです。	ジョブの送信者と実行ロールの ARN が同じ AWS アカウントのものであることを確認します。
1 検証エラーが検出されました:「executionRoleArn 」の値###が ARN 正規表現パターンを満たすことができませんでした: ^arn:(aws[a-zA-Z0-9-]*):iam::(\d{12})?:(role((\u002F)|(\u002F[\u0021-\u007F]+\u002F))[\w+=,.@-]+)	呼び出し元は、条件キーを介して実行ロールのアクセス許可を持っていますが、ロールは ARN 形式の制約を満たしていません。	ARN 形式に従って実行ロールを指定します。「Amazon EMR on EKS でのジョブ実行ロールの使用」を参照してください。

エラーメッセージ	エラー状態	推奨される次のステップ
StartJobRun オペレーションを呼び出すときにエラー (ResourceNotFoundException) が発生しました: 仮想クラスター*####### ID* は存在しません。	仮想クラスター ID が見つかりません。	Amazon EMR on EKS に登録された仮想クラスター ID を指定します。
StartJobRun オペレーションを呼び出すときにエラー (ValidationException) が発生しました: 仮想クラスター状態*state* はリソース JobRun の作成には有効ではありません。	仮想クラスターはジョブを実行する準備ができていません。	「仮想クラスターの状態」を参照してください。
StartJobRun オペレーションを呼び出すときにエラー (ResourceNotFoundException) が発生しました: リリース *RELEASE* は存在しません。	ジョブの送信で指定されたリリースが正しくありません。	「Amazon EMR on EKS リリース」を参照してください。

エラーメッセージ	エラー状態	推奨される次のステップ
[StartJobRun オペレーションの呼び出し時にエラー (AccessDeniedException) が発生しました: ユーザー *ARN* はリソースに対する emr-containers:StartJobRun の実行を承認されていません: *ARN* (明示的な拒否を使用)] [StartJobRun オペレーションの呼び出し時にエラー (AccessDeniedException) が発生しました: ユーザー *ARN* にはリソースに対して emr-containers:StartJobRun を実行する権限がありません: *ARN*]	ユーザーに StartJobRun を呼び出す権限がありません。	「Amazon EMR on EKS でのジョブ実行ロールの使用」を参照してください。
StartJobRun オペレーションの呼び出し時にエラー (ValidationException) が発生しました: configurationOverrides.monitoringConfiguration.s3MonitoringConfiguration.logUri が制約を満たすことができませんでした: %s	S3 パス URI 構文が無効です。	LogURI は s3://... の形式でなければなりません

ジョブの実行前に `DescribeJobRun` API を実行すると、次のエラーが発生する場合があります。

エラーメッセージ	エラー状態	推奨される次のステップ
stateDetails: JobRun の送信に失敗しました。 分類 *classification* はサポート対象外です。	StartJobRun のパラメータは無効です。	「Amazon EMR on EKS リリース」を参照してください。

エラーメッセージ	エラー状態	推奨される次のステップ
failureReason: VALIDATION_ERROR 状態: FAILED。		
stateDetails: クラスター *EKS ##### ID* は存在しません。 failureReason: CLUSTER_UNAVAILABLE 状態: FAILED	EKS クラスターは使用できません。	EKS クラスターが存在し、適切なアクセス許可を持っているかどうかを確認します。詳細については、「Amazon EMR on EKS のセットアップ」を参照してください。
stateDetails: クラスター *EKS ##### ID* に十分なアクセス許可がありません。 failureReason: CLUSTER_UNAVAILABLE 状態: FAILED	Amazon EMR に、EKS クラスターにアクセスする権限がありません。	登録された名前空間で、アクセス許可が Amazon EMR に設定されていることを確認します。詳細については、「Amazon EMR on EKS のセットアップ」を参照してください。
stateDetails: クラスター *EKS ##### ID* には現在到達できません。 failureReason: CLUSTER_UNAVAILABLE 状態: FAILED	EKS クラスターに到達できません。	EKS クラスターが存在し、適切なアクセス許可を持っているかどうかを確認します。詳細については、「Amazon EMR on EKS のセットアップ」を参照してください。
stateDetails: 内部エラーのため、JobRun の送信に失敗しました。 failureReason: INTERNAL_ERROR 状態: FAILED	EKS クラスターで内部エラーが発生しました。	該当なし

エラーメッセージ	エラー状態	推奨される次のステップ
stateDetails: クラスター *EKS ##### ID* に十分なリソースがありません。 failureReason: USER_ERROR 状態: FAILED	EKS クラスターでジョブを実行するためのリソースが不足しています。	EKS ノードグループに容量を追加するか、EKS Autoscaler を設定します。詳細については、Cluster Autoscaler を参照してください。

ジョブの実行後に `DescribeJobRun` API を実行すると、次のエラーが発生する場合があります。

エラーメッセージ	エラー状態	推奨される次のステップ
stateDetails: JobRun のモニタリング中に問題が発生しました。 クラスター *EKS ##### ID* は存在しません。 failureReason: CLUSTER_UNAVAILABLE 状態: FAILED	EKS クラスターは存在しません。	EKS クラスターが存在し、適切なアクセス許可を持っているかどうかを確認します。詳細については、「Amazon EMR on EKS のセットアップ」を参照してください。
stateDetails: JobRun のモニタリング中に問題が発生しました。 クラスター *EKS ##### ID* に十分なアクセス許可がありません。 failureReason: CLUSTER_UNAVAILABLE 状態: FAILED	Amazon EMR に、EKS クラスターにアクセスする権限がありません。	登録された名前空間で、アクセス許可が Amazon EMR に設定されていることを確認します。詳細については、「Amazon EMR on EKS のセットアップ」を参照してください。

エラーメッセージ	エラー状態	推奨される次のステップ
stateDetails: JobRun のモニタリング中に問題が発生しました。 クラスター *EKS ##### ID* には現在到達できません。 failureReason: CLUSTER_UNAVAILABLE 状態: FAILED	EKS クラスターに到達できません。	EKS クラスターが存在し、適切なアクセス許可を持っているかどうかを確認します。詳細については、「Amazon EMR on EKS のセットアップ」を参照してください。
stateDetails: 内部エラーのため、JobRun のモニタリング中に問題が発生しました failureReason: INTERNAL_ERROR 状態: FAILED	内部エラーが発生し、JobRun のモニタリングを妨げています。	該当なし

ジョブを開始できず、ジョブが SUBMITTED 状態で 15 分間待機すると、以下のエラーが発生することがあります。クラスターリソースの不足が原因である可能性があります。

エラーメッセージ	エラー状態	推奨される次のステップ
クラスタータイムアウト	ジョブが 15 分以上 SUBMITTED 状態になっている。	このパラメータのデフォルト設定である 15 分は、以下に示す設定オーバーライドでオーバーライドできます。

以下の設定を使用して、クラスタータイムアウト設定を 30 分に変更します。新しい `job-start-timeout` 値を秒単位で指定していることに注意してください。

```
{
"configurationOverrides": {
```

```
  "applicationConfiguration": [{
      "classification": "emr-containers-defaults",
      "properties": {
          "job-start-timeout":"1800"
      }
  }]
}
```

ジョブテンプレートの使用

ジョブテンプレートには、ジョブ実行の開始時に StartJobRun API 呼び出し間で共有できる値が格納されます。以下の 2 つのユースケースをサポートしています。

- StartJobRun API リクエスト値が繰り返し発生するのを防ぐため。
- StartJobRun API リクエストを通じて特定の値を指定する必要があるというルールを適用するため。

ジョブテンプレートを使用すると、ジョブ実行用の再利用可能なテンプレートを定義して、以下のような追加のカスタマイズを適用できます。

- エグゼキューターとドライバーのコンピューティング性能の設定
- IAM ロールなどのセキュリティとガバナンスのプロパティの設定
- 複数のアプリケーションやデータパイプラインで使用するための Docker イメージのカスタマイズ

以下のトピックでは、テンプレートを使用してジョブ実行を開始する方法やテンプレートパラメータを変更する方法など、テンプレートの使用に関する詳細な情報を提供します。

トピック

ジョブ実行を開始するためのジョブテンプレートの作成と使用

このセクションでは、ジョブテンプレートを作成し、テンプレートを使用して AWS Command Line Interface () でジョブ実行を開始する方法について説明しますAWS CLI。

ジョブテンプレートを作成するには

1. create-job-template-request.json ファイルを作成し、以下の JSON ファイルの例に示すように、ジョブテンプレートに必要なパラメータを指定します。使用可能なすべてのパラメータの詳細については、「CreateJobTemplate API」を参照してください。

   StartJobRun API に必要なほとんどの値は jobTemplateData にも必要です。パラメータにプレースホルダーを使用し、ジョブテンプレートを使用して StartJobRun を呼び出すときに値を指定する場合は、ジョブテンプレートのパラメータに関する次のセクションを参照してください。

```
{
   "name": "mytemplate",
   "jobTemplateData": {
        "executionRoleArn": "iam_role_arn_for_job_execution",
        "releaseLabel": "emr-6.7.0-latest",
        "jobDriver": {
            "sparkSubmitJobDriver": {
                "entryPoint": "entryPoint_location",
                "entryPointArguments": [ "argument1","argument2",...],
                "sparkSubmitParameters": "--class <main_class> --conf
 spark.executor.instances=2 --conf spark.executor.memory=2G --conf
 spark.executor.cores=2 --conf spark.driver.cores=1"
            }
        },
        "configurationOverrides": {
            "applicationConfiguration": [
                {
                    "classification": "spark-defaults",
                    "properties": {
                         "spark.driver.memory":"2G"
                    }
                }
            ],
            "monitoringConfiguration": {
                "persistentAppUI": "ENABLED",
                "cloudWatchMonitoringConfiguration": {
                    "logGroupName": "my_log_group",
                    "logStreamNamePrefix": "log_stream_prefix"
                },
                "s3MonitoringConfiguration": {
                    "logUri": "s3://my_s3_log_location/"
                }
            }
```

```
        }
    }
}
```

2. ローカルに保存されている create-job-template-request.json ファイルへのパスを指定して、create-job-template コマンドを使用します。

```
aws emr-containers create-job-template \
--cli-input-json file://./create-job-template-request.json
```

ジョブテンプレートを使用してジョブ実行を開始するには

以下の例に示すように、StartJobRun コマンドで、仮想クラスター ID、ジョブテンプレート ID、ジョブ名を指定します。

```
aws emr-containers start-job-run \
--virtual-cluster-id 123456 \
--name myjob \
--job-template-id 1234abcd
```

ジョブテンプレートパラメータの定義

ジョブテンプレートパラメータを使用すると、ジョブテンプレート内の変数を指定できます。これらのパラメータ変数の値は、そのジョブテンプレートを使用してジョブ実行を開始するときに指定する必要があります。ジョブテンプレートパラメータは ${parameterName} フォーマットで指定されます。jobTemplateData フィールド内の任意の値をジョブテンプレートパラメータとして指定できます。ジョブテンプレートパラメータ変数ごとに、データタイプ (STRING または NUMBER) を指定し、オプションでデフォルト値を指定します。以下の例は、エントリポイントロケーション、メインクラス、S3 ログロケーション値のジョブテンプレートパラメータを指定する方法を示しています。

エントリポイントの場所、メインクラス、Amazon S3 ログの場所をジョブテンプレートパラメータとして指定するには

1. create-job-template-request.json ファイルを作成し、以下の JSON ファイルの例に示すように、ジョブテンプレートに必要なパラメータを指定します。パラメータの詳細については、「CreateJobTemplate API」を参照してください。

```
{
   "name": "mytemplate",
```

```
  "jobTemplateData": {
       "executionRoleArn": "iam_role_arn_for_job_execution",
       "releaseLabel": "emr-6.7.0-latest",
       "jobDriver": {
           "sparkSubmitJobDriver": {
               "entryPoint": "${EntryPointLocation}",
               "entryPointArguments": [ "argument1","argument2",...],
               "sparkSubmitParameters": "--class ${MainClass} --conf
spark.executor.instances=2 --conf spark.executor.memory=2G --conf
spark.executor.cores=2 --conf spark.driver.cores=1"
           }
       },
       "configurationOverrides": {
           "applicationConfiguration": [
               {
                   "classification": "spark-defaults",
                   "properties": {
                        "spark.driver.memory":"2G"
                   }
               }
           ],
           "monitoringConfiguration": {
               "persistentAppUI": "ENABLED",
               "cloudWatchMonitoringConfiguration": {
                   "logGroupName": "my_log_group",
                   "logStreamNamePrefix": "log_stream_prefix"
               },
               "s3MonitoringConfiguration": {
                   "logUri": "${LogS3BucketUri}"
               }
           }
       },
       "parameterConfiguration": {
           "EntryPointLocation": {
               "type": "STRING"
           },
           "MainClass": {
               "type": "STRING",
               "defaultValue":"Main"
           },
           "LogS3BucketUri": {
               "type": "STRING",
               "defaultValue":"s3://my_s3_log_location/"
           }
```

```
        }
    }
}
```

2. ローカルまたは Amazon S3 に保存されている create-job-template-request.json ファイルへのパスを指定して、create-job-template コマンドを使用します。

```
aws emr-containers create-job-template \
--cli-input-json file://./create-job-template-request.json
```

ジョブテンプレートパラメータを指定したジョブテンプレートを使用してジョブ実行を開始するには

ジョブテンプレートパラメータを含むジョブテンプレートを使用してジョブ実行を開始するには、以下に示すように StartJobRun API リクエストでジョブテンプレート ID とジョブテンプレートパラメータの値を指定します。

```
aws emr-containers start-job-run \
--virtual-cluster-id 123456 \
--name myjob \
--job-template-id 1234abcd \
--job-template-parameters '{"EntryPointLocation": "entry_point_location","MainClass":
 "ExampleMainClass","LogS3BucketUri": "s3://example_s3_bucket/"}'
```

ジョブテンプレートへのアクセスの制御

StartJobRun ポリシーにより、ユーザーまたはロールは、指定したジョブテンプレートを使用してジョブのみを実行でき、指定されたジョブテンプレートを使用しないと StartJobRun 操作を実行できないように強制できます。そのためには、まず、以下に示すように、指定したジョブテンプレートに対する読み取り権限をユーザーまたはロールに付与してください。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "emr-containers:DescribeJobTemplate",
            "Resource": [
                "job_template_1_arn",
                "job_template_2_arn",
                ...
```

```
            ]
        }
    ]
}
```

指定したジョブテンプレートを使用する場合にのみユーザーまたはロールが StartJobRun 操作を呼び出せるようにするには、特定のユーザーまたはロールに以下の StartJobRun ポリシー権限を割り当てます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "emr-containers:StartJobRun",
            "Resource": [
                "virtual_cluster_arn",
            ],
            "Condition": [
                "StringEquals": {
                    "emr-containers:JobTemplateArn": [
                        "job_template_1_arn",
                        "job_template_2_arn",
                        ...
                    ]
                 }
                ]
            }
        }
    ]
}
```

ジョブテンプレートが実行ロール ARN フィールド内にジョブテンプレートパラメータを指定している場合、ユーザーはこのパラメータに値を指定できるため、任意の実行ロールを使用して StartJobRun を呼び出すことができます。ユーザーが指定できる実行ロールを制限するには、「Amazon EMR on EKS でのジョブ実行ロールの使用」の「実行ロールへのアクセスの制御」を参照してください。

特定のユーザーまたはロールに対して上記の StartJobRun アクションポリシーで条件が指定されていない場合、ユーザーまたはロールは、読み取りアクセス権のある任意のジョブテンプレートを使

用するか、任意の実行ロールを使用して、指定された仮想クラスター上で StartJobRun アクションを呼び出すことができます。

ポッドテンプレートの使用

Amazon EMR バージョン 5.33.0 または 6.3.0 以降、Amazon EMR on EKS では Spark のポッドテンプレート機能がサポートされます。ポッドは、共有ストレージとネットワークリソース、およびコンテナの実行方法についての仕様を持つ 1 つ以上のコンテナのグループです。ポッドテンプレートは、各ポッドの実行方法を決定する仕様です。ポッドテンプレートファイルを使用して、Spark 構成でサポートされていないドライバーまたはエグゼキューターポッドの設定を定義できます。Spark のポッドテンプレート機能の詳細については、ポッドテンプレートを参照してください。

> **Note**
>
> ポッドテンプレート機能は、ドライバーポッドとエグゼキューターポッドでのみ機能します。ポッドテンプレートを使用してジョブ送信者ポッドを設定することはできません。

一般的なシナリオ

Amazon EMR on EKS でポッドテンプレートを使用して、共有 EKS クラスター上で Spark ジョブを実行する方法を定義し、コストを削減し、リソース使用率とパフォーマンスを向上させることができます。

- コストを削減するために、Spark エグゼキュータータスクを Amazon EC2 スポットインスタンスで実行するようにスケジューリングするのと同時に、Spark ドライバータスクを Amazon EC2 オンデマンドインスタンスで実行するようにスケジュールできます。
- リソースの使用率を高めるために、複数のチームが同じ EKS クラスターでワークロードを実行するのをサポートできます。各チームは、ワークロードを実行するために指定された Amazon EC2 ノードグループを取得します。ポッドテンプレートを使用して、対応する許容値をワークロードに適用できます。
- モニタリングを改善するために、別のログ記録コンテナを実行して、既存のモニタリングアプリケーションにログを転送できます。

例えば、次のポッドテンプレートファイルは、一般的な使用シナリオを示しています。

```
apiVersion: v1
```

```
kind: Pod
spec:
  volumes:
    - name: source-data-volume
      emptyDir: {}
    - name: metrics-files-volume
      emptyDir: {}
  nodeSelector:
    eks.amazonaws.com/nodegroup: emr-containers-nodegroup
  containers:
  - name: spark-kubernetes-driver # This will be interpreted as driver Spark main
 container
    env:
      - name: RANDOM
        value: "random"
    volumeMounts:
      - name: shared-volume
        mountPath: /var/data
      - name: metrics-files-volume
        mountPath: /var/metrics/data
  - name: custom-side-car-container # Sidecar container
    image: <side_car_container_image>
    env:
      - name: RANDOM_SIDECAR
        value: random
    volumeMounts:
      - name: metrics-files-volume
        mountPath: /var/metrics/data
    command:
      - /bin/sh
      - '-c'
      -  <command-to-upload-metrics-files>
  initContainers:
  - name: spark-init-container-driver # Init container
    image: <spark-pre-step-image>
    volumeMounts:
      - name: source-data-volume # Use EMR predefined volumes
        mountPath: /var/data
    command:
      - /bin/sh
      - '-c'
      -  <command-to-download-dependency-jars>
```

ポッドテンプレートは以下のタスクを実行します。

- 新しい init コンテナを追加します。これは Spark メインコンテナが起動する前に実行されます。init コンテナは、`source-data-volume` という EmptyDir ボリュームを Spark メインコンテナと共有します。init コンテナで、依存関係のダウンロードや入力データの生成などの初期化ステップを実行できます。次に、Spark メインコンテナがデータを消費します。
- Spark メインコンテナとともに実行される別のサイドカーコンテナを追加します。2 つのコンテナは `metrics-files-volume` という別の `EmptyDir` ボリュームを共有しています。Spark ジョブは Prometheus メトリクスなどのメトリクスを生成できます。その後、Spark ジョブはメトリクスをファイルに入れ、サイドカーコンテナで自分の BI システムにファイルをアップロードして将来の分析を行うことができます。
- Spark メインコンテナに新しい環境変数を追加します。ジョブに環境変数を使用させることができます。
- ポッドが `emr-containers-nodegroup` ノードグループでのみスケジュールされるように、ノードセレクタを定義します。これは、ジョブとチーム間でコンピューティングリソースを分離するのに役立ちます。

Amazon EMR on EKS でポッドテンプレートを有効にする

Amazon EMR on EKS でポッドテンプレート機能を有効にするには、Amazon S3 のポッドテンプレートファイルを指すように Spark プロパティ `spark.kubernetes.driver.podTemplateFile` および `spark.kubernetes.executor.podTemplateFile` を設定します。その後、Spark は、ポッドテンプレートファイルをダウンロードし、それを使用してドライバーポッドとエグゼキューターポッドを作成します。

> **Note**
>
> Spark はジョブ実行ロールを使用してポッドテンプレートをロードするため、ジョブ実行ロールには Amazon S3 にアクセスしてポッドテンプレートをロードするためのアクセス許可が必要です。詳細については、「ジョブ実行ロールを作成する」を参照してください。

次のジョブ実行 JSON ファイルが示すように、`SparkSubmitParameters` を使用して、ポッドテンプレートへの Amazon S3 パスを指定できます。

```
{
  "name": "myjob",
```

```
  "virtualClusterId": "123456",
  "executionRoleArn": "iam_role_name_for_job_execution",
  "releaseLabel": "release_label",
  "jobDriver": {
    "sparkSubmitJobDriver": {
      "entryPoint": "entryPoint_location",
      "entryPointArguments": ["argument1", "argument2", ...],
       "sparkSubmitParameters": "--class <main_class> \
        --conf
 spark.kubernetes.driver.podTemplateFile=s3://path_to_driver_pod_template \
        --conf
 spark.kubernetes.executor.podTemplateFile=s3://path_to_executor_pod_template \
        --conf spark.executor.instances=2 \
        --conf spark.executor.memory=2G \
        --conf spark.executor.cores=2 \
        --conf spark.driver.cores=1"
    }
  }
}
```

別の方法として、次のジョブ実行 JSON ファイルが示すように、configurationOverrides を使用して、ポッドテンプレートへの Amazon S3 パスを指定できます。

```
{
  "name": "myjob",
  "virtualClusterId": "123456",
  "executionRoleArn": "iam_role_name_for_job_execution",
  "releaseLabel": "release_label",
  "jobDriver": {
    "sparkSubmitJobDriver": {
      "entryPoint": "entryPoint_location",
      "entryPointArguments": ["argument1", "argument2", ...],
       "sparkSubmitParameters": "--class <main_class> \
        --conf spark.executor.instances=2 \
        --conf spark.executor.memory=2G \
        --conf spark.executor.cores=2 \
        --conf spark.driver.cores=1"
    }
  },
  "configurationOverrides": {
    "applicationConfiguration": [
      {
        "classification": "spark-defaults",
```

```
        "properties": {
          "spark.driver.memory":"2G",
          "spark.kubernetes.driver.podTemplateFile":"s3://path_to_driver_pod_template",

 "spark.kubernetes.executor.podTemplateFile":"s3://path_to_executor_pod_template"
         }
       }
     ]
   }
}
```

Note

1. 信頼できないアプリケーションコードの分離など、Amazon EMR on EKS でポッドテンプレート機能を使用する場合は、セキュリティガイドラインに従う必要があります。詳細については、「Amazon EMR on EKS でのセキュリティのベストプラクティス」を参照してください。
2. Spark メインコンテナ名は `spark-kubernetes-driver` および `spark-kubernetes-executors` としてハードコードされているため、`spark.kubernetes.driver.podTemplateContainerName` および `spark.kubernetes.executor.podTemplateContainerName` を使用して変更することはできません。Spark メインコンテナをカスタマイズする場合は、これらのハードコードされた名前を使用してポッドテンプレートでコンテナを指定する必要があります。

ポッドテンプレートフィールド

Amazon EMR on EKS でポッドテンプレートを設定する場合は、次のフィールド制限を考慮してください。

- Amazon EMR on EKS では、ポッドテンプレート内の以下のフィールドのみで適切なジョブスケジューリングの有効化が許可されます。

  許可されるポッドレベルのフィールドは次のとおりです。

  - `apiVersion`
  - `kind`
  - `metadata`
  - `spec.activeDeadlineSeconds`

- spec.affinity
- spec.containers
- spec.enableServiceLinks
- spec.ephemeralContainers
- spec.hostAliases
- spec.hostname
- spec.imagePullSecrets
- spec.initContainers
- spec.nodeName
- spec.nodeSelector
- spec.overhead
- spec.preemptionPolicy
- spec.priority
- spec.priorityClassName
- spec.readinessGates
- spec.runtimeClassName
- spec.schedulerName
- spec.subdomain
- spec.terminationGracePeriodSeconds
- spec.tolerations
- spec.topologySpreadConstraints
- spec.volumes

許可されている Spark メインコンテナレベルのフィールドは次のとおりです。

- env
- envFrom
- name
- lifecycle
- livenessProbe
- readinessProbe
- resources

- startupProbe
- stdin
- stdinOnce
- terminationMessagePath
- terminationMessagePolicy
- tty
- volumeDevices
- volumeMounts
- workingDir

ポッドテンプレートで許可されていないフィールドを使用すると、Spark は例外をスローし、ジョブは失敗します。次の例では、許可されていないフィールドが原因で Spark コントローラーログにエラーメッセージが表示されています。

```
Executor pod template validation failed.
Field container.command in Spark main container not allowed but specified.
```

- Amazon EMR on EKS は、ポッドテンプレートで次のパラメータを事前定義しています。ポッドテンプレートで指定するフィールドは、これらのフィールドと重複してはいけません。

  事前定義済みのボリューム名は次のとおりです。

  - emr-container-communicate
  - config-volume
  - emr-container-application-log-dir
  - emr-container-event-log-dir
  - temp-data-dir
  - mnt-dir
  - home-dir
  - emr-container-s3

  Spark メインコンテナにのみ適用される事前定義済みのボリュームマウントを次に示します。

  - 名前: emr-container-communicate; MountPath: /var/log/fluentd
  - 名前: emr-container-application-log-dir; MountPath: /var/log/spark/user
  - 名前: emr-container-event-log-dir; MountPath: /var/log/spark/apps

- 名前: mnt-dir; MountPath: /mnt
- 名前: temp-data-dir; MountPath: /tmp
- 名前: home-dir; MountPath: /home/hadoop

Spark メインコンテナにのみ適用される事前定義済みの環境変数を次に示します。

- SPARK_CONTAINER_ID
- K8S_SPARK_LOG_URL_STDERR
- K8S_SPARK_LOG_URL_STDOUT
- SIDECAR_SIGNAL_FILE

> **Note**
>
> これらの事前定義済みボリュームを使用し、そのボリュームを追加のサイドカーコンテナにマウントすることもできます。例えば、emr-container-application-log-dir を使用してポッドテンプレートで定義されている独自のサイドカーコンテナにマウントできます。

指定したフィールドがポッドテンプレートの事前定義済みフィールドのいずれかと競合する場合、Spark は例外をスローし、ジョブは失敗します。次の例では、事前定義済みのフィールドとの競合が原因で Spark アプリケーションログにエラーメッセージが表示されています。

```
Defined volume mount path on main container must not overlap with reserved mount
 paths: [<reserved-paths>]
```

サイドカーコンテナの考慮事項

Amazon EMR は、Amazon EMR on EKS によってプロビジョニングされるポッドのライフサイクルを制御します。サイドカーコンテナは Spark メインコンテナと同じライフサイクルに従う必要があります。追加のサイドカーコンテナをポッドに挿入する場合は、Spark メインコンテナが終了したときにサイドカーコンテナが自動的に停止できるように、Amazon EMR が定義するポッドライフサイクル管理と統合することをお勧めします。

コストを削減するため、サイドカーコンテナを備えたドライバーポッドがジョブの完了後に実行し続けないようにするプロセスを実装することをお勧めします。Spark ドライバーは、エグゼキューターの完了時にエグゼキューターポッドを削除します。ただし、ドライバープログラムが完了すると、追

加のサイドカーコンテナは引き続き実行されます。ポッドは、Amazon EMR on EKS がドライバーポッドをクリーンアップするまで課金されます。通常は、ドライバー Spark メインコンテナが完了してから 1 分未満です。コストを削減するために、次のセクションで説明するように、追加のサイドカーコンテナをライフサイクル管理メカニズムと統合できます。このメカニズムは Amazon EMR on EKS がドライバーポッドとエグゼキューターポッドの両方に対して定義するものです。

ドライバーポッドとエクゼキューターポッドの Spark メインコンテナは、2 秒ごとに `heartbeat` をファイル `/var/log/fluentd/main-container-terminated` に送信します。Amazon EMR の事前定義済み `emr-container-communicate` ボリュームマウントをサイドカーコンテナに追加すると、サイドカーコンテナのサブプロセスを定義して、このファイルの最終更新時刻を定期的に追跡できます。その後、サブプロセスにより Spark メインコンテナがより長い期間 `heartbeat` を停止していることが検出されると、サブプロセス自体が停止します。

次の例は、ハートビートファイルを追跡し、それ自体を停止するサブプロセスを示しています。*`your_volume_mount`* を、事前定義されたボリュームをマウントするパスに置き換えます。スクリプトは、サイドカーコンテナで使用されるイメージ内にバンドルされています。ポッドテンプレートファイルでは、次のコマンドを使用してサイドカーコンテナを指定できます。`sub_process_script.sh` および `main_command`。

```
MOUNT_PATH="your_volume_mount"
FILE_TO_WATCH="$MOUNT_PATH/main-container-terminated"
INITIAL_HEARTBEAT_TIMEOUT_THRESHOLD=60
HEARTBEAT_TIMEOUT_THRESHOLD=15
SLEEP_DURATION=10

function terminate_main_process() {
  # Stop main process
}

# Waiting for the first heartbeat sent by Spark main container
echo "Waiting for file $FILE_TO_WATCH to appear..."
start_wait=$(date +%s)
while ! [[ -f "$FILE_TO_WATCH" ]]; do
    elapsed_wait=$(expr $(date +%s) - $start_wait)
    if [ "$elapsed_wait" -gt "$INITIAL_HEARTBEAT_TIMEOUT_THRESHOLD" ]; then
        echo "File $FILE_TO_WATCH not found after $INITIAL_HEARTBEAT_TIMEOUT_THRESHOLD
 seconds; aborting"
        terminate_main_process
        exit 1
    fi
    sleep $SLEEP_DURATION;
```

```
done;
echo "Found file $FILE_TO_WATCH; watching for heartbeats..."

while [[ -f "$FILE_TO_WATCH" ]]; do
    LAST_HEARTBEAT=$(stat -c %Y $FILE_TO_WATCH)
    ELAPSED_TIME_SINCE_AFTER_HEARTBEAT=$(expr $(date +%s) - $LAST_HEARTBEAT)
    if [ "$ELAPSED_TIME_SINCE_AFTER_HEARTBEAT" -gt "$HEARTBEAT_TIMEOUT_THRESHOLD" ];
 then
        echo "Last heartbeat to file $FILE_TO_WATCH was more than
 $HEARTBEAT_TIMEOUT_THRESHOLD seconds ago at $LAST_HEARTBEAT; terminating"
        terminate_main_process
        exit 0
    fi
    sleep $SLEEP_DURATION;
done;
echo "Outside of loop, main-container-terminated file no longer exists"

# The file will be deleted once the fluentd container is terminated

echo "The file $FILE_TO_WATCH doesn't exist any more;"
terminate_main_process
exit 0
```

ジョブ再試行ポリシーの使用

Amazon EMR on EKS バージョン 6.9.0 以降では、ジョブ実行の再試行ポリシーを設定できます。再試行ポリシーにより、ジョブドライバーポッドが失敗したり削除されたりすると、自動的に再起動されます。これにより、長時間実行されている Spark ストリーミングジョブの障害に対する耐性が高まります。

ジョブの再試行ポリシーの設定

再試行ポリシーを設定するには、StartJobRun API を使用して `RetryPolicyConfiguration` フィールドを指定します。例 `retryPolicyConfiguration` を以下に示します。

```
aws emr-containers start-job-run \
--virtual-cluster-id cluster_id \
--name sample-job-name \
--execution-role-arn execution-role-arn \
--release-label emr-6.9.0-latest \
--job-driver '{
```

```
  "sparkSubmitJobDriver": {
    "entryPoint": "local:///usr/lib/spark/examples/src/main/python/pi.py",
    "entryPointArguments": [ "2" ],
    "sparkSubmitParameters": "--conf spark.executor.instances=2 --conf
 spark.executor.memory=2G --conf spark.executor.cores=2 --conf spark.driver.cores=1"
  }
}' \
--retry-policy-configuration '{
    "maxAttempts": 5
  }' \
--configuration-overrides '{
  "monitoringConfiguration": {
    "cloudWatchMonitoringConfiguration": {
      "logGroupName": "my_log_group_name",
      "logStreamNamePrefix": "my_log_stream_prefix"
    },
    "s3MonitoringConfiguration": {
       "logUri": "s3://amzn-s3-demo-logging-bucket"
    }
  }
}'
```

> **Note**
>
> `retryPolicyConfiguration` は、 AWS CLI 1.27.68 以降のバージョンからのみ使用できます。を最新バージョン AWS CLI に更新するには、「 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

`maxAttempts` フィールドには、ジョブドライバーポッドが失敗したり削除されたりした場合に再起動する最大回数を設定します。ジョブドライバーが 2 回再試行する間の実行間隔は、Kubernetes ドキュメントで説明されているように、指数関数的な再試行間隔 (10 秒、20 秒、40 秒...) で、上限は 6 分です。

> **Note**
>
> ジョブドライバーを追加実行するたびに、別のジョブ実行として請求され、Amazon EMR on EKS 料金が適用されます。

再試行ポリシー設定値

- ジョブのデフォルト再試行ポリシー: StartJobRun には、デフォルトで最大試行回数 1 回に設定された再試行ポリシーが含まれます。再試行ポリシーは、必要に応じて設定できます。

> **Note**
>
> retryPolicyConfiguration の maxAttempts が 1 に設定されている場合、失敗時にドライバーポッドを起動するための再試行は行われません。

- ジョブの再試行ポリシーを無効にする: 再試行ポリシーを無効にするには、retryPolicyConfiguration の最大試行値を 1 に設定します。

```
"retryPolicyConfiguration": {
    "maxAttempts": 1
}
```

- 有効範囲内のジョブに maxAttempts を設定する: maxAttempts 値が有効範囲外の場合、StartJobRun 呼び出しは失敗します。有効な maxAttempts 範囲は 1 から 2,147,483,647 (32 ビット整数) で、これは Kubernetes の backOffLimit 構成設定でサポートされている範囲です。詳細については、Kubernetes ドキュメントの「Pod 失敗のバックオフポリシー」を参照してください。maxAttempts 値が無効な場合、以下のエラーメッセージが返されます。

```
{
 "message": "Retry policy configuration's parameter value of maxAttempts is invalid"
}
```

ジョブの再試行ポリシーステータスの取得

ListJobRuns および DescribeJobRun API を使用して、ジョブの再試行のステータスを表示できます。再試行ポリシー設定が有効になっているジョブをリクエストすると、ListJobRun および DescribeJobRun レスポンスには、RetryPolicyExecution フィールドの再試行ポリシーのステータスが含まれます。さらに、DescribeJobRun レスポンスにはジョブの StartJobRun リクエストに入力された RetryPolicyConfiguration が含まれます。

レスポンス例

ListJobRuns response

```
{
  "jobRuns": [
    ...
    ...
    "retryPolicyExecution" : {
      "currentAttemptCount": 2
    }
    ...
    ...
  ]
}
```

DescribeJobRun response

```
{
  ...
  ...
  "retryPolicyConfiguration": {
    "maxAttempts": 5
   },
   "retryPolicyExecution" : {
    "currentAttemptCount": 2
  },
  ...
  ...
}
```

これらのフィールドは、以下の「再試行ポリシー設定値」で説明するように、ジョブで再試行ポリシーが無効になっている場合は表示されません。

再試行ポリシーを使用してジョブをモニタリングする

再試行ポリシーを有効にすると、作成されたジョブドライバーごとに CloudWatch イベントが生成されます。これらのイベントをサブスクライブするには、以下のコマンドを使用して CloudWatch イベントルールを設定します。

```
aws events put-rule \
--name cwe-test \
--event-pattern '{"detail-type": ["EMR Job Run New Driver Attempt"]}'
```

このイベントはジョブドライバーの newDriverPodName、newDriverCreatedAt タイムスタンプ、previousDriverFailureMessage、および currentAttemptCount に関する情報を返します。再試行ポリシーが無効になっている場合、これらのイベントは作成されません。

CloudWatch イベントを使用してジョブをモニタリングする方法の詳細については、「Amazon CloudWatch Events でジョブをモニタリングする」を参照してください。

ドライバーとエグゼキュター用のログを検索する

ドライバーポッド名は形式 spark-<job id>-driver-<random-suffix> に従います。ドライバーが生成するエグゼキューターポッド名にも同じ random-suffix が追加されます。この random-suffix を使用すると、ドライバーとそれに関連するエグゼキューターのログを検索できます。random-suffix は、そのジョブの再試行ポリシーが有効になっている場合にのみ存在します。それ以外の場合は、random-suffix は存在しません。

ログ記録用のモニタリング設定を使用してジョブを設定する方法の詳細については、「Spark アプリケーションの実行」を参照してください。

Spark イベントログのローテーションを使用する

Amazon EMR 6.3.0 以降では、Amazon EMR on EKS の Spark イベントログローテーション機能をオンにすることができます。この機能は、単一のイベントログファイルを生成する代わりに、設定された時間間隔に基づいてファイルをローテーションし、最も古いイベントログファイルを削除します。

Spark イベントログをローテーションすると、長時間実行されるジョブやストリーミングジョブ用に生成された大きな Spark イベントログファイルに関する潜在的な問題を回避できます。例えば、persistentAppUI パラメータでイベントログを有効にして、長時間実行される Spark ジョブを開始します。Spark ドライバーは、イベントログファイルを生成します。ジョブが数時間または数日間実行され、Kubernetes ノードのディスク容量が制限されている場合、イベントログファイルは使用可能なすべてのディスク領域を消費する可能性があります。Spark イベントログのローテーション機能をオンにすると、ログファイルを複数のファイルに分割し、最も古いファイルを削除することで問題が解決されます。

> **Note**
>
> この機能は、Amazon EMR on EKS でのみ使用できます。Amazon EC2 上で実行する Amazon EMR は Spark イベントログのローテーションをサポートしていません。

Spark イベントログのローテーション機能をオンにするには、次の Spark パラメータを設定します。

- `spark.eventLog.rotation.enabled` - ログローテーションをオンにします。Spark 設定ファイルではデフォルトで無効となっています。この機能をオンにするには、true に設定します。
- `spark.eventLog.rotation.interval` - ログローテーションの時間間隔を指定します。最小値は 60 秒です。デフォルト値は 300 秒です。
- `spark.eventLog.rotation.minFileSize` - ログファイルをローテーションするための最小ファイルサイズを指定します。最小値およびデフォルト値は 1 MBです。
- `spark.eventLog.rotation.maxFilesToRetain` - クリーンアップ中に保持するローテーションログファイルの数を指定します。有効な範囲は 1～10 です。デフォルト値は 2 です。

これらのパラメータは、次の例に示すように、StartJobRun APIの `sparkSubmitParameters` セクションで指定できます。

```
"sparkSubmitParameters": "--class org.apache.spark.examples.SparkPi --conf
 spark.eventLog.rotation.enabled=true --conf spark.eventLog.rotation.interval=300 --
conf spark.eventLog.rotation.minFileSize=1m --conf
 spark.eventLog.rotation.maxFilesToRetain=2"
```

Spark コンテナログのローテーションを使用する

Amazon EMR 6.11.0 以降では、Amazon EMR on EKS の Spark コンテナログローテーション機能をオンにすることができます。この機能は、単一の `stdout` または `stderr` ログファイルを生成する代わりに、設定されたローテーションサイズに基づいてファイルをローテーションし、コンテナから最も古いログファイルを削除します。

Spark コンテナログをローテーションすると、長時間実行されるジョブやストリーミングジョブ用に生成された大きな Spark ログファイルに関する潜在的な問題を回避できます。例えば、長時間実行される Spark ジョブを開始し、Spark ドライバーがコンテナログファイルを生成するとします。ジョブが数時間または数日間実行され、Kubernetes ノードのディスク容量が制限されている場合、コンテナログファイルは使用可能なすべてのディスク容量を消費する可能性があります。Spark コンテナログローテーションを有効にすると、ログファイルを複数のファイルに分割し、最も古いファイルを削除します。

Spark コンテナログのローテーション機能をオンにするには、以下の Spark パラメータを設定します。

containerLogRotationConfiguration

このパラメータを monitoringConfiguration に組み込むと、ログローテーションが有効になります。デフォルトでは無効となっています。s3MonitoringConfiguration に加えて containerLogRotationConfiguration を使用する必要があります。

rotationSize

rotationSize パラメータはログローテーションのファイルサイズを指定します。指定できる値の範囲は 2KB ~ 2GB です。rotationSize パラメータの数値単位部分は整数として渡されます。小数値はサポートされていないため、値 1500MB などに 1.5 GB のローテーションサイズを指定できます。

maxFilesToKeep

maxFilesToKeep パラメータは、ローテーションが行われた後にコンテナに保持するファイルの最大数を指定します。最小値は 1 で、最大値は 50 です。

これらのパラメータは、次の例に示すように、StartJobRun APIの monitoringConfiguration セクションで指定できます。この例では、rotationSize = "10 MB" と maxFilesToKeep = 3 を使用すると、Amazon EMR on EKS はログを 10 MB でローテーションし、新しいログファイルを生成し、ログファイルの数が 3 に達すると最も古いログファイルを削除します。

```
{
  "name": "my-long-running-job",
  "virtualClusterId": "123456",
  "executionRoleArn": "iam_role_name_for_job_execution",
  "releaseLabel": "emr-6.11.0-latest",
  "jobDriver": {
    "sparkSubmitJobDriver": {
      "entryPoint": "entryPoint_location",
      "entryPointArguments": ["argument1", "argument2", ...],
       "sparkSubmitParameters": "--class main_class --conf spark.executor.instances=2
 --conf spark.executor.memory=2G --conf spark.executor.cores=2 --conf
 spark.driver.cores=1"
    }
 },
 "configurationOverrides": {
    "applicationConfiguration": [
      {
        "classification": "spark-defaults",
        "properties": {
```

```
          "spark.driver.memory":"2G"
        }
      }
    ],
    "monitoringConfiguration": {
      "persistentAppUI": "ENABLED",
      "cloudWatchMonitoringConfiguration": {
        "logGroupName": "my_log_group",
        "logStreamNamePrefix": "log_stream_prefix"
      },
      "s3MonitoringConfiguration": {
        "logUri": "s3://my_s3_log_location"
      },
      "containerLogRotationConfiguration": {
        "rotationSize":"10MB",
        "maxFilesToKeep":"3"
      }
    }
  }
}
```

Spark コンテナログローテーションを使用してジョブ実行を開始するには、これらのパラメータで設定した JSON ファイルへのパスを StartJobRun コマンドに含めます。

```
aws emr-containers start-job-run \
--cli-input-json file://path-to-json-request-file
```

Amazon EMR Spark ジョブで垂直的自動スケーリングを使用する

Amazon EMR on EKS の垂直的自動スケーリングは、Amazon EMR Spark アプリケーションに提供するワークロードのニーズに合わせてメモリと CPU リソースを自動的に調整します。これにより、リソース管理が簡単になります。

Amazon EMR Spark アプリケーションのリアルタイムおよび過去のリソース使用率を追跡するために、垂直的自動スケーリングは Kubernetes Vertical Pod Autoscaler (VPA) を活用します。垂直的自動スケーリング機能では、VPA が収集したデータを使用して、Spark アプリケーションに割り当てられるメモリと CPU リソースを自動的に調整します。このシンプルなプロセスにより、信頼性が向上し、コストが最適化されます。

トピック

Amazon EMR on EKS の垂直的自動スケーリングのセットアップ

このトピックは、Amazon EKS クラスターが垂直的自動スケーリングを使用して Amazon EMR Spark ジョブを送信できるように準備するのに役立ちます。セットアッププロセスでは、以下のセクションのタスクを確認または完了する必要があります。

トピック

前提条件

クラスターに垂直的自動スケーリング Kubernetes 演算子をインストールする前に、以下のタスクを完了します。前提条件のいずれかを既に完了している場合は、その前提条件をスキップして、次の前提条件に進むことができます。

- の最新バージョンをインストールまたは更新する AWS CLI – を既にインストールしている場合は AWS CLI、最新バージョンであることを確認します。
- kubectl をインストールする – kubectl は Kubernetes API サーバーと通信するために使用するコマンドラインツールです。Amazon EKS クラスターに垂直的自動スケーリング関連のアーティファクトをインストールしてモニタリングするには kubectl が必要です。
- Operator SDK をインストールする — Amazon EMR on EKS は、クラスターにインストールする垂直的自動スケーリング演算子の存続期間中、Operator SDK をパッケージマネージャーとして使用します。
- Docker をインストールする — Amazon EKS クラスターにインストールする垂直的自動スケーリング関連の Docker イメージを認証して取得するには、Docker CLI にアクセスする必要があります。

- Kubernetes メトリクスサーバーをインストールする – まずメトリクスサーバーをインストールして、垂直ポッドオートスケーラーが Kubernetes API サーバーからメトリクスを取得できるようにする必要があります。
- Amazon EKS – eksctl の使用開始 (バージョン 1.24 以降) — 垂直的自動スケーリングは Amazon EKS バージョン 1.24 以降でサポートされています。クラスターを作成したら、Amazon EMR で使用できるように登録します。
- Amazon EMR ベースイメージ URI (リリース 6.10.0 以上) を選択する — 垂直的自動スケーリングは Amazon EMR リリース 6.10.0 以降でサポートされています。

Amazon EKS クラスターに Operator Lifecycle Manager (OLM) をインストールする

Operator SDK CLI を使用して、以下の例に示すように、垂直的自動スケーリングを設定する Amazon EMR on EKS クラスターに Operator Lifecycle Manager (OLM) をインストールします。セットアップが完了すると、OLM を使用して Amazon EMR 垂直的自動スケーリング演算子のライフサイクルをインストールして管理できます。

```
operator-sdk olm install
```

インストールを検証するには、以下の `olm status` コマンドを実行します。

```
operator-sdk olm status
```

コマンドが以下の出力例のような成功した結果を返すことを確認します。

```
INFO[0007] Successfully got OLM status for version X.XX
```

インストールが成功しなかった場合は、「Amazon EMR on EKS 垂直自動スケーリングのトラブルシューティング」を参照してください。

Amazon EMR on EKS の垂直的自動スケーリング演算子をインストールする

以下の手順を使用して Amazon EKS クラスターに垂直的自動スケーリング演算子をインストールします。

1. インストールを完了するために使用する以下の環境変数を設定します。
   - **`$REGION`** はクラスターの AWS リージョン を指します。例えば、`us-west-2` と指定します。

- **$ACCOUNT_ID** はリージョンの Amazon ECR アカウント ID を指します。詳細については、「リージョン別の Amazon ECR レジストリアカウント」を参照してください。
- **$RELEASE** はクラスターに使用したい Amazon EMR リリースを指します。垂直的自動スケーリングでは、Amazon EMR リリース 6.10.0 以降を使用する必要があります。

2. 次に、演算子の Amazon ECR レジストリへの認証トークンを取得します。

```
aws ecr get-login-password \
 --region region-id | docker login \
 --username AWS \
 --password-stdin $ACCOUNT_ID.dkr.ecr.region-id.amazonaws.com
```

3. Amazon EMR on EKS 垂直的自動スケーリング演算子を以下のコマンドでインストールします。

```
ECR_URL=$ACCOUNT_ID.dkr.ecr.$REGION.amazonaws.com && \
REPO_DEST=dynamic-sizing-k8s-operator-olm-bundle && \
BUNDLE_IMG=emr-$RELEASE-dynamic-sizing-k8s-operator && \
operator-sdk run bundle \
$ECR_URL/$REPO_DEST/$BUNDLE_IMG\:latest
```

これにより、Amazon EKS クラスターのデフォルト名前空間に垂直的自動スケーリング演算子のリリースが作成されます。このコマンドを使用して別の名前空間にインストールします。

```
operator-sdk run bundle \
$ACCOUNT_ID.dkr.ecr.$REGION.amazonaws.com/dynamic-sizing-k8s-operator-olm-bundle/
emr-$RELEASE-dynamic-sizing-k8s-operator:latest \
-n operator-namespace
```

> **Note**
>
> 指定した名前空間が存在しない場合、OLM は演算子をインストールしません。詳細については、「Kubernetes 名前空間が見つからない」を参照してください。

4. kubectl Kubernetes コマンドラインツールを使用して、演算子が正常にインストールされたことを確認します。

```
kubectl get csv -n operator-namespace
```

kubectl コマンドは、新しくデプロイされた垂直的自動スケーラー演算子を [フェーズ] ステータスが [成功] で返します。インストールやセットアップに問題がある場合は、「Amazon EMR on EKS 垂直自動スケーリングのトラブルシューティング」を参照してください。

Amazon EMR on EKS の垂直的自動スケーリングを開始する

Amazon EMR Spark アプリケーションのワークロードに合わせてメモリと CPU リソースを自動的に調整する場合は、Amazon EMR on EKS の垂直的自動スケーリングを使用します。詳細については、「Amazon EMR Spark ジョブで垂直的自動スケーリングを使用する」を参照してください。

垂直的自動スケーリングを使用して Spark ジョブを送信する

StartJobRun API を使用してジョブを送信する場合、Spark ジョブのドライバーに以下の 2 つの設定を追加して、垂直的自動スケーリングを有効にします。

```
"spark.kubernetes.driver.annotation.emr-containers.amazonaws.com/
dynamic.sizing":"true",
"spark.kubernetes.driver.annotation.emr-containers.amazonaws.com/
dynamic.sizing.signature":"YOUR_JOB_SIGNATURE"
```

上のコードでは、最初の行で垂直的自動スケーリング機能が有効になっています。次の行は、ジョブの署名を選択するために必須の署名設定です。

これらの設定と許容されるパラメータ値の詳細については、「Amazon EMR on EKS の垂直的自動スケーリングを設定する」を参照してください。デフォルトでは、ジョブは垂直的自動スケーリングのモニタリング専用 [オフ] モードで送信されます。このモニタリング状態では、自動スケーリングを実行しなくてもリソースレコメンデーションを計算して表示できます。詳細については、「垂直的自動スケーリングモード」を参照してください。

以下の例では、垂直的自動スケーリングを使用してサンプル start-job-run コマンドを完了する方法を示しています。

```
aws emr-containers start-job-run \
--virtual-cluster-id $VIRTUAL_CLUSTER_ID \
--name $JOB_NAME \
--execution-role-arn $EMR_ROLE_ARN \
--release-label emr-6.10.0-latest \
--job-driver '{
  "sparkSubmitJobDriver": {
```

```
     "entryPoint": "local:///usr/lib/spark/examples/src/main/python/pi.py"
   }
 }' \
--configuration-overrides '{
    "applicationConfiguration": [{
        "classification": "spark-defaults",
        "properties": {
          "spark.kubernetes.driver.annotation.emr-containers.amazonaws.com/
dynamic.sizing": "true",
          "spark.kubernetes.driver.annotation.emr-containers.amazonaws.com/
dynamic.sizing.signature": "test-signature"
        }
    }]
  }'
```

垂直的自動スケーリング機能の検証

垂直的自動スケーリングが送信されたジョブで正しく機能することを確認するには、kubectl を使用して verticalpodautoscaler カスタムリソースを取得し、スケーリングの推奨事項を確認してください。例えば、以下のコマンドは、垂直的自動スケーリングを使用して Spark ジョブを送信する セクションのサンプルジョブに関するレコメンデーションを問い合わせます。

```
kubectl get verticalpodautoscalers --all-namespaces \
-l=emr-containers.amazonaws.com/dynamic.sizing.signature=test-signature
```

このクエリの出力は以下のようになります。

```
NAME                                                          MODE   CPU          MEM
 PROVIDED   AGE
ds-jceyefkxnhrvdzw6djum3naf2abm6o63a6dvjkkedqtkhlrf25eq-vpa   Off    3304504865   True
         87m
```

出力が類似していない場合やエラーコードが含まれている場合は、「Amazon EMR on EKS 垂直自動スケーリングのトラブルシューティング」を参照して、問題の解決に役立つ手順をご覧ください。

Amazon EMR on EKS の垂直的自動スケーリングを設定する

StartJobRun API を使用して Amazon EMR Spark ジョブを送信するときに、垂直的自動スケーリングを設定できます。「垂直的自動スケーリングを使用して Spark ジョブを送信する」の例に示すように、Spark ドライバーポッドに自動スケーリング関連の設定パラメータを設定します。

Amazon EMR on EKS 垂直的自動スケーリング演算子は、自動スケーリング機能を備えたドライバーポッドをリッスンし、ドライバーポッドの設定を使用して Kubernetes Vertical Pod Autoscaler (VPA) との統合を設定します。これにより、Spark エグゼキューターポッドのリソーストラッキングと自動スケーリングが容易になります。

以下のセクションでは、Amazon EKS クラスターの垂直的自動スケーリングを設定するときに使用できるパラメータについて説明します。

> **Note**
>
> 機能切り替えパラメータをラベルとして設定し、残りのパラメータを Spark ドライバーポッドのアノテーションとして設定します。自動スケーリングパラメータは `emr-containers.amazonaws.com/` ドメインに属し、`dynamic.sizing` プレフィックスが付いています。

必須パラメータ

ジョブを送信する際には、Spark ジョブドライバーに以下の 2 つのパラメータを含める必要があります。

キー	説明	使用できる値	デフォルト値	タイプ	Spark パラメータ [1]
`dynamic.sizing`	機能の切り替え	`true`, `false`	未設定	ラベル	`spark.kubernetes.driver.label.emr-containers.amazonaws.com/dynamic.sizing`
`dynamic.sizing.signature`	ジョブの署名	string	未設定	注釈	`spark.kubernetes.driver.annotation.e`

キー	説明	使用できる値	デフォルト値	タイプ	Spark パラメータ [1]
					mr-containers.amazonaws.com/dynamic.sizing.signature

[1] このパラメータを StartJobRun API で SparkSubmitParameter または ConfigurationOverride として使用します。

- **dynamic.sizing** — dynamic.sizing ラベルを使用して垂直的自動スケーリングのオンとオフを切り替えることができます。垂直的自動スケーリングをオンにするには、Spark ドライバーポッドの dynamic.sizing を true に設定します。このラベルを省略したり、true 以外の値に設定したりすると、垂直的自動スケーリングはオフになります。
- **dynamic.sizing.signature** — dynamic.sizing.signature ドライバーポッドのアノテーションを使用してジョブの署名を設定します。垂直的自動スケーリングは、Amazon EMR Spark ジョブのさまざまな実行にわたるリソース使用状況データを集約して、リソースのレコメンデーションを導き出します。ジョブを結び付けるための一意の識別子を指定します。

> **Note**
>
> ジョブが毎日あるいは毎週などの一定の間隔で繰り返される場合、ジョブの新しいインスタンスごとにジョブの署名を同じままにしておく必要があります。これにより、垂直的自動スケーリングでは、ジョブのさまざまな実行にわたるレコメンデーションを計算して集計できます。

[1] このパラメータを StartJobRun API で SparkSubmitParameter または ConfigurationOverride として使用します。

任意指定のパラメータ

垂直的自動スケーリングは、以下のオプションパラメータもサポートしています。これらをドライバーポッドのアノテーションとして設定します。

キー	説明	使用できる値	デフォルト値	タイプ	Spark パラメータ [1]
dynamic.sizing.mode	垂直的自動スケーリングモード	Off, Initial, Auto	Off	注釈	spark.kubernetes.driver.annotation.emr-containers.amazonaws.com/dynamic.sizing.mode
dynamic.sizing.scale.memory	メモリスケーリングを有効にする	*true*, *false*	true	注釈	spark.kubernetes.driver.annotation.emr-containers.amazonaws.com/dynamic.sizing.scale.memory
dynamic.sizing.scale.cpu	CPU スケーリングをオンまたはオフにする	*true*, *false*	false	注釈	spark.kubernetes.driver.annotation.emr-containers.amazonaws.com/dynamic.sizing.scale.cpu

キー	説明	使用できる値	デフォルト値	タイプ	Spark パラメータ [1]
dynamic.sizing.scale.memory.min	メモリスケーリングの最小制限	文字列、K8s リソース量 例: 1G	未設定	注釈	spark.kubernetes.driver.annotation.emr-containers.amazonaws.com/dynamic.sizing.scale.memory.min
dynamic.sizing.scale.memory.max	メモリスケーリングの最大制限	文字列、K8s リソース量 例: 4G	未設定	注釈	spark.kubernetes.driver.annotation.emr-containers.amazonaws.com/dynamic.sizing.scale.memory.max

キー	説明	使用できる値	デフォルト値	タイプ	Spark パラメータ [1]
dynamic.sizing.scale.cpu.min	CPU スケーリングの最小制限	文字列、K8s リソース量 例: 1	未設定	注釈	spark.kubernetes.driver.annotation.emr-containers.amazonaws.com/dynamic.sizing.scale.cpu.min
dynamic.sizing.scale.cpu.max	CPU スケーリングの最大制限	文字列、K8s リソース量 例: 2	未設定	注釈	spark.kubernetes.driver.annotation.emr-containers.amazonaws.com/dynamic.sizing.scale.cpu.max

垂直的自動スケーリングモード

mode パラメータは VPA がサポートするさまざまな自動スケーリングモードにマッピングされます。ドライバーポッドの dynamic.sizing.mode アノテーションを使用してモードを設定します。このパラメータでは以下の値がサポートされています。

- オフ — レコメンデーションをモニタリングできる dry-run モードですが、自動スケーリングは実行されません。これが垂直的自動スケーリングのデフォルトモードです。このモードでは、関連する Vertical Pod Autoscaler リソースがレコメンデーションを計算

し、kubectl、Prometheus、Grafana などのツールを使用してレコメンデーションをモニタリングできます。

- 初期 — このモードでは、ジョブが繰り返し実行される場合など、ジョブの実行履歴に基づいてレコメンデーションがある場合、VPA はジョブの開始時にリソースを自動スケーリングします。
- 自動 — このモードでは、VPA は Spark エグゼキューターポッドを削除し、Spark ドライバーポッドがエグゼキューターポッドを再起動したときに推奨リソース設定でそれらを自動スケーリングします。VPA は実行中の Spark エグゼキューターポッドを削除することがあるため、中断されたエグゼキューターを再試行すると、さらにレイテンシーが発生する可能性があります。

リソースのスケーリング

垂直的自動スケーリングを設定する場合は、CPU およびメモリリソースをスケーリングするかどうかを選択することができます。`dynamic.sizing.scale.cpu` および `dynamic.sizing.scale.memory` アノテーションを `true` または `false` に設定します。デフォルトでは、CPU スケーリングは `false` に設定され、メモリスケーリングは `true` に設定されます。

リソースの最小値と最大値 (境界)

必要に応じて、CPU およびメモリリソースに境界を設定することもできます。自動スケーリングを有効にするときに、`dynamic.sizing.[memory/cpu].[min/max]` アノテーション付きのこれらのリソースの最小値と最大値を選択します。デフォルトでは、リソースに制限はありません。アノテーションは Kubernetes リソース量を表す文字列値として設定します。例えば、4 GB を表すようにするには、`dynamic.sizing.memory.max` を `4G` に設定します。

Amazon EMR on EKS の垂直的自動スケーリングをモニタリングする

kubectl Kubernetes コマンドラインツールを使用して、クラスター上のアクティブな垂直的自動スケーリング関連のレコメンデーションを一覧表示できます。また、追跡したジョブの署名を表示したり、その署名に関連する不要なリソースを削除したりすることもできます。

クラスターの垂直的自動スケーリングのレコメンデーションを一覧表示する

kubectl を使用して `verticalpodautoscaler` リソースを取得し、現在のステータスとレコメンデーションを表示します。以下のクエリ例は、Amazon EKS クラスターのすべてのアクティブなリソースを返します。

```
kubectl get verticalpodautoscalers \
-o custom-columns="NAME:.metadata.name,"\
```

```
"SIGNATURE:.metadata.labels.emr-containers\.amazonaws\.com/dynamic\.sizing
\.signature,"\
"MODE:.spec.updatePolicy.updateMode,"\
"MEM:.status.recommendation.containerRecommendations[0].target.memory" \
--all-namespaces
```

このクエリの出力は以下のようになります。

```
NAME                  SIGNATURE              MODE      MEM
ds-example-id-1-vpa   job-signature-1        Off       none
ds-example-id-2-vpa   job-signature-2        Initial   12936384283
```

クラスターの垂直的自動スケーリングのレコメンデーションをクエリして削除します。

Amazon EMR 垂直的自動スケーリングジョブ実行リソースを削除すると、レコメンデーションを追跡して保存する関連付けられた VPA オブジェクトが自動的に削除されます。

以下の例では、kubectl を使用して、署名で識別されるジョブのレコメンデーションを削除します。

```
kubectl delete jobrun -n emr -l=emr-containers\.amazonaws\.com/dynamic\.sizing
\.signature=integ-test
jobrun.dynamicsizing.emr.services.k8s.aws "ds-job-signature" deleted
```

特定のジョブの署名がわからない場合や、クラスター上のリソースをすべて削除したい場合は、以下の例のように、一意のジョブ ID の代わりに `--all` または `--all-namespaces` をコマンドで使用できます。

```
kubectl delete jobruns --all --all-namespaces
jobrun.dynamicsizing.emr.services.k8s.aws "ds-example-id" deleted
```

Amazon EMR on EKS の垂直的自動スケーリング演算子をアンインストールする

Amazon EKS クラスターから垂直的自動スケーリング演算子を削除する場合は、以下の例に示すように Operator SDK CLI で `cleanup` コマンドを使用します。これにより、Vertical Pod Autoscaler など、演算子と共にインストールされたアップストリームの依存関係も削除されます。

```
operator-sdk cleanup emr-dynamic-sizing
```

演算子を削除したときにクラスター上に実行中のジョブがある場合、それらのジョブは垂直的自動スケーリングなしで実行され続けます。演算子を削除した後にクラスターでジョブを送信すると、Amazon EMR on EKS は、設定時に定義した垂直的自動スケーリング関連のパラメータをすべて無視します。

EKS での Amazon EMR 上のインタラクティブワークロードの実行

インタラクティブエンドポイントは、Amazon EMR Studio を EKS での Amazon EMR に接続するゲートウェイです。これにより、インタラクティブワークロードを実行できます。EMR Studio でインタラクティブエンドポイントを使用すると、Amazon S3 や Amazon DynamoDB などのデータストアにあるデータセットを使用してインタラクティブな分析を実行できます。

ユースケース

- EMR スタジオ 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。IDE はオンプレミスデータを取り込み、変換後に Amazon S3 に保存し、後で分析できるようにします。
- ノートブックを使用してデータセットを調べ、データセットの異常を検出するように機械学習モデルをトレーニングします。
- ビジネスダッシュボードなどの分析アプリケーションの日次レポートを生成するスクリプトを作成します。

トピック

インタラクティブエンドポイントの概要

インタラクティブエンドポイントは、Amazon EMR Studio などのインタラクティブクライアントが EKS クラスター上の Amazon EMR に接続してインタラクティブワークロードを実行できるようにします。インタラクティブエンドポイントは、インタラクティブクライアントが必要とするリモートカーネルライフサイクル管理機能を提供する Jupyter Enterprise Gateway によって支えられてい

ます。カーネルは、Jupyter ベースの Amazon EMR Studio クライアントと相互作用してインタラクティブワークロードを実行する言語固有のプロセスです。

インタラクティブエンドポイントは、次のカーネルをサポートします。

- Python 3
- Kubernetes での PySpark
- Scala を使用した Apache Spark

> **Note**
>
> EKS での Amazon EMR の料金は、インタラクティブエンドポイントとカーネルに適用されます。詳細については、「Amazon EMR on EKS pricing」のページを参照してください。

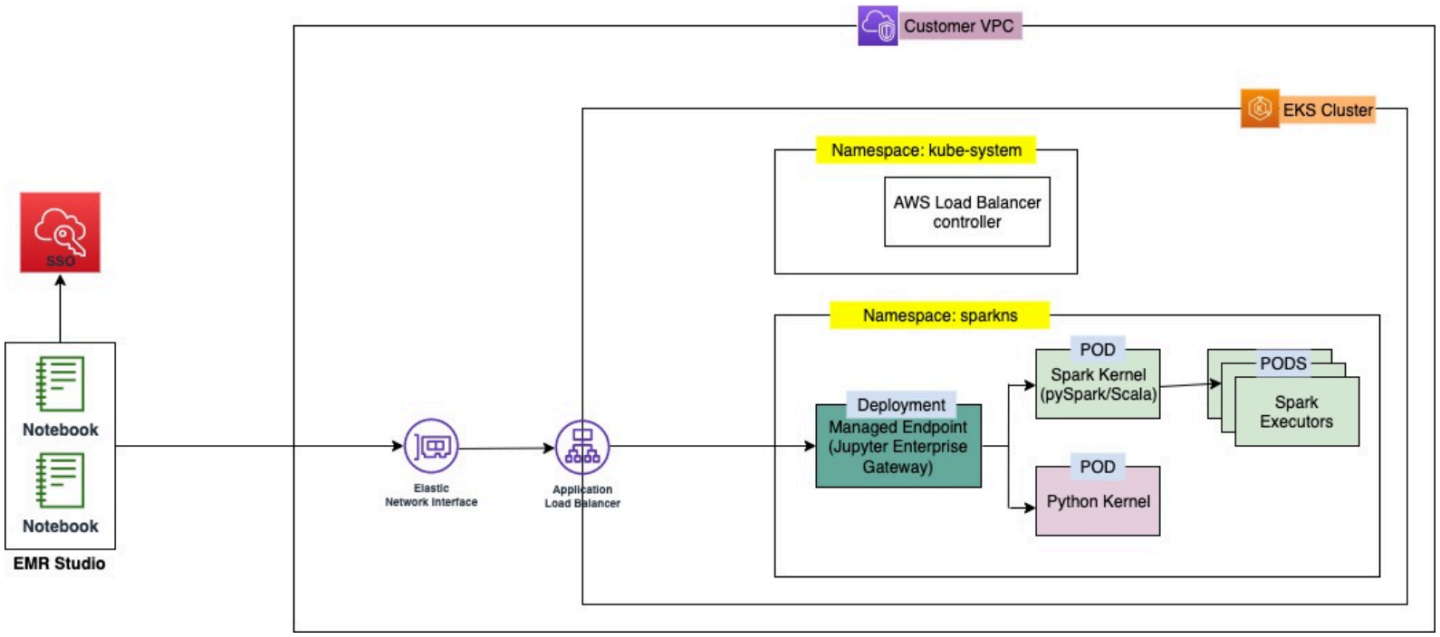
EMR Studio を EKS での Amazon EMR に接続するには、以下のエンティティが必要です。

- EKS での Amazon EMR 仮想クラスター — 仮想クラスターは、Amazon EMR を登録する際に使用する Kubernetes 名前空間です。Amazon EMR では、仮想クラスターを使用してジョブを実行し、エンドポイントをホストします。複数の仮想クラスターを同じ物理クラスターでバックアップできます。ただし、各仮想クラスターは Amazon EKS クラスター上の 1 つの名前空間にマッピングされます。仮想クラスターでは、請求に適用されるアクティブなリソースや、サービスの外部でライフサイクル管理を必要とするアクティブなリソースは作成されません。
- EKS での Amazon EMR のインタラクティブエンドポイント — インタラクティブエンドポイントは、EMR Studio ユーザーがワークスペースに接続できる HTTPS エンドポイントです。HTTPS エンドポイントには EMR Studio からのみアクセスでき、Amazon EKS クラスターの Amazon Virtual Private Cloud (Amazon VPC) のプライベートサブネットにこれを作成します。

  Python、PySpark、および Spark Scala カーネルは、EKS での Amazon EMR ジョブ実行ロールで定義されたアクセス許可を使用して、他の AWS のサービスを呼び出します。インタラクティブエンドポイントに接続するすべてのカーネルとユーザーは、エンドポイントの作成時に指定したロールを使用します。ユーザーごとに個別のエンドポイントを作成し、ユーザーが異なる AWS Identity and Access Management (IAM) ロールを持つことをお勧めします。
- AWS Application Load Balancer コントローラー – AWS Application Load Balancer コントローラーは、Amazon EKS Kubernetes クラスターの Elastic Load Balancing を管理します。コントローラーは、Kubernetes Ingress リソースを作成するときに、Application Load Balancer (ALB) をプロビジョニングします。ALB は、インタラクティブエンドポイントなどの Kubernetes サービ

スを Amazon EKS クラスターの外部に公開しますが、同じ Amazon VPC 内に公開します。インタラクティブエンドポイントを作成すると、インタラクティブクライアントが接続できるように ALB を使用してインタラクティブエンドポイントを公開する Ingress リソースもデプロイされます。Amazon EKS クラスターごとに 1 つの AWS Application Load Balancer コントローラーのみをインストールする必要があります。

次の図に、EKS での Amazon EMR のインタラクティブエンドポイントのアーキテクチャを示します。Amazon EKS クラスターは、分析ワークロードを実行するコンピューティングとインタラクティブエンドポイントで構成されます。Application Load Balancer コントローラーは `kube-system` 名前空間で実行されます。ワークロードとインタラクティブエンドポイントは、仮想クラスターの作成時に指定した名前空間で実行されます。インタラクティブエンドポイントを作成すると、EKS での Amazon EMR コントロールプレーンは Amazon EKS クラスターにインタラクティブエンドポイントデプロイを作成します。さらに、Application Load Balancer Ingress のインスタンスは AWS 、ロードバランサーコントローラーによって作成されます。Application Load Balancer は、EMR Studio などのクライアントが Amazon EMR クラスターに接続し、インタラクティブワークロードを実行するための外部インターフェイスを提供します。



EKS での Amazon EMR でインタラクティブエンドポイントを作成するための前提条件

このセクションでは、EMR Studio が EKS での Amazon EMR クラスターに接続してインタラクティブワークロードを実行するために使用できるインタラクティブエンドポイントを設定するための前提条件について説明します。

AWS CLI

「 の最新バージョンをインストールまたは更新する AWS CLI」の手順に従って、 () の最新バージョンをインストールしますAWS CLI。 AWS Command Line Interface

eksctl のインストール

最新バージョンの eksctl をインストールするには、「kubectl をインストールする」の手順を行います。Amazon EKS クラスターに Kubernetes バージョン 1.22 以降を使用している場合は、0.117.0 より大きいバージョンの eksctl を使用してください。

Amazon EKS クラスター

Amazon EKS クラスターを作成します。クラスターを EKS での Amazon EMR に仮想クラスターとして登録します。このクラスターの要件と考慮事項は次のとおりです。

- クラスターは EMR Studio と同じ Amazon Virtual Private Cloud (VPC) 内にある必要があります。

- クラスターには、インタラクティブエンドポイントの有効化、Git ベースのリポジトリのリンク、および Application Load Balancer のプライベートモードでの起動のために、少なくとも 1 つのプライベートサブネットが必要です。
- EMR Studio と、仮想クラスターの登録に使用する Amazon EKS クラスターの間に少なくとも 1 つの共通のプライベートサブネットが必要です。これにより、インタラクティブエンドポイントが Studio ワークスペースにオプションとして表示され、Studio から Application Load Balancer への接続が有効になります。

  Studio と Amazon EKS クラスターを接続するには、2 つの方法から選択できます。
  - Amazon EKS クラスターを作成し、EMR Studio に属するサブネットに関連付けます。
  - または、EMR Studio を作成し、Amazon EKS クラスターのプライベートサブネットを指定します。

- Amazon EKS 最適化 ARM Amazon Linux AMI は、EKS での Amazon EMR インタラクティブエンドポイントではサポートされていません。
- インタラクティブエンドポイントは、バージョン 1.30 までの Kubernetes を使用する Amazon EKS クラスターで動作します。
- Amazon EKS マネージドノードグループのみがサポートされます。

EKS での Amazon EMR のクラスターアクセスを許可する

「EKS での Amazon EMR のクラスターアクセスを許可する」のステップを使用して、EKS での Amazon EMR にクラスター内の特定の名前空間へのアクセスを許可します。

Amazon EKS クラスターでの IRSA の有効化

Amazon EKS クラスターでサービスアカウント (IRSA) の IAM ロールを有効化するには、「Enable IAM Roles for Service Accounts (IRSA)」の手順に従ってください。

IAM ジョブ実行ロールを作成する

EKS での Amazon EMR インタラクティブエンドポイントでワークロードを実行するには、IAM ロールを作成する必要があります。このドキュメントでは、この IAM ロールをジョブ実行ロールと呼びます。この IAM ロールは、インタラクティブエンドポイントコンテナと、EMR Studio でジョブを送信したときに作成される実際の実行コンテナの両方に割り当てられます。EKS での Amazon EMR のジョブ実行ロールの Amazon リソースネーム (ARN) が必要になります。これには次の 2 つのステップが必要です。

- ジョブ実行の IAM ロールを作成する。
- ジョブ実行ロールの信頼ポリシーを更新する。

Amazon EMR on EKS へのアクセス許可をユーザーに付与する

インタラクティブエンドポイントの作成のリクエストを行う IAM エンティティ (ユーザーまたはロール) には、次の Amazon EC2 および `emr-containers` アクセス権限も必要です。「Amazon EMR on EKS へのアクセス許可をユーザーに付与する」で説明した手順に従って、EKS での Amazon EMR が、インタラクティブエンドポイントのロードバランサーへのインバウンドトラフィックを制限するセキュリティグループを作成、管理、および削除できるようにするアクセス権限を付与します。

以下の emr-containers アクセス権限により、ユーザーは基本的なインタラクティブエンドポイントオペレーションを実行できます。

```
"ec2:CreateSecurityGroup",
"ec2:DeleteSecurityGroup",
"ec2:AuthorizeSecurityGroupEgress",
"ec2:AuthorizeSecurityGroupIngress",
"ec2:RevokeSecurityGroupEgress",
"ec2:RevokeSecurityGroupIngress"

"emr-containers:CreateManagedEndpoint",
"emr-containers:ListManagedEndpoints",
"emr-containers:DescribeManagedEndpoint",
"emr-containers:DeleteManagedEndpoint"
```

Amazon EKS クラスターをAmazon EMR に登録する

仮想クラスターを設定し、ジョブを実行する Amazon EKS クラスター内の名前空間にマッピングします。 AWS Fargate専用クラスターの場合は、Amazon EMR on EKS 仮想クラスターと Fargate プロファイルの両方に同じ名前空間を使用します。

EKS での Amazon EMR 仮想クラスターの設定の詳細については、「Amazon EKS クラスターを Amazon EMR に登録する」を参照してください。

Amazon EKS クラスターへの Deploy AWS Load Balancer Controller

Amazon EKS クラスターには AWS Application Load Balancer が必要です。Amazon EKS クラスターごとに 1 つの Application Load Balancer コントローラーを設定する必要があります。 AWS Application Load Balancer コントローラーの設定については、「Amazon EKS ユーザーガイド」の AWS Load Balancer Controller アドオンのインストール」を参照してください。

仮想クラスターのインタラクティブエンドポイントを作成する

このトピックでは、 コマンドラインインターフェイス (AWS CLI) AWS を使用してインタラクティブエンドポイントを作成するいくつかの方法と、使用可能な設定パラメータの詳細について説明します。

create-managed-endpoint コマンドでインタラクティブエンドポイントを作成する

次のように create-managed-endpoint コマンドでパラメータを指定します。EKS での Amazon EMR は、Amazon EMR リリース 6.7.0 以降でのインタラクティブエンドポイントの作成をサポートしています。

```
aws emr-containers create-managed-endpoint \
--type JUPYTER_ENTERPRISE_GATEWAY \
--virtual-cluster-id 1234567890abcdef0xxxxxxxx \
--name example-endpoint-name \
--execution-role-arn arn:aws:iam::444455556666:role/JobExecutionRole \
--release-label emr-6.9.0-latest \
--configuration-overrides '{
    "applicationConfiguration": [{
        "classification": "spark-defaults",
        "properties": {
            "spark.driver.memory": "2G"
        }
    }],
    "monitoringConfiguration": {
        "cloudWatchMonitoringConfiguration": {
            "logGroupName": "log_group_name",
            "logStreamNamePrefix": "log_stream_prefix"
        },
        "persistentAppUI": "ENABLED",
        "s3MonitoringConfiguration": {
            "logUri": "s3://my_s3_log_location"
        }
    }
}'
```

詳細については、「インタラクティブエンドポイントを作成するためのパラメータ」を参照してください。

JSON ファイルで指定されたパラメータを使用してインタラクティブエンドポイントを作成する

1. create-managed-endpoint-request.json ファイルを作成し、次の JSON ファイルに示すように、エンドポイントに必要なパラメータを指定します。

```
{
    "name": "MY_TEST_ENDPOINT",
    "virtualClusterId": "MY_CLUSTER_ID",
    "type": "JUPYTER_ENTERPRISE_GATEWAY",
    "releaseLabel": "emr-6.9.0-latest",
    "executionRoleArn": "arn:aws:iam::444455556666:role/JobExecutionRole",
    "configurationOverrides":
    {
        "applicationConfiguration":
        [
            {
                "classification": "spark-defaults",
                "properties":
                {
                    "spark.driver.memory": "8G"
                }
            }
        ],
        "monitoringConfiguration":
        {
            "persistentAppUI": "ENABLED",
            "cloudWatchMonitoringConfiguration":
            {
                "logGroupName": "my_log_group",
                "logStreamNamePrefix": "log_stream_prefix"
            },
            "s3MonitoringConfiguration":
            {
                "logUri": "s3://my_s3_log_location"
            }
        }
    }
}
```

2. ローカルまたは Amazon S3 に保存されている `create-managed-endpoint-request.json` ファイルへのパスを指定して、`create-managed-endpoint` コマンドを使用します。

```
aws emr-containers create-managed-endpoint \
--cli-input-json  file://./create-managed-endpoint-request.json --region AWS-Region
```

インタラクティブエンドポイントの作成の出力

ターミナルで、次の出力が表示されます。出力には、新しいインタラクティブエンドポイントの名前と ID が含まれます。

```
{
    "id": "1234567890abcdef0",
    "name": "example-endpoint-name",
    "arn": "arn:aws:emr-containers:us-west-2:111122223333:/
virtualclusters/444455556666/endpoints/444455556666",
    "virtualClusterId": "111122223333xxxxxxxx"
}
```

`aws emr-containers create-managed-endpoint` を実行すると、EMR Studio とインタラクティブエンドポイントサーバー間の HTTPS 通信を可能にする自己署名証明書が作成されます。

`create-managed-endpoint` を実行しても前提条件を満たしていない場合、Amazon EMR は続行するために必要なアクションを含むエラーメッセージを返します。

インタラクティブエンドポイントを作成するためのパラメータ

トピック

インタラクティブエンドポイントの必須パラメータ

インタラクティブエンドポイントを作成するときに次のパラメータを指定する必要があります。

`--type`

JUPYTER_ENTERPRISE_GATEWAY を使用します。これはサポートされている唯一のタイプです。

`--virtual-cluster-id`

EKS での Amazon EMR に登録した仮想クラスターの ID。

`--name`

EMR Studio ユーザーがドロップダウンリストから選択しやすいようにする、インタラクティブエンドポイントのわかりやすい名前。

`--execution-role-arn`

前提条件の一部として作成された EKS での Amazon EMR の IAM ジョブ実行 IAM ロールの Amazon リソースネーム (ARN)。

`--release-label`

エンドポイントに使用する Amazon EMR リリースのリリースラベル。例えば、`emr-6.9.0-latest` と指定します。EKS での Amazon EMR は、Amazon EMR リリース 6.7.0 以降でインタラクティブエンドポイントをサポートしています。

インタラクティブエンドポイントの任意指定のパラメータ

インタラクティブエンドポイントを作成するときに、オプションで以下を指定することもできます。

`--configuration-overrides`

アプリケーションのデフォルト設定を上書きするには、設定オブジェクトを指定します。短縮構文を使用して、設定を指定したり、JSON ファイルの設定オブジェクトを参照したりできます。

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。単一の 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。EKS での Amazon EMR リリースによって使用可能な設定分類は異なります。EKS での Amazon EMR の各リリースで使用可能な設定分類の一覧については、「Amazon EMR on EKS リリース」を参照してください。各リリースにリストされている設定分類に加えて、インタラクティブエンドポイントには追加の分類 `jeg-config` が組み込まれています。詳細については、「Jupyter エンタープライズゲートウェイ (JEG) 設定オプション」を参照してください。

インタラクティブエンドポイントの設定

このセクションでは、インタラクティブエンドポイントとポッド設定のさまざまな設定をカバーする一連のトピックについて説明します。これにより、障害のモニタリングとトラブルシューティング、Amazon S3 または への口グ情報の送信 Amazon CloudWatch Logs、カスタムポッドテンプレートを指定するインタラクティブエンドポイントの作成を行うことができます。

トピック

- モニタリングSpark ジョブ

- インタラクティブエンドポイントでカスタムポッドテンプレートを指定する
- JEG ポッドのノードグループへのデプロイ
- Jupyter エンタープライズゲートウェイ (JEG) 設定オプション
- PySpark セッションパラメータの変更
- インタラクティブエンドポイントを含むカスタムカーネルイメージ

モニタリングSpark ジョブ

障害のモニタリングとトラブルシューティングをできるようにするには、エンドポイントで起動したジョブで Amazon S3、Amazon CloudWatch Logs、またはその両方に口グ情報を送信できるようにインタラクティブエンドポイントを設定します。以下のセクションでは、EKS での Amazon EMR インタラクティブエンドポイントで起動した Spark ジョブの Amazon S3 へ Spark アプリケーションログを送信する方法について説明します。

Amazon S3 口グ用の IAM ポリシーを設定する

カーネルが Amazon S3 に口グデータを送信できるようにするには、ジョブ実行口ールのアクセス権限ポリシーに次のアクセス権限を含める必要があります。*amzn-s3-demo-destination-bucket* を口グ記録バケットの名前に置き換えます。

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

> **Note**
>
> EKS での Amazon EMR は、S3 バケットを作成することもできます。S3 バケットが利用できない場合は、IAM ポリシーに `s3:CreateBucket` アクセス権限を含めてください。

S3 バケットにログを送信するために必要なアクセス権限を実行ロールに付与すると、ログデータは次の Amazon S3 ロケーションに送信されます。これは、`s3MonitoringConfiguration` が `create-managed-endpoint` リクエストの `monitoringConfiguration` セクションで渡されるときに起こります。

- ドライバーログ — `logUri/virtual-cluster-id/endpoints/endpoint-id/containers/spark-application-id/spark-application-id-driver/(stderr.gz/stdout.gz)`
- エグゼキューターログ — `logUri/virtual-cluster-id/endpoints/endpoint-id/containers/spark-application-id/executor-pod-name-exec-<Number>/(stderr.gz/stdout.gz)`

> **Note**
>
> EKS での Amazon EMR は、エンドポイントログを S3 バケットにアップロードしません。

インタラクティブエンドポイントでカスタムポッドテンプレートを指定する

ドライバーとエグゼキューター用のカスタムポッドテンプレートを指定するインタラクティブエンドポイントを作成できます。ポッドテンプレートは、各ポッドの実行方法を決定する仕様です。ポッドテンプレートファイルを使用して、Spark 構成でサポートされていないドライバーまたはエグゼキューターポッドの設定を定義できます。ポッドテンプレートは現在、Amazon EMR リリース 6.3.0 以降でサポートされています。

ポッドテンプレートの詳細については、「Amazon EMR on EKS Development Guide」の「Using pod templates」を参照してください。

次の例は、ポッドテンプレートを使用してインタラクティブエンドポイントを作成する方法を示しています。

```
aws emr-containers create-managed-endpoint \
    --type JUPYTER_ENTERPRISE_GATEWAY \
    --virtual-cluster-id virtual-cluster-id \
    --name example-endpoint-name \
    --execution-role-arn arn:aws:iam::aws-account-id:role/EKSClusterRole \
    --release-label emr-6.9.0-latest \
    --configuration-overrides '{
        "applicationConfiguration": [
        {
            "classification": "spark-defaults",
            "properties": {
                "spark.kubernetes.driver.podTemplateFile": "path/to/driver/
template.yaml",
                "spark.kubernetes.executor.podTemplateFile": "path/to/executor/
template.yaml"
            }
        }]
    }'
```

JEG ポッドのノードグループへのデプロイ

JEG (Jupyter Enterprise Gateway) ポッド配置は、特定のノードグループにインタラクティブエンドポイントをデプロイできるようにする機能です。この機能により、インタラクティブエンドポイントの `instance type` などの設定を行うことができます。

マネージドノードグループに JEG ポッドを関連付ける

次の設定プロパティでは、JEG ポッドをデプロイする Amazon EKS クラスター上のマネージドノードグループの名前を指定できます。

```
//payload
--configuration-overrides '{
      "applicationConfiguration": [
            {
                "classification": "endpoint-configuration",
                "properties": {
                    "managed-nodegroup-name": NodeGroupName
                }
            }
        ]
    }'
```

ノードグループには、ノードグループに含まれるすべてのノードに Kubernetes ラベル `for-use-with-emr-containers-managed-endpoint-ng=NodeGroupName` がアタッチされている必要があります。このタグを持つノードグループのすべてのノードをリストするには、次のコマンドを使用します。

```
kubectl get nodes --show-labels | grep for-use-with-emr-containers-managed-endpoint-ng=NodeGroupName
```

上記のコマンドの出力でマネージドノードグループの一部であるノードが返されない場合は、`for-use-with-emr-containers-managed-endpoint-ng=NodeGroupName` Kubernetes ラベルがアタッチされたノードがノードグループにありません。この場合、以下の手順に従って、そのラベルをノードグループ内のノードにアタッチします。

1. 次のコマンドを使用して、マネージドノードグループ *NodeGroupName* のすべてのノードに `for-use-with-emr-containers-managed-endpoint-ng=NodeGroupName` Kubernetes ラベルを追加します。

   ```
   kubectl label nodes --selector eks:nodegroup-name=NodeGroupName for-use-with-emr-containers-managed-endpoint-ng=NodeGroupName
   ```

2. 次のコマンドを使用して、ノードが正しくラベル付けされていることを確認します。

   ```
   kubectl get nodes --show-labels | grep for-use-with-emr-containers-managed-endpoint-ng=NodeGroupName
   ```

マネージドノードグループは Amazon EKS クラスターのセキュリティグループに関連付ける必要があります。通常、`eksctl` を使用してクラスターとマネージドノードグループを作成した場合にこれが当てはまります。これは、次の手順を使用して AWS コンソールで確認できます。

1. Amazon EKS コンソールでクラスターに移動します。
2. クラスターの [ネットワーキング] タブに移動し、クラスターセキュリティグループを書き留めます。
3. クラスターの [コンピューティング] タブに移動し、マネージドノードグループ名をクリックします。
4. マネージドノードグループの [詳細] タブで、前にメモしたクラスターセキュリティグループが [セキュリティグループ] に表示されていることを確認します。

マネージドノードグループが Amazon EKS クラスターセキュリティグループにアタッチされていない場合は、`for-use-with-emr-containers-managed-endpoint-sg=`*`ClusterName`*`/`*`NodeGroupName` タグをノードグループのセキュリティグループにアタッチする必要があります。このタグをアタッチするには、次の手順を使用します。

1. Amazon EC2 コンソールに移動し、左側のナビゲーションペインで [セキュリティグループ] をクリックします。
2. チェックボックスをクリックして、マネージドノードグループのセキュリティグループを選択します。
3. [タグ] タブの [タグの管理] ボタンを使用してタグ `for-use-with-emr-containers-managed-endpoint-sg=`*`ClusterName`*`/`*`NodeGroupName` を追加します。

セルフマネージドノードグループに JEG ポッドを関連付ける

次の設定プロパティでは、JEG ポッドがデプロイされる Amazon EKS クラスター上のセルフマネージドノードグループまたはアンマネージドノードグループの名前を指定できます。

```
//payload
--configuration-overrides '{
      "applicationConfiguration": [
            {
                "classification": "endpoint-configuration",
                "properties": {
                    "self-managed-nodegroup-name": NodeGroupName
                }
            }
        ]
    }'
```

ノードグループには、ノードグループに含まれるすべてのノードに `for-use-with-emr-containers-managed-endpoint-ng=`*`NodeGroupName`* Kubernetes ラベルがアタッチされている必要があります。このタグを持つノードグループのすべてのノードをリストするには、次のコマンドを使用します。

```
kubectl get nodes --show-labels | grep for-use-with-emr-containers-managed-endpoint-ng=NodeGroupName
```

上記のコマンドの出力でセルフマネージドノードグループの一部であるノードが返されない場合は、`for-use-with-emr-containers-managed-endpoint-ng=`*`NodeGroupName`* Kubernetes

ラベルがアタッチされたノードがノードグループにありません。この場合、以下の手順に従って、そのラベルをノードグループ内のノードにアタッチします。

1. eksctl を使用してセルフマネージドノードグループを作成した場合は、以下のコマンドを使用して、セルフマネージドノードグループ *NodeGroupName* のすべてのノードに for-use-with-emr-containers-managed-endpoint-ng=*NodeGroupName* Kubernetes ラベルを一度に追加します。

   ```
   kubectl label nodes --selector alpha.eksctl.io/nodegroup-name=NodeGroupName for-use-with-emr-containers-managed-endpoint-ng=NodeGroupName
   ```

   セルフマネージドノードグループの作成に eksctl を使用しなかった場合は、上記のコマンドのセレクターを、ノードグループのすべてのノードにアタッチされている別の Kubernetes ラベルに置き換える必要があります。
2. 次のコマンドを使用して、ノードが正しくラベル付けされていることを確認します。

   ```
   kubectl get nodes --show-labels | grep for-use-with-emr-containers-managed-endpoint-ng=NodeGroupName
   ```

セルフマネージドノードグループのセキュリティグループには for-use-with-emr-containers-managed-endpoint-sg=*ClusterName*/*NodeGroupName* タグがアタッチされている必要があります。次の手順に従って、 AWS Management Consoleからセキュリティグループにタグをアタッチします。

1. Amazon EC2 コンソールに移動します。左側のナビゲーションペインで [セキュリティグループ] を選択します。
2. セルフマネージドノードグループのセキュリティグループの横にあるチェックボックスをオンにします。
3. [タグ] タブの [タグの管理] ボタンを使用してタグ for-use-with-emr-containers-managed-endpoint-sg=*ClusterName*/*NodeGroupName* を追加します。*ClusterName* と *NodeGroupName* を適切な値に置き換えます。

オンデマンドインスタンスがあるマネージドノードグループに JEG ポッドを関連付ける

Kubernetes ラベルセレクターと呼ばれる追加のラベルを定義して、特定のノードまたはノードグループでインタラクティブエンドポイントを実行するための追加の制約や制限を指定することもできます。次の例は、JEG ポッドにオンデマンド Amazon EC2 インスタンスを使用する方法を示しています。

```
--configuration-overrides '{
      "applicationConfiguration": [
            {
                "classification": "endpoint-configuration",
                "properties": {
                    "managed-nodegroup-name": NodeGroupName,
                    "node-labels": "eks.amazonaws.com/capacityType:ON_DEMAND"
                }
            }
        ]
    }'
```

> **Note**
>
> `node-labels` プロパティは、`managed-nodegroup-name` または `self-managed-nodegroup-name` プロパティでのみ使用できます。

Jupyter エンタープライズゲートウェイ (JEG) 設定オプション

EKS での Amazon EMR は Jupyter エンタープライズゲートウェイ (JEG) を使用してインタラクティブエンドポイントを有効にします。エンドポイントを作成するときに、許可リストに登録されている JEG 設定に次の値を設定できます。

- **`RemoteMappingKernelManager.cull_idle_timeout`** — 秒単位 (整数) のタイムアウト。この時間が過ぎると、カーネルはアイドル状態になり、カリングできる状態になります。`0` 以下の値はカリングを無効にします。タイムアウトが短いと、ネットワーク接続が不十分なユーザーのカーネルがカリングされる可能性があります。
- **`RemoteMappingKernelManager.cull_interval`** — カリングタイムアウト値を超えるアイドル状態のカーネルをチェックする秒単位の間隔 (整数)。

PySpark セッションパラメータの変更

EKS での Amazon EMR リリース 6.9.0 以降、Amazon EMR Studio では EMR ノートブックセルで %%configure マジックコマンドを実行することで、PySpark セッションに関連付けられた Spark 設定を調整できるようになりました。

次の例は、Spark ドライバーとエグゼキューターのメモリ、コア、その他のプロパティを変更するために使用できるサンプルペイロードを示しています。conf 設定では、Apache Spark の設定ドキュメントで説明されている任意の Spark 設定を構成できます。

```
%%configure -f
{
  "driverMemory": "16G",
  "driverCores" 4,
  "executorMemory" : "32G"
  "executorCores": 2,
  "conf": {
     "spark.dynamicAllocation.maxExecutors" : 10,
     "spark.dynamicAllocation.minExecutors": 1
  }
}
```

次の例は、ファイル、pyFiles、および jar 依存関係を Spark ランタイムに追加するために使用できるサンプルペイロードを示しています。

```
%%configure -f
{
  "files": "s3://amzn-s3-demo-bucket-emr-eks/sample_file.txt",
  "pyFiles": : "path-to-python-files",
  "jars" : "path-to-jars
}
```

インタラクティブエンドポイントを含むカスタムカーネルイメージ

Amazon EMR Studio からインタラクティブワークロードを実行するときにアプリケーションに適切な依存関係があることを確認するには、インタラクティブエンドポイントの Docker イメージをカスタマイズし、カスタマイズされたベースカーネルイメージを実行します。インタラクティブエンドポイントを作成してカスタム Docker イメージに接続するには、以下の手順を実行します。

> **Note**
>
> 上書きできるのはベースイメージだけです。新しいカーネルイメージタイプを追加することはできません。

1. カスタマイズした Docker イメージを作成して公開します。ベースイメージには、Spark ランタイムと、Spark ランタイムで実行されるノートブックのカーネルが含まれています。イメージを作成するには、Docker イメージをカスタマイズする方法 のステップ 1～4 に従います。ステップ 1 では、Docker ファイル内のベースイメージ URI で、notebook-spark を spark の代わりに使用する必要があります。

   ```
   ECR-registry-account.dkr.ecr.Region.amazonaws.com/notebook-spark/container-image-tag
   ```

   AWS リージョン およびコンテナイメージタグの選択方法の詳細については、「」を参照してくださいベースイメージ URI の選択の詳細。

2. カスタムイメージで使用できるインタラクティブエンドポイントを作成します。

   a. 次の内容を含む JSON ファイル custom-image-managed-endpoint.json を作成します。この例では、Amazon EMR リリース 6.9.0 を使用します。

      Example

      ```
      {
          "name": "endpoint-name",
          "virtualClusterId": "virtual-cluster-id",
          "type": "JUPYTER_ENTERPRISE_GATEWAY",
          "releaseLabel": "emr-6.9.0-latest",
          "executionRoleArn": "execution-role-arn",
          "configurationOverrides": {
              "applicationConfiguration": [
                  {
                      "classification": "jupyter-kernel-overrides",
                      "configurations": [
                          {
                              "classification": "python3",
                              "properties": {
                                  "container-image": "123456789012.dkr.ecr.us-west-2.amazonaws.com/custom-notebook-python:latest"
      ```

```
                        }
                    },
                    {
                        "classification": "spark-python-kubernetes",
                        "properties": {
                            "container-image": "123456789012.dkr.ecr.us-
west-2.amazonaws.com/custom-notebook-spark:latest"
                        }
                    }
                ]
            }
        ]
    }
}
```

b. 次の例で示すように、JSON ファイルで指定された設定を使用してインタラクティブエンドポイントを作成します。詳細については、「create-managed-endpoint コマンドでインタラクティブエンドポイントを作成する」を参照してください。

```
aws emr-containers create-managed-endpoint --cli-input-json custom-image-
managed-endpoint.json
```

3. EMR Studio 経由でインタラクティブエンドポイントに接続します。詳細と完了手順については、 AWS Workshop Studio ドキュメントの Amazon EMR on EKS セクションの「Connecting from Studio」を参照してください。

インタラクティブエンドポイントのモニタリング

EKS での Amazon EMR 6.10 以降では、インタラクティブエンドポイントは、カーネルのライフサイクルオペレーションのモニタリングとトラブルシューティングのための Amazon CloudWatch メトリクスを生成します。メトリクスは、EMR Studio やセルフホストの Jupyter Notebook などのインタラクティブクライアントによってトリガーされます。インタラクティブエンドポイントがサポートする各操作には、メトリクスが関連付けられています。以下の表に示すように、操作は各メトリクスのディメンションとしてモデル化されます。インタラクティブエンドポイントによって生成されたメトリクスは、アカウントのカスタム名前空間 EMRContainers に表示されます。

メトリクス	説明	[単位]
RequestCount	インタラクティブエンドポイントによって処理されたオペ	カウント

メトリクス	説明	[単位]
	レーションのリクエストの累積数。	
RequestLatency	リクエストがインタラクティブエンドポイントに到着し、インタラクティブエンドポイントからレスポンスが送信されるまでの時間。	ミリ秒
4XXError	操作のリクエストが処理中に 4xx エラーになった場合に生成されます。	カウント
5XXError	操作のリクエストが 5Xxx サーバー側エラーになった場合に生成されます。	カウント
KernelLaunchSuccess	CreateKernel 操作にのみ適用されます。このリクエストを含む、このリクエストまでに成功したカーネル起動の累積数を示します。	カウント
KernelLaunchFailure	CreateKernel 操作にのみ適用されます。このリクエストを含む、このリクエストまでに失敗したカーネル起動の累積数を示します。	カウント

各インタラクティブエンドポイントメトリクスには、次のディメンションが関連付けられています。

- **ManagedEndpointId**— インタラクティブエンドポイントの識別子
- **OperationName**— インタラクティブクライアントによってトリガーされる操作

OperationName ディメンションに指定できる値を次の表に示します。

operationName	操作の説明
CreateKernel	インタラクティブエンドポイントにカーネルの起動をリクエストします。
ListKernels	インタラクティブエンドポイントに、同じセッショントークンを使用して以前に起動されたカーネルを一覧表示するようにリクエストします。
GetKernel	インタラクティブエンドポイントに、以前に起動された特定のカーネルの詳細を取得するようにリクエストします。
ConnectKernel	インタラクティブエンドポイントに、ノートブッククライアントとカーネル間の接続を確立するようにリクエストします。
ConfigureKernel	pyspark カーネルで %%configure magic request を公開します。
ListKernelSpecs	インタラクティブエンドポイントに、利用可能なカーネル仕様を一覧表示するようリクエストします。
GetKernelSpec	インタラクティブエンドポイントに、以前に起動したカーネルのカーネル仕様を取得するようリクエストします。
GetKernelSpecResource	インタラクティブエンドポイントに、以前に起動されたカーネル仕様に関連する特定のリソースを取得するようにリクエストします。

例

特定の日にインタラクティブエンドポイントで起動されたカーネルの総数を確認するには

1. カスタム名前空間を選択します: EMRContainers
2. ManagedEndpointId、OperationName – CreateKernel を選択します。
3. 統計 SUM と期間 1 day を含む RequestCount メトリクスは、過去 24 時間以内に行われたすべてのカーネル起動リクエストを提供します。
4. 統計 SUM と期間 1 day を含む KernelLaunchSuccess メトリクスは、過去 24 時間以内に行われたすべての成功したカーネル起動リクエストを提供します。

特定の日にインタラクティブエンドポイントで発生したカーネル障害の数を確認するには

1. カスタム名前空間を選択します: EMRContainers
2. ManagedEndpointId、OperationName – CreateKernel を選択します。
3. 統計 SUM と期間 1 day を含む KernelLaunchFailure メトリクスは、過去 24 時間以内に行われたすべての失敗したカーネル起動リクエストを提供します。また、4XXError と 5XXError メトリクスを選択して、どのようなカーネル起動エラーが発生したかを確認することもできます。

セルフホスト型 Jupyter Notebook を使用する

Jupyter または JupyterLab Notebook は、Amazon EC2 インスタンスまたは独自の Amazon EKS クラスターでセルフホスト型 Jupyter Notebook としてホストおよび管理できます。その後、セルフホスト型 Jupyter Notebook でインタラクティブワークロードを実行できます。以下のセクションでは、セルフホスト型 Jupyter Notebook を Amazon EKS クラスターにセットアップしてデプロイするプロセスについて説明します。

EKS クラスターでのセルフホスト型 Jupyter Notebook の作成

- セキュリティグループの作成
- EKS での Amazon EMR インタラクティブエンドポイントを作成する
- インタラクティブエンドポイントのゲートウェイサーバー URL を取得する
- 認証トークンを取得してインタラクティブエンドポイントに接続します。

- 例: JupyterLab ノートブックをデプロイする
- セルフホスト型 Jupyter Notebook を削除する

セキュリティグループの作成

インタラクティブエンドポイントを作成し、セルフホスト型 Jupyter または JupyterLab ノートブックを実行する前に、ノートブックとインタラクティブエンドポイント間のトラフィックを制御するセキュリティグループを作成する必要があります。Amazon EC2 コンソールまたは Amazon EC2 SDK を使用してセキュリティグループを作成するには、「Amazon EC2 ユーザーガイド」の「セキュリティグループの作成」の手順を参照してください。ノートブックサーバーをデプロイする VPC にセキュリティグループを作成する必要があります。

このガイドの例に従うには、Amazon EKS クラスターと同じ VPC を使用してください。Amazon EKS クラスターの VPC とは異なる VPC でノートブックをホストする場合、これらの 2 つの VPC 間にピアリング接続を作成する必要がある場合があります。2 つの VPC 間にピアリング接続を作成する手順については、「Amazon VPC Getting Started Guide」の「Create a VPC peering connection」を参照してください。

次のステップで EKS での Amazon EMR インタラクティブエンドポイントを作成するには、セキュリティグループの ID が必要です。

EKS での Amazon EMR インタラクティブエンドポイントを作成する

ノートブックのセキュリティグループを作成したら、「仮想クラスターのインタラクティブエンドポイントを作成する」に記載されている手順に従ってインタラクティブエンドポイントを作成します。「セキュリティグループの作成」でノートブック用に作成したセキュリティグループ ID を指定する必要があります。

次の構成オーバーライド設定で、*your-notebook-security-group-id* の代わりにセキュリティ ID を挿入します。

```
--configuration-overrides '{
    "applicationConfiguration": [
        {
            "classification": "endpoint-configuration",
            "properties": {
                "notebook-security-group-id": "your-notebook-security-group-id"
            }
        }
```

```
    ],
    "monitoringConfiguration": {
    ...'
```

インタラクティブエンドポイントのゲートウェイサーバー URL を取得する

インタラクティブエンドポイントを作成したら、 AWS CLIで describe-managed-endpoint コマンドを使用してゲートウェイサーバーの URL を取得します。この URL は、ノートブックをエンドポイントに接続するために必要です。ゲートウェイサーバー URL はプライベートエンドポイントです。

```
aws emr-containers describe-managed-endpoint \
--region region \
--virtual-cluster-id virtualClusterId \
--id endpointId
```

最初、エンドポイントの状態は CREATING です。数分後、ACTIVE 状態に遷移します。エンドポイントが ACTIVE となっていれば、使用準備が完了しています。

serverUrl コマンドがアクティブなエンドポイントから返す aws emr-containers describe-managed-endpoint 属性を書き留めておきます。セルフホスト型 Jupyter または JupyterLab ノートブックをデプロイするときに、ノートブックをエンドポイントに接続するには、この URL が必要です。

認証トークンを取得してインタラクティブエンドポイントに接続します。

Jupyter または JupyterLab ノートブックからインタラクティブエンドポイントに接続するには、GetManagedEndpointSessionCredentials API を使用してセッショントークンを生成する必要があります。このトークンは、インタラクティブエンドポイントサーバーに接続するための認証証明として機能します。

次のコマンドについて、出力例を示してさらに詳しく説明します。

```
aws emr-containers get-managed-endpoint-session-credentials \
--endpoint-identifier endpointArn \
--virtual-cluster-identifier virtualClusterArn \
--execution-role-arn executionRoleArn \
--credential-type "TOKEN" \
--duration-in-seconds durationInSeconds \
--region region
```

endpointArn

エンドポイントの ARN。ARN は `describe-managed-endpoint` 呼び出しの結果で確認できます。

virtualClusterArn

仮想クラスターの ARN。

executionRoleArn

実行ロールの ARN。

durationInSeconds

トークンの有効期間 (秒単位)。デフォルトの有効期間は 15 分 (`900`) で、最長は 12 時間 (`43200`) です。

region

エンドポイントと同じリージョン。

出力は次の例のようになります。セルフホスト型 Jupyter または JupyterLab ノートブックをデプロイするときに使用する *session-token* 値を書き留めておいてください。

```
{
    "id": "credentialsId",
    "credentials": {
        "token": "session-token"
    },
    "expiresAt": "2022-07-05T17:49:38Z"
}
```

例: JupyterLab ノートブックをデプロイする

上記のステップを完了したら、このサンプル手順を試して、インタラクティブエンドポイントを使用して JupyterLab ノートブックを Amazon EKS クラスターにデプロイできます。

1. ノートブックサーバーを実行する名前空間を作成します。
2. `notebook.yaml` というファイルを次の内容でローカルに作成します。ファイルの内容は次のとおりです。

```
apiVersion: v1
```

```
kind: Pod
metadata:
  name: jupyter-notebook
  namespace: namespace
spec:
  containers:
  - name: minimal-notebook
    image: jupyter/all-spark-notebook:lab-3.1.4 # open source image
    ports:
    - containerPort: 8888
    command: ["start-notebook.sh"]
    args: ["--LabApp.token=''"]
    env:
    - name: JUPYTER_ENABLE_LAB
      value: "yes"
    - name: KERNEL_LAUNCH_TIMEOUT
      value: "400"
    - name: JUPYTER_GATEWAY_URL
      value: "serverUrl"
    - name: JUPYTER_GATEWAY_VALIDATE_CERT
      value: "false"
    - name: JUPYTER_GATEWAY_AUTH_TOKEN
      value: "session-token"
```

Jupyter Notebook を Fargate 専用クラスターにデプロイする場合は、次の例のように Jupyter ポッドに `role` ラベルを付けます。

```
...
metadata:
  name: jupyter-notebook
  namespace: default
  labels:
    role: example-role-name-label
spec:
            ...
```

namespace

ノートブックをデプロイする Kubernetes 名前空間。

serverUrl

インタラクティブエンドポイントのゲートウェイサーバー URL を取得する で describe-managed-endpoint コマンドが返した serverUrl 属性。

session-token

認証トークンを取得してインタラクティブエンドポイントに接続します。 で get-managed-endpoint-session-credentials コマンドが返した session-token 属性。

KERNEL_LAUNCH_TIMEOUT

インタラクティブエンドポイントが、カーネルが RUNNING 状態になるのを待つ時間 (秒単位)。カーネル起動タイムアウトを適切な値 (最大 400 秒) に設定して、カーネルの起動が完了するまで十分な時間を確保します。

KERNEL_EXTRA_SPARK_OPTS

オプションで、Spark カーネルに追加の Spark 構成を渡すことができます。以下の例のように、この環境変数に Spark 設定プロパティとして値を設定します。

```
- name: KERNEL_EXTRA_SPARK_OPTS
  value: "--conf spark.driver.cores=2
          --conf spark.driver.memory=2G
          --conf spark.executor.instances=2
          --conf spark.executor.cores=2
          --conf spark.executor.memory=2G
          --conf spark.dynamicAllocation.enabled=true
          --conf spark.dynamicAllocation.shuffleTracking.enabled=true
          --conf spark.dynamicAllocation.minExecutors=1
          --conf spark.dynamicAllocation.maxExecutors=5
          --conf spark.dynamicAllocation.initialExecutors=1
          "
```

3. Amazon EKS クラスターにポッドの仕様をデプロイします。

```
kubectl apply -f notebook.yaml -n namespace
```

これにより、EKS での Amazon EMR インタラクティブエンドポイントに接続された最小限の JupyterLab ノートブックが起動します。ポッドが RUNNING になるまでお待ちください。次のコマンドを使用して、ポッドのステータスを確認します。

```
kubectl get pod jupyter-notebook -n namespace
```

ポッドの準備が整うと、get pod コマンドは次のような出力を返します。

```
NAME              READY   STATUS    RESTARTS   AGE
jupyter-notebook  1/1     Running   0          46s
```

4. ノートブックセキュリティグループを、ノートブックがスケジュールされているノードにアタッチします。

   a. まず、jupyter-notebook ポッドがスケジュールされているノードを describe pod コマンドで特定します。

   ```
   kubectl describe pod jupyter-notebook -n namespace
   ```

   b. Amazon EKS コンソール (https://console.aws.amazon.com/eks/home#/clusters) を開きます。

   c. Amazon EKS クラスターの [コンピューティング] タブに移動し、describe pod コマンドで特定したノードを選択します。ノードのインスタンス ID を選択します。

   d. [アクション] メニューから [セキュリティ] > [セキュリティグループを変更] を選択し、セキュリティグループの作成 で作成したセキュリティグループをアタッチします。

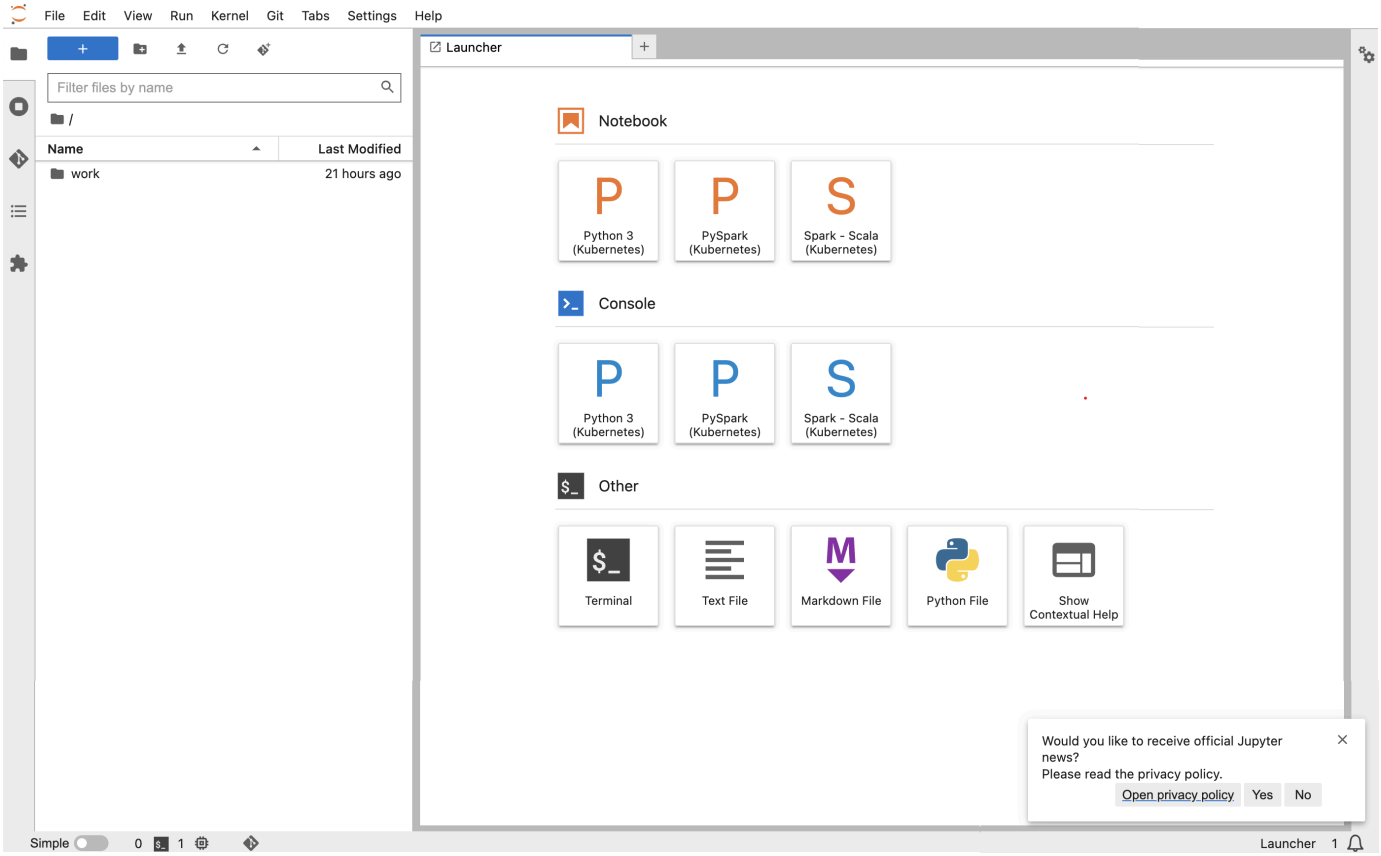
   e. Jupyter Notebook ポッドを にデプロイする場合は AWS Fargate、 ロールラベルを使用して Jupyter Notebook ポッドSecurityGroupPolicyに適用する を作成します。

   ```
   cat >my-security-group-policy.yaml <<EOF
   apiVersion: vpcresources.k8s.aws/v1beta1
   kind: SecurityGroupPolicy
   metadata:
     name: example-security-group-policy-name
     namespace: default
   spec:
     podSelector:
       matchLabels:
         role: example-role-name-label
     securityGroups:
       groupIds:
         - your-notebook-security-group-id
   EOF
   ```

5. 次に、ポート転送して JupyterLab インターフェイスにローカルにアクセスできるようにします。

```
kubectl port-forward jupyter-notebook 8888:8888 -n namespace
```

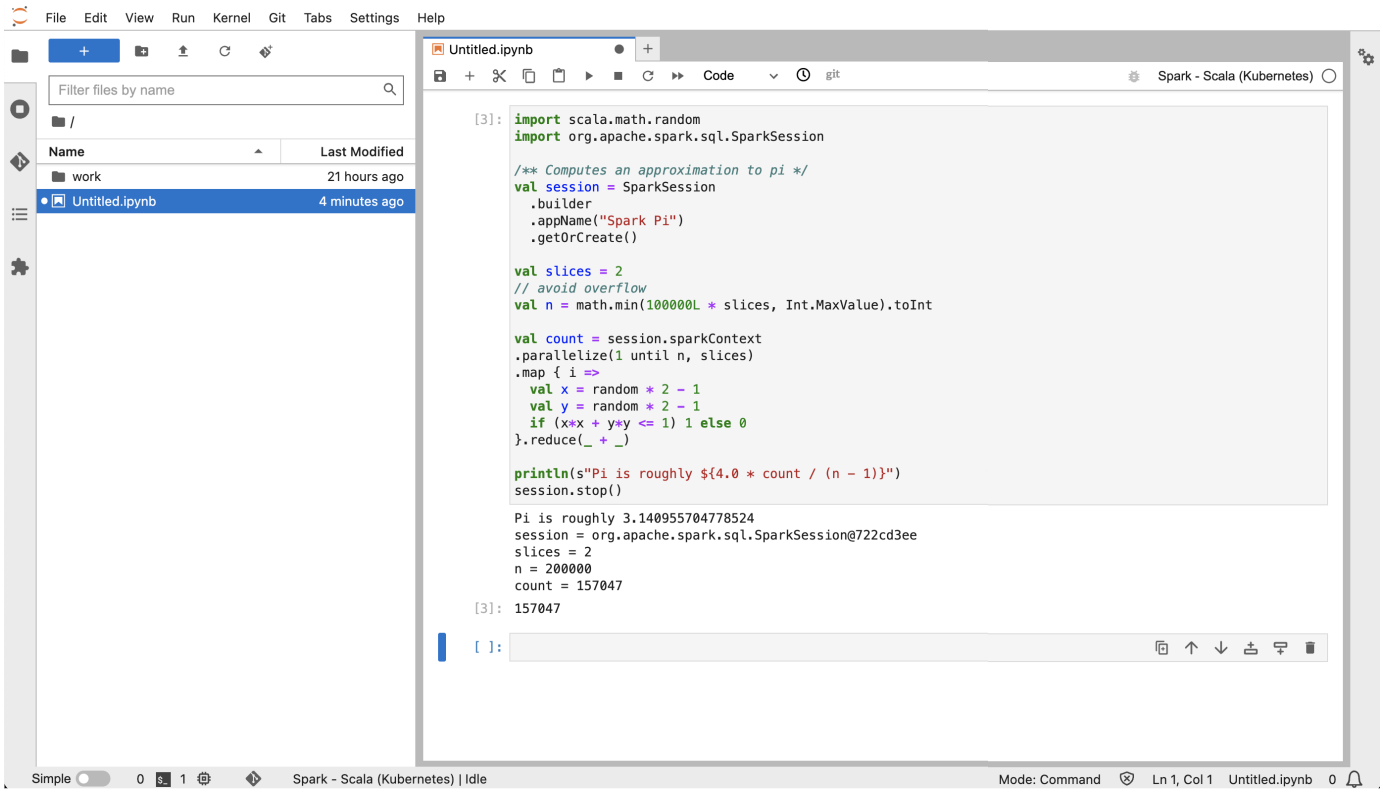
それが実行したら、ローカルブラウザに移動し、`localhost:8888` にアクセスして JupyterLab インターフェイスを確認してください。



6. JupyterLab から、新しい Scala ノートブックを作成します。Pi の値を概算するために実行できるサンプルコードスニペットを次に示します。

```
import scala.math.random
import org.apache.spark.sql.SparkSession

/** Computes an approximation to pi */
val session = SparkSession
  .builder
  .appName("Spark Pi")
  .getOrCreate()

val slices = 2
```

```
// avoid overflow
val n = math.min(100000L * slices, Int.MaxValue).toInt

val count = session.sparkContext
.parallelize(1 until n, slices)
.map { i =>
  val x = random * 2 - 1
  val y = random * 2 - 1
  if (x*x + y*y <= 1) 1 else 0
}.reduce(_ + _)

println(s"Pi is roughly ${4.0 * count / (n - 1)}")
session.stop()
```



セルフホスト型 Jupyter Notebook を削除する

セルフホスト型ノートブックを削除する準備ができたら、インタラクティブエンドポイントとセキュリティグループも削除できます。次の順番でアクションを実行します。

1. 次のコマンドを使用して、jupyter-notebook ポッドを削除します。

```
kubectl delete pod jupyter-notebook -n namespace
```

2. 次に、delete-managed-endpoint コマンドを使用してインタラクティブエンドポイントを削除します。インタラクティブエンドポイントを削除する手順については、「インタラクティブエンドポイントを削除する」を参照してください。最初、エンドポイントは TERMINATING 状態になります。すべてのリソースがクリーンアップされると、TERMINATED 状態に移行します。
3. セキュリティグループの作成 で作成したノートブックセキュリティグループを他の Jupyter Notebook デプロイに使用する予定がない場合は、削除できます。詳細については、「Amazon EC2 User Guide」の「Delete a security group」を参照してください。

CLI コマンドを使用したインタラクティブエンドポイントに関する情報の取得

このトピックでは、create-managed-endpoint 以外のインタラクティブエンドポイントでサポートされるオペレーションについて説明します。

インタラクティブエンドポイントの詳細を取得する

インタラクティブエンドポイントを作成したら、 describe-managed-endpoint AWS CLI コマンドを使用してその詳細を取得できます。*managed-endpoint-id*、*virtual-cluster-id*、および *region* に独自の値を挿入してください。

```
aws emr-containers describe-managed-endpoint --id managed-endpoint-id \
 --virtual-cluster-id virtual-cluster-id --region region
```

出力は、ARN、ID、名前などの指定されたエンドポイントを含む次のようになります。

```
{
   "id": "as3ys2xxxxxxx",
   "name": "endpoint-name",
    "arn": "arn:aws:emr-containers:us-east-1:1828xxxxxxxx:/virtualclusters/
lbhl6kwwyoxxxxxxxxxxxxxxx/endpoints/as3ysxxxxxxxx",
    "virtualClusterId": "lbhl6kwwyoxxxxxxxxxxxxxxx",
    "type": "JUPYTER_ENTERPRISE_GATEWAY",
    "state": "ACTIVE",
    "releaseLabel": "emr-6.9.0-latest",
   "executionRoleArn": "arn:aws:iam::1828xxxxxxxx:role/RoleName",
```

```
    "certificateAuthority": {
        "certificateArn": "arn:aws:acm:us-east-1:1828xxxxxxxx:certificate/zzzzzzzz-
e59b-4ed0-aaaa-bbbbbbbbbbbb",
        "certificateData": "certificate-data"
    },
    "configurationOverrides": {
        "applicationConfiguration": [
            {
                "classification": "spark-defaults",
                "properties": {
                    "spark.driver.memory": "8G"
                }
            }
        ],
        "monitoringConfiguration": {
            "persistentAppUI": "ENABLED",
            "cloudWatchMonitoringConfiguration": {
                "logGroupName": "log-group-name",
                "logStreamNamePrefix": "log-stream-name-prefix"
            },
            "s3MonitoringConfiguration": {
                "logUri": "s3-bucket-name"
            }
        }
    },
   "serverUrl": "https://internal-k8s-namespace-ingressa-aaaaaaaaaa-
zzzzzzzzzz.us-east-1.elb.amazonaws.com:18888 (https://internal-k8s-nspluto-
ingressa-51e860abbd-1620715833.us-east-1.elb.amazonaws.com:18888/)",
    "createdAt": "2022-09-19T12:37:49+00:00",
    "securityGroup": "sg-aaaaaaaaaaaaaa",
    "subnetIds": [
        "subnet-11111111111",
        "subnet-22222222222",
        "subnet-33333333333"
    ],
    "stateDetails": "Endpoint created successfully. It took 3 Minutes 15 Seconds",
    "tags": {}
 }
```

仮想クラスターに関連付けられたすべてのインタラクティブエンドポイントをリストする

list-managed-endpoints AWS CLI コマンドを使用して、指定された仮想クラスターに関連付けられているすべてのインタラクティブエンドポイントのリストを取得します。virtual-cluster-id は、仮想クラスターの ID に置き換えます。

```
aws emr-containers list-managed-endpoints --virtual-cluster-id virtual-cluster-id
```

次のような list-managed-endpoint コマンドの出力が表示されます。

```
{
    "endpoints": [{
        "id": "as3ys2xxxxxxx",
        "name": "endpoint-name",
        "arn": "arn:aws:emr-containers:us-east-1:1828xxxxxxxx:/virtualclusters/
lbhl6kwwyoxxxxxxxxxxxxxxx/endpoints/as3ysxxxxxxxx",
        "virtualClusterId": "lbhl6kwwyoxxxxxxxxxxxxxxx",
        "type": "JUPYTER_ENTERPRISE_GATEWAY",
        "state": "ACTIVE",
        "releaseLabel": "emr-6.9.0-latest",
        "executionRoleArn": "arn:aws:iam::1828xxxxxxxx:role/RoleName",
        "certificateAuthority": {
            "certificateArn": "arn:aws:acm:us-east-1:1828xxxxxxxx:certificate/zzzzzzzz-
e59b-4ed0-aaaa-bbbbbbbbbbbb",
            "certificateData": "certificate-data"
        },
        "configurationOverrides": {
            "applicationConfiguration": [{
                "classification": "spark-defaults",
                "properties": {
                    "spark.driver.memory": "8G"
                }
            }],
            "monitoringConfiguration": {
                "persistentAppUI": "ENABLED",
                "cloudWatchMonitoringConfiguration": {
                    "logGroupName": "log-group-name",
                    "logStreamNamePrefix": "log-stream-name-prefix"
                },
                "s3MonitoringConfiguration": {
                    "logUri": "s3-bucket-name"
```

```
                }
            }
        },
        "serverUrl": "https://internal-k8s-namespace-ingressa-aaaaaaaaaa-
zzzzzzzzzz.us-east-1.elb.amazonaws.com:18888 (https://internal-k8s-nspluto-
ingressa-51e860abbd-1620715833.us-east-1.elb.amazonaws.com:18888/)",
        "createdAt": "2022-09-19T12:37:49+00:00",
        "securityGroup": "sg-aaaaaaaaaaaaaa",
        "subnetIds": [
            "subnet-11111111111",
            "subnet-22222222222",
            "subnet-33333333333"
        ],
        "stateDetails": "Endpoint created successfully. It took 3 Minutes 15 Seconds",
        "tags": {}
    }]
}
```

インタラクティブエンドポイントを削除する

Amazon EMR on EKS 仮想クラスターに関連付けられているインタラクティブエンドポイントを削除するには、 `delete-managed-endpoint` AWS CLI コマンドを使用します。インタラクティブエンドポイントを削除すると、EKS での Amazon EMR はそのエンドポイント用に作成されたデフォルトのセキュリティグループを削除します。

コマンドに次のパラメータの値を指定します。

- --id: 削除するインタラクティブエンドポイントの識別子。
- --virtual-cluster-id - 削除するインタラクティブエンドポイントに関連付けられている仮想クラスターの識別子。これは、インタラクティブエンドポイントの作成時に指定された仮想クラスター ID と同じです。

```
aws emr-containers delete-managed-endpoint --id managed-endpoint-id --virtual-cluster-
id virtual-cluster-id
```

このコマンドにより、インタラクティブエンドポイントが削除されたことを確認するために、次のような出力が返されます。

```
{
    "id":"8gai4l4exxxxx",
```

```
    "virtualClusterId":"0b0qvauoy3ch1nqodxxxxxxxx"
}
```

Amazon EMR on EKS を使用して Amazon S3 Express One Zone にデータをアップロードする

リリース 7.2.0 以降の Amazon EMR では、Amazon EMR on EKS を Amazon S3 Express One Zone ストレージクラスと組み合わせて使用することで、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 1 桁のミリ秒単位で一貫したデータアクセスを提供する、高パフォーマンスの単一ゾーンの 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

Amazon EMR on EKS で S3 Express One Zone を使用するには、次の前提条件を満たしている必要があります。

- Amazon EMR on EKS のセットアップの完了。
- Amazon EMR on EKS をセットアップしたら、仮想クラスターを作成します。

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. `CreateSession` アクセス許可をジョブの実行ロールに追加します。S3 Express One Zone が S3 オブジェクトに対して `GET`、`LIST`、`PUT` などのアクションを最初に実行すると、ストレージクラスがユーザーに代わって `CreateSession` を呼び出します。次の例は、`CreateSession` 権限を付与する方法を示しています。

```
{
  "Version":"2012-10-17",
  "Statement": [
        {
            "Effect": "Allow",
            "Resource": "arn:aws:s3express:<AWS_REGION>:<ACCOUNT_ID>:bucket/DOC-EXAMPLE-BUCKET",
            "Action": [
                "s3express:CreateSession"
```

```
            ]
        }
    ]
}
```

2. Apache Hadoop コネクタ S3A を使用して S3 Express バケットにアクセスする必要があるため、Amazon S3 URI を変更して s3a スキームを使用してコネクタを使用する必要があります。このスキームを使用していない場合は、s3 スキームと s3n スキームに使用するファイルシステム実装を変更できます。

   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"
    }
  }
]
```

3. spark-submit 設定で、ウェブ ID 認証情報プロバイダーを使用します。

```
"spark.hadoop.fs.s3a.aws.credentials.provider=com.amazonaws.auth.WebIdentityTokenCredential
```

ジョブのモニタリング

Amazon CloudWatch Events を使用して、Amazon EMR on EKS 仮想クラスターで実行されるジョブを追跡できます。イベントを使用して、仮想クラスターで実行しているジョブのアクティビティと正常性を追跡できます。以下の各トピックでは、リソースの正常性を維持するためにモニタリングを効果的に設定する方法を示します。

トピック

Amazon CloudWatch Events でジョブをモニタリングする

Amazon EMR on EKS では、ジョブ実行の状態が変化したときにイベントが出力されます。各イベントが提供する情報には、そのイベントが発生した日時のほか、仮想クラスターの ID や影響を受けたジョブ実行の ID などの詳細が含まれています。

イベントを使用して、仮想クラスターで実行しているジョブのアクティビティと正常性を追跡できます。また、Amazon CloudWatch Events を使用すると、指定したパターンと一致するイベントがジョブ実行によって生成された場合に実行するアクションを定義できます。イベントは、ジョブ実行のライフサイクル中に特定の発生をモニタリングするのに便利です。例えば、ジョブ実行の状態が `submitted` から `running` に変化するタイミングをモニタリングできます。CloudWatch イベントの詳細については、「Amazon EventBridge ユーザーガイド」を参照してください。

次の表は、Amazon EMR on EKS のイベント、イベントが示す状態や状態の変化、イベントの重要度、イベントメッセージを示しています。各イベントは、イベントのストリームに自動的に送信される JSON オブジェクトとして表されます。JSON オブジェクトにはイベントの詳細が含まれます。JSON オブジェクトは、CloudWatch Events を使用してイベント処理のルールを設定する場合に特に重要です。ルールでは JSON オブジェクトでパターンのマッチングが行われるからです。詳細については、「Amazon EventBridge ユーザーガイド」の「Amazon EventBridge のイベントパターン」と「Amazon EMR on EKS イベント」を参照してください。

ジョブ実行の状態変更イベント

状態	緊急度	メッセージ
SUBMITTED	INFO	*Time* UTC に、ジョブ実行 *JobRunId* (*JobRunName*) が仮想クラスター *VirtualClusterId* に正常に送信されました。
RUNNING	INFO	仮想クラスター *VirtualClusterId* 内のジョブ実行 *JobRunId* (*JobRunName*) が、*Time* に実行を開始しました。
COMPLETED	INFO	仮想クラスター *VirtualClusterId* 内のジョブ実行 *jobRunId* (*JobRunName*) が *Time* に完了しました。ジョブ実行は *Time* に実行を開始し、完了するまでに *Num* 分かかりました。
CANCELLED	WARN	仮想クラスター *VirtualClusterId* 内のジョブ実行 *JobRunId* (*JobRunName*) のキャンセルリクエストが *Time* に成功し、ジョブ実行がキャンセルされました。
FAILED	ERROR	仮想クラスター *VirtualClusterId* 内のジョブ実行 *JobRunId* (*JobRunName*) が *Time* に失敗しました。

CloudWatch Events で Amazon EMR on EKS を自動化する

Amazon CloudWatch Events を使用すると、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対応するために、 AWS サービスを自動化できます。 AWS サービスからのイベントは、ほぼリアルタイムで CloudWatch Events に配信されます。どのイベントに興味があるのか、イベントがルールに一致した場合にどのように自動的に実行するアクションをとるのか簡単なルールを指定して書き込みすることができます。自動的にトリガーできるオペレーションには、以下が含まれます。

- AWS Lambda 関数の呼び出し

- Amazon EC2 Run Command の呼び出し
- Amazon Kinesis Data Streams へのイベントの中継
- AWS Step Functions ステートマシンのアクティブ化
- Amazon Simple Notification Service (SNS) トピックまたは Amazon Simple Queue Service (SQS) キューの通知

Amazon EMR on EKS で CloudWatch Events を使用する例は次のとおりです。

- ジョブ実行が成功したときの Lambda 関数のアクティブ化
- ジョブ実行が失敗したときの Amazon SNS トピックの通知

`"detail-type:" "EMR Job Run State Change"` の CloudWatch Events は、`SUBMITTED`、`RUNNING`、`CANCELLED`、`FAILED`、`COMPLETED` の状態が変化したときに Amazon EMR on EKS によって生成されます。

例: Lambda を呼び出すルールを設定する

次の手順に従って、"EMR Job Run State Change" イベントの発生時に Lambda を呼び出す CloudWatch Events ルールを設定します。

```
aws events put-rule \
--name cwe-test \
--event-pattern '{"detail-type": ["EMR Job Run State Change"]}'
```

次のように、所有する Lambda 関数を新しいターゲットとして追加し、Lambda 関数を呼び出すためのアクセス許可を CloudWatch Events に付与します。*123456789012* は自分のアカウント ID に置き換えます。

```
aws events put-targets \
--rule cwe-test \
--targets Id=1,Arn=arn:aws:lambda:us-east-1:123456789012:function:MyFunction
```

```
aws lambda add-permission \
--function-name MyFunction \
--statement-id MyId \
--action 'lambda:InvokeFunction' \
```

```
--principal events.amazonaws.com
```

> **Note**
>
> 通知イベントは順番が間違っていたり欠落していたりする可能性があるため、通知イベントの順序や有無に依存するプログラムは作成できません。イベントはベストエフォートベースで発生します。

Amazon CloudWatch Events を使用して再試行ポリシーでジョブのドライバーポッドを監視する

CloudWatch イベントを使用すると、再試行ポリシーのあるジョブで作成されたドライバーポッドを監視できます。詳細については、このガイドの「再試行ポリシーを使用してジョブをモニタリングする」を参照してください。

仮想クラスターの管理

仮想クラスターとは、Amazon EMR が登録されている Kubernetes 名前空間です。仮想クラスターを作成、説明、一覧表示、および削除できます。システム内の追加のリソースは消費されません。1 つの仮想クラスターは、1 つの Kubernetes 名前空間にマップされます。この関係により、Kubernetes 名前空間をモデル化するのと同じ方法で仮想クラスターをモデル化して、要件を満たすことができます。Kubernetes 概念概要 ドキュメントで、考えられるユースケースを参照してください。

Amazon EMR を Amazon EKS クラスターの Kubernetes 名前空間に登録するには、EKS クラスターの名前と、ワークロードを実行するためにセットアップされた名前空間が必要です。Amazon EMR に登録されたこれらのクラスターは、物理的なコンピューティングやストレージを管理するのではなく、ワークロードがスケジュールされている Kubernetes 名前空間を指しているため、仮想クラスターと呼ばれます。

> **Note**
>
> 仮想クラスターを作成する前に、まず「Amazon EMR on EKS のセットアップ」のステップ 1 ~ 8 を完了する必要があります。

トピック

仮想クラスターを作成する

次のコマンドを実行して、Amazon EMR を EKS クラスター上の名前空間に登録し、仮想クラスターを作成します。*virtual_cluster_name* は、仮想クラスターに指定する名前に置き換えます。*eks_cluster_name* は、EKS クラスターの名前に置き換えます。*namespace_name* は、Amazon EMR を登録する名前空間に置き換えます。

```
aws emr-containers create-virtual-cluster \
--name virtual_cluster_name \
--container-provider '{
    "id": "eks_cluster_name",
    "type": "EKS",
    "info": {
        "eksInfo": {
            "namespace": "namespace_name"
        }
    }
}'
```

または、次の例に示すように、仮想クラスターに必要なパラメータを含む JSON ファイルを作成することもできます。

```
{
    "name": "virtual_cluster_name",
    "containerProvider": {
        "type": "EKS",
        "id": "eks_cluster_name",
        "info": {
            "eksInfo": {
                "namespace": "namespace_name"
            }
        }
    }
}
```

JSON ファイルへのパスを指定して、次の `create-virtual-cluster` コマンドを実行します。

```
aws emr-containers create-virtual-cluster \
--cli-input-json file://./create-virtual-cluster-request.json
```

> **Note**
>
> 仮想クラスターが正常に作成されたことを確認するには、`list-virtual-clusters` コマンドを実行するか、Amazon EMR コンソールで [仮想クラスター] ページに移動して、仮想クラスターのステータスを確認します。

仮想クラスターを一覧表示する

次のコマンドを実行して、仮想クラスターのステータスを表示します。

```
aws emr-containers list-virtual-clusters
```

仮想クラスターを説明する

次のコマンドを実行して、名前空間、ステータス、登録日など、仮想クラスターの詳細を取得します。*123456* は、仮想クラスター ID に置き換えます。

```
aws emr-containers describe-virtual-cluster --id 123456
```

仮想クラスターを削除する

次のコマンドを実行して、仮想クラスターを削除します。*123456* は、仮想クラスター ID に置き換えます。

```
aws emr-containers delete-virtual-cluster --id 123456
```

仮想クラスターの状態

次の表には、仮想クラスターの 4 つの状態が示されています。

State	説明
RUNNING	仮想クラスターは RUNNING 状態です。
TERMINATING	仮想クラスターの要求された終了処理が進行中です。
TERMINATED	要求された終了処理が完了しています。
ARRESTED	要求された終了処理は、アクセス許可が不十分なため、失敗しました。

Amazon EMR on EKS のチュートリアル

このセクションでは、Amazon EMR on EKS アプリケーションで作業するときによく使用されるユースケースについて説明します。各アプリケーションは専用であり、設定するための独自の手順を実施できます。これらのトピックでは、各アプリケーションを使用する手順について説明します。

トピック

Amazon EMR on EKS での Delta Lake の使用

Delta Lake は、Lakehouse アーキテクチャを構築するためのオープンソースストレージフレームワークです。使用するためにセットアップする方法を以下に示しています。

Amazon EMR on EKS アプリケーションで Delta Lake を使用するには

1. アプリケーション設定でジョブ実行を開始して Spark ジョブを送信するときに、Delta Lake JAR ファイルを含めます。

```
--job-driver '{"sparkSubmitJobDriver" : {
      "sparkSubmitParameters" : "--jars local:///usr/share/aws/delta/lib/delta-
core.jar,local:///usr/share/aws/delta/lib/delta-storage.jar,local:///usr/share/aws/
delta/lib/delta-storage-s3-dynamodb.jar"}}'
```

> **Note**
>
> Amazon EMR リリース 7.0.0 以降では、Delta Lake 3.0 が使用され、`delta-core.jar` の名前が `delta-spark.jar` に変更されます。Amazon EMR リリース 7.0.0 以降を使用する場合は、次の例のように、正しいファイル名を使用してください。
>
> ```
> --jars local:///usr/share/aws/delta/lib/delta-spark.jar
> ```

2. Delta Lake の追加設定を含め、 Glue データカタログ AWS をメタストアとして使用します。

```
--configuration-overrides '{
        "applicationConfiguration": [
        {
          "classification" : "spark-defaults",
          "properties" : {
            "spark.sql.extensions" : "io.delta.sql.DeltaSparkSessionExtension",

 "spark.sql.catalog.spark_catalog":"org.apache.spark.sql.delta.catalog.DeltaCatalog",
"spark.hadoop.hive.metastore.client.factory.class":"com.amazonaws.glue.catalog.metastore.AW
           }
        }]}'
```

Amazon EMR on EKS での Apache Iceberg の使用

Iceberg のランタイム JAR には、Spark ランタイムのサポートに必要な Iceberg クラスが含まれています。次の手順は、Iceberg スパークランタイムを使用してジョブ実行を開始する方法を示しています。

Amazon EMR on EKS アプリケーションで Apache Iceberg を使用するには

1. アプリケーション設定でジョブ実行を開始して Spark ジョブを送信するときに、Iceberg Spark ランタイム JAR ファイルを含めます。

```
--job-driver '{"sparkSubmitJobDriver" : {"sparkSubmitParameters" : "--jars
 local:///usr/share/aws/iceberg/lib/iceberg-spark3-runtime.jar"}}'
```

2. さらに別の Iceberg 設定を含めます。

```
--configuration-overrides '{
    "applicationConfiguration": [
    "classification" : "spark-defaults",
    "properties" : {
        "spark.sql.catalog.dev.warehouse" : "s3://amzn-s3-demo-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.sql.catalog.dev.io-impl": "org.apache.iceberg.aws.s3.S3FileIO"
        }
    ]
}'
```

EMR の Apache Iceberg リリースバージョンの詳細については、「Iceberg release history」を参照してください。

カタログ統合用の Spark セッション設定

Iceberg Glue カタログ統合の Spark AWS セッション設定

このサンプルでは、Iceberg を次の と統合する方法を示します AWS Glue クローラー。

```
spark-sql \
  --conf spark.sql.catalog.rms = org.apache.iceberg.spark.SparkCatalog \
  --conf spark.sql.catalog.rms.type = glue \
  --conf spark.sql.catalog.rms.glue.id = glue RMS catalog ID \
  --conf spark.sql.catalog.rms.glue.account-id = AWS account ID \

  --conf spark.sql.extensions=
    org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions
```

サンプルクエリを以下に示します。

```
SELECT * FROM rms.rmsdb.table1
```

Iceberg REST Glue カタログ統合の Spark AWS セッション設定

このサンプルでは、Iceberg REST を次の と統合する方法を示します AWS Glue クローラー。

```
spark-sql \
  --conf spark.sql.catalog.rms = org.apache.iceberg.spark.SparkCatalog \
  --conf spark.sql.catalog.rms.type = rest \
  --conf spark.sql.catalog.rms.warehouse = glue RMS catalog ID \
  --conf spark.sql.catalog.rms.uri = glue endpoint URI/iceberg \
  --conf spark.sql.catalog.rms.rest.sigv4-enabled = true \
  --conf spark.sql.catalog.rms.rest.signing-name = glue \

  --conf spark.sql.extensions=
    org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions
```

サンプルクエリを以下に示します。

```
SELECT * FROM rms.rmsdb.table1
```

この設定は、Redshift マネージドストレージでのみ機能します。Amazon S3 の FGAC はサポートされていません。

PyFlink の使用

Amazon EMR on EKS リリース 6.15.0 以降では、Flink がサポートされています。既に PyFlink スクリプトを保有されている場合は、次のいずれかを実行できます。

- PyFlink スクリプトを配置したカスタムイメージを作成します。
- スクリプトを Amazon S3 の場所にアップロードする

スクリプトをまだ保有されていない場合は、次の例を使用して PyFlink ジョブを起動できます。この例では、S3 からスクリプトを取得します。スクリプトでイメージに既に含まれているカスタムイメージを使用している場合は、スクリプトパスをスクリプトを保存した場所に更新する必要があります。スクリプトが S3 の場所にある場合、Amazon EMR on EKS はスクリプトを取得し、Flink コンテナの `/opt/flink/usrlib/` ディレクトリに配置します。

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
```

```
  name: python-example
spec:
  flinkVersion: v1_17
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "1"
  executionRoleArn: job-execution-role
  emrReleaseLabel: "emr-6.15.0-flink-latest"
  jobManager:
    highAvailabilityEnabled: false
    replicas: 1
    resource:
      memory: "2048m"
      cpu: 1
  taskManager:
    resource:
      memory: "2048m"
      cpu: 1
  job:
    jarURI: s3://S3 bucket with your script/pyflink-script.py
    entryClass: "org.apache.flink.client.python.PythonDriver"
    args: ["-py", "/opt/flink/usrlib/pyflink-script.py"]
    parallelism: 1
    upgradeMode: stateless
```

Flink AWS での Glue の使用

Amazon EMR on EKS with Apache Flink リリース 6.15.0 以降では、ストリーミングおよびバッチ SQL ワークフローのメタデータストアとして AWS Glue データカタログの使用がサポートされています。

まず、Flink SQL Catalog AWS `default`として機能する という名前の Glue データベースを作成する必要があります。この Flink Catalog は、データベース、テーブル、パーション、ビュー、関数、およびその他の外部システムのデータにアクセスするために必要なその他の情報などのメタデータを保存します。

```
aws glue create-database \
    --database-input "{\"Name\":\"default\"}"
```

AWS Glue サポートを有効にするには、`FlinkDeployment`仕様を使用します。この仕様例では、Python スクリプトを使用して、 Glue カタログとやり取りするための Flink SQL AWS ステートメントをすばやく発行します。

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: python-example
spec:
  flinkVersion: v1_17
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "1"
    aws.glue.enabled: "true"
  executionRoleArn: job-execution-role-arn;
  emrReleaseLabel: "emr-6.15.0-flink-latest"
  jobManager:
    highAvailabilityEnabled: false
    replicas: 1
    resource:
      memory: "2048m"
      cpu: 1
  taskManager:
    resource:
      memory: "2048m"
      cpu: 1
  job:
    jarURI: s3://<S3_bucket_with_your_script/pyflink-glue-script.py
    entryClass: "org.apache.flink.client.python.PythonDriver"
    args: ["-py", "/opt/flink/usrlib/pyflink-glue-script.py"]
    parallelism: 1
    upgradeMode: stateless
```

以下に示しているのは、PyFlink スクリプトの具体的な例です。

```
import logging
import sys
from pyflink.datastream import StreamExecutionEnvironment
from pyflink.table import StreamTableEnvironment

def glue_demo():
    env = StreamExecutionEnvironment.get_execution_environment()
    t_env = StreamTableEnvironment.create(stream_execution_environment=env)
    t_env.execute_sql("""
          CREATE CATALOG glue_catalog WITH (
          'type' = 'hive',
          'default-database' = 'default',
          'hive-conf-dir' = '/glue/confs/hive/conf',
```

```
          'hadoop-conf-dir' = '/glue/confs/hadoop/conf'
          )
                      """)
    t_env.execute_sql("""
          USE CATALOG glue_catalog;
                      """)
    t_env.execute_sql("""
          DROP DATABASE IF EXISTS eks_flink_db CASCADE;
                      """)
    t_env.execute_sql("""
          CREATE DATABASE IF NOT EXISTS eks_flink_db WITH ('hive.database.location-
uri'= 's3a://S3-bucket-to-store-metadata/flink/flink-glue-for-hive/warehouse/');
                      """)
    t_env.execute_sql("""
          USE eks_flink_db;
                  """)
    t_env.execute_sql("""
          CREATE TABLE IF NOT EXISTS eksglueorders (
            order_number BIGINT,
            price        DECIMAL(32,2),
            buyer        RO first_name STRING, last_name STRING,
            order_time   TIMESTAMP(3)
          ) WITH (
            'connector' = 'datagen'
          );
                      """)
    t_env.execute_sql("""
          CREATE TABLE IF NOT EXISTS eksdestglueorders (
            order_number BIGINT,
            price        DECIMAL(32,2),
            buyer        ROW first_name STRING, last_name STRING,
            order_time   TIMESTAMP(3)
          ) WITH (
            'connector' = 'filesystem',
            'path' = 's3://S3-bucket-to-store-metadata/flink/flink-glue-for-hive/
warehouse/eksdestglueorders',
            'format' = 'json'
          );
                  """)
    t_env.execute_sql("""
          CREATE TABLE IF NOT EXISTS print_table (
            order_number BIGINT,
            price        DECIMAL(32,2),
            buyer        ROW first_name STRING, last_name STRING,
```

```
            order_time   TIMESTAMP(3)
          ) WITH (
            'connector' = 'print'
          );
                """)
    t_env.execute_sql("""
          EXECUTE STATEMENT SET
          BEGIN
          INSERT INTO eksdestglueorders SELECT * FROM  eksglueorders LIMIT 10;
          INSERT INTO print_table SELECT * FROM eksdestglueorders;
          END;
            """)


if __name__ == '__main__':
    logging.basicConfig(stream=sys.stdout, level=logging.INFO, format="%(message)s")
    glue_demo()
```

Apache Flink での Apache Hudi の使用

Apache Hudi は、挿入、更新、アップサート、削除などのレコードレベルのオペレーションを備えたオープンソースのデータ管理フレームワークであり、データ管理とデータパイプライン開発を簡素化するために使用できます。Amazon S3 の効率的なデータ管理と組み合わせて Hudi を使用すると、リアルタイムでデータを取り込んで更新できます。Hudi は、データセットで実行したすべてのオペレーションのメタデータを保持するため、すべてのアクションはアトミックで一貫性があります。

Apache Hudi は、Amazon EMR リリース 7.2.0 以降の Apache Flink を搭載した Amazon EMR on EKS で使用できます。Apache Hudi ジョブを開始して送信する方法については、次の手順を参照してください。

Apache Hudi ジョブを送信する

Apache Hudi ジョブを送信する方法については、次の手順を参照してください。

1. という名前 AWS の Glue データベースを作成しますdefault。

   ```
   aws glue create-database --database-input "{\"Name\":\"default\"}"
   ```

2. Flink Kubernetes Operator SQL の例に従って、flink-sql-runner.jar ファイルを構築します。
3. 次のような Hudi SQL スクリプトを作成します。

```
CREATE CATALOG hudi_glue_catalog WITH (
'type' = 'hudi',
'mode' = 'hms',
'table.external' = 'true',
'default-database' = 'default',
'hive.conf.dir' = '/glue/confs/hive/conf/',
'catalog.path' = 's3://<hudi-example-bucket>/FLINK_HUDI/warehouse/'
);

USE CATALOG hudi_glue_catalog;
CREATE DATABASE IF NOT EXISTS hudi_db;
use hudi_db;

CREATE TABLE IF NOT EXISTS hudi-flink-example-table(
    uuid VARCHAR(20),
    name VARCHAR(10),
    age INT,
    ts TIMESTAMP(3),
    `partition` VARCHAR(20)
)
PARTITIONED BY (`partition`)
WITH (
  'connector' = 'hudi',
  'path' = 's3://<hudi-example-bucket>/hudi-flink-example-table',
  'hive_sync.enable' = 'true',
  'hive_sync.mode' = 'glue',
  'hive_sync.table' = 'hudi-flink-example-table',
  'hive_sync.db' = 'hudi_db',
  'compaction.delta_commits' = '1',
  'hive_sync.partition_fields' = 'partition',
  'hive_sync.partition_extractor_class' =
 'org.apache.hudi.hive.MultiPartKeysValueExtractor',
  'table.type' = 'COPY_ON_WRITE'
);

EXECUTE STATEMENT SET
BEGIN

INSERT INTO hudi-flink-example-table VALUES
    ('id1','Alex',23,TIMESTAMP '1970-01-01 00:00:01','par1'),
    ('id2','Stephen',33,TIMESTAMP '1970-01-01 00:00:02','par1'),
    ('id3','Julian',53,TIMESTAMP '1970-01-01 00:00:03','par2'),
    ('id4','Fabian',31,TIMESTAMP '1970-01-01 00:00:04','par2'),
```

```
    ('id5','Sophia',18,TIMESTAMP '1970-01-01 00:00:05','par3'),
    ('id6','Emma',20,TIMESTAMP '1970-01-01 00:00:06','par3'),
    ('id7','Bob',44,TIMESTAMP '1970-01-01 00:00:07','par4'),
    ('id8','Han',56,TIMESTAMP '1970-01-01 00:00:08','par4');

END;
```

4. Hudi SQL スクリプトと flink-sql-runner.jar ファイルを S3 の場所にアップロードします。
5. FlinkDeployments YAML ファイルで、hudi.enabled を true に設定します。

```
spec:
  flinkConfiguration:
    hudi.enabled: "true"
```

6. YAML ファイルを作成して設定を実行します。この例では、ファイル名は hudi-write.yaml です。

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: hudi-write-example
spec:
  flinkVersion: v1_18
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "2"
    hudi.enabled: "true"
  executionRoleArn: "<JobExecutionRole>"
  emrReleaseLabel: "emr-7.7.0-flink-latest"
  jobManager:
    highAvailabilityEnabled: false
    replicas: 1
    resource:
      memory: "2048m"
      cpu: 1
  taskManager:
    resource:
      memory: "2048m"
      cpu: 1
  job:
    jarURI: local:///opt/flink/usrlib/flink-sql-runner.jar
    args: ["/opt/flink/scripts/hudi-write.sql"]
    parallelism: 1
```

```
    upgradeMode: stateless
  podTemplate:
    spec:
      initContainers:
        - name: flink-sql-script-download
          args:
            - s3
            - cp
            - s3://<s3_location>/hudi-write.sql
            - /flink-scripts
          image: amazon/aws-cli:latest
          imagePullPolicy: Always
          resources: {}
          terminationMessagePath: /dev/termination-log
          terminationMessagePolicy: File
          volumeMounts:
            - mountPath: /flink-scripts
              name: flink-scripts
        - name: flink-sql-runner-download
          args:
            - s3
            - cp
            - s3://<s3_location>/flink-sql-runner.jar
            - /flink-artifacts
          image: amazon/aws-cli:latest
          imagePullPolicy: Always
          resources: {}
          terminationMessagePath: /dev/termination-log
          terminationMessagePolicy: File
          volumeMounts:
            - mountPath: /flink-artifacts
              name: flink-artifact
      containers:
        - name: flink-main-container
          volumeMounts:
            - mountPath: /opt/flink/scripts
              name: flink-scripts
            - mountPath: /opt/flink/usrlib
              name: flink-artifact
      volumes:
        - emptyDir: {}
          name: flink-scripts
        - emptyDir: {}
```

```
            name: flink-artifact
```

7. Flink Hudi ジョブを Flink Kubernetes オペレータに送信します。

```
kubectl apply -f hudi-write.yaml
```

Amazon EMR on EKS での RAPIDS Accelerator for Apache Spark の使用

Amazon EMR on EKS では、Nvidia RAPIDS Accelerator for Apache Spark のジョブを実行できます。このチュートリアルでは、EC2 グラフィック処理ユニット (GPU) インスタンスタイプで RAPIDS を使用して Spark ジョブを実行する方法について説明します。このチュートリアルでは、次のバージョンを使用します。

- Amazon EMR on EKS リリースバージョン 6.9.0 以降
- Apache Spark 3.x

Nvidia RAPIDS Accelerator for Apache Spark プラグインを使用すると、Amazon EC2 GPU インスタンスタイプで Spark を高速化できます。これらのテクノロジーを併用すると、コードを変更しなくてもデータサイエンスパイプラインを高速化できます。これにより、データ処理とモデルトレーニングに必要な実行時間が短縮されます。短時間で多くのことができるようになるので、インフラストラクチャのコストを抑えることができます。

開始する前に、次のリソースが使用可能であることを確認してください。

- Amazon EMR on EKS 仮想クラスター
- GPU 対応のノードグループを備えた Amazon EKS クラスター

Amazon EKS 仮想クラスターは、Amazon EKS クラスター上の Kubernetes 名前空間への登録ハンドルであり、Amazon EMR on EKS によって管理されます。このハンドルにより、Amazon EMR は Kubernetes 名前空間をジョブの実行先として使用できます。仮想クラスターをセットアップする方法の詳細については、このガイドの「Amazon EMR on EKS のセットアップ」を参照してください。

ノードグループに GPU インスタンスを含めて、Amazon EKS 仮想クラスターを設定する必要があります。ノードには、Nvidia デバイスプラグインを設定する必要があります。詳細については、「Managed node groups」を参照してください。

GPU 対応のノードグループを追加するように Amazon EKS クラスターを設定するには、次の手順を実行します。

GPU 対応のノードグループを追加するには

1. 次の create-nodegroup コマンドで GPU 対応のノードグループを作成します。Amazon EKS クラスターに合わせて適切なパラメータに置き換えてください。インスタンスタイプは、P4、P3、G5、G4dn など Spark RAPIDS をサポートするものを使用します。

```
aws eks create-nodegroup \
 --cluster-name EKS_CLUSTER_NAME \
 --nodegroup-name NODEGROUP_NAME \
 --scaling-config minSize=0,maxSize=5,desiredSize=2 CHOOSE_APPROPRIATELY \
 --ami-type AL2_x86_64_GPU \
 --node-role NODE_ROLE \
 --subnets SUBNETS_SPACE_DELIMITED  \
 --remote-access ec2SshKey= SSH_KEY \
 --instance-types GPU_INSTANCE_TYPE \
 --disk-size DISK_SIZE \
 --region AWS_REGION
```

2. クラスターに Nvidia デバイスプラグインをインストールすると、クラスターの各ノードに多数の GPU を出力し、クラスターで GPU 対応のコンテナを実行できます。次のコードを実行して、このプラグインをインストールします。

```
kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/v0.9.0/
nvidia-device-plugin.yml
```

3. クラスターの各ノードで使用可能な GPU の数を確認するには、次のコマンドを実行します。

```
kubectl get nodes  "-o=custom-
columns=NAME:.metadata.name,GPU:.status.allocatable.nvidia\.com/gpu"
```

Spark RAPIDS ジョブを実行するには

1. Amazon EMR on EKS クラスターに Spark RAPIDS ジョブを送信します。ジョブを開始するには、次のコードに示したようなコマンドを使用します。ジョブを初めて実行する場合、イメージをダウンロードしてノードにキャッシュするまでに数分かかる場合があります。

```
aws emr-containers start-job-run \
--virtual-cluster-id VIRTUAL_CLUSTER_ID \
--execution-role-arn JOB_EXECUTION_ROLE \
--release-label emr-6.9.0-spark-rapids-latest \
--job-driver '{"sparkSubmitJobDriver": {"entryPoint": "local:///usr/lib/
spark/examples/jars/spark-examples.jar","entryPointArguments":  ["10000"],
 "sparkSubmitParameters":"--class org.apache.spark.examples.SparkPi "}}' \
---configuration-overrides '{"applicationConfiguration": [{"classification":
 "spark-defaults","properties": {"spark.executor.instances":
 "2","spark.executor.memory": "2G"}}],"monitoringConfiguration":
 {"cloudWatchMonitoringConfiguration": {"logGroupName": "LOG_GROUP
 _NAME"},"s3MonitoringConfiguration": {"logUri": "LOG_GROUP_STREAM"}}}'
```

2. Spark RAPIDS Accelerator が使用可能であることを確認するには、Spark ドライバーログをチェックします。このログは、CloudWatch か、または `start-job-run` コマンドを実行するときに指定した S3 の場所に保存されています。次の例は、一般的にログの各行がどのように表示されるかを示しています。

```
22/11/15 00:12:44 INFO RapidsPluginUtils: RAPIDS Accelerator build:
 {version=22.08.0-amzn-0, user=release, url=, date=2022-11-03T03:32:45Z, revision=,
 cudf_version=22.08.0, branch=}
22/11/15 00:12:44 INFO RapidsPluginUtils: RAPIDS Accelerator JNI build:
 {version=22.08.0, user=, url=https://github.com/NVIDIA/spark-rapids-jni.git,
 date=2022-08-18T04:14:34Z, revision=a1b23cd_sample, branch=HEAD}
22/11/15 00:12:44 INFO RapidsPluginUtils: cudf build: {version=22.08.0,
 user=, url=https://github.com/rapidsai/cudf.git, date=2022-08-18T04:14:34Z,
 revision=a1b23ce_sample, branch=HEAD}
22/11/15 00:12:44 WARN RapidsPluginUtils: RAPIDS Accelerator 22.08.0-amzn-0 using
 cudf 22.08.0.
22/11/15 00:12:44 WARN RapidsPluginUtils:
 spark.rapids.sql.multiThreadedRead.numThreads is set to 20.
22/11/15 00:12:44 WARN RapidsPluginUtils: RAPIDS Accelerator is enabled, to disable
 GPU support set `spark.rapids.sql.enabled` to false.
22/11/15 00:12:44 WARN RapidsPluginUtils: spark.rapids.sql.explain is set to
 `NOT_ON_GPU`. Set it to 'NONE' to suppress the diagnostics logging about the query
 placement on the GPU.
```

3. GPU で実行される操作を確認するには、次の手順を実行して、他の口グも記録されるようにします。「spark.rapids.sql.explain : ALL」の設定をメモしておきます。

```
aws emr-containers start-job-run \
--virtual-cluster-id VIRTUAL_CLUSTER_ID \
--execution-role-arn JOB_EXECUTION_ROLE \
--release-label emr-6.9.0-spark-rapids-latest \
--job-driver '{"sparkSubmitJobDriver": {"entryPoint": "local:///usr/lib/
spark/examples/jars/spark-examples.jar","entryPointArguments":  ["10000"],
 "sparkSubmitParameters":"--class org.apache.spark.examples.SparkPi "}}' \
---configuration-overrides '{"applicationConfiguration":
 [{"classification": "spark-defaults","properties":
 {"spark.rapids.sql.explain":"ALL","spark.executor.instances":
 "2","spark.executor.memory": "2G"}}],"monitoringConfiguration":
 {"cloudWatchMonitoringConfiguration": {"logGroupName":
 "LOG_GROUP_NAME"},"s3MonitoringConfiguration": {"logUri": "LOG_GROUP_STREAM"}}}'
```

前掲のコマンドは、GPU を使用するジョブの一例です。その出力は、次の例のようになります。以下を参考に出力内容を理解してください。

- * — GPU で動作する操作を示します。
- ! — GPU で実行できない操作を示します。
- @ — GPU で動作するものの、GPU で実行できないプラン内にあるため実行できない操作を示します。

```
22/11/15 01:22:58 INFO GpuOverrides: Plan conversion to the GPU took 118.64 ms
22/11/15 01:22:58 INFO GpuOverrides: Plan conversion to the GPU took 4.20 ms
22/11/15 01:22:58 INFO GpuOverrides: GPU plan transition optimization took 8.37 ms
22/11/15 01:22:59 WARN GpuOverrides:
   *Exec <ProjectExec> will run on GPU
     *Expression <Alias> substring(cast(date#149 as string), 0, 7) AS month#310
will run on GPU
       *Expression <Substring> substring(cast(date#149 as string), 0, 7) will run
on GPU
         *Expression <Cast> cast(date#149 as string) will run on GPU
     *Exec <SortExec> will run on GPU
       *Expression <SortOrder> date#149 ASC NULLS FIRST will run on GPU
       *Exec <ShuffleExchangeExec> will run on GPU
         *Partitioning <RangePartitioning> will run on GPU
```

```
            *Expression <SortOrder> date#149 ASC NULLS FIRST will run on GPU
          *Exec <UnionExec> will run on GPU
            !Exec <ProjectExec> cannot run on GPU because not all expressions can
 be replaced
              @Expression <AttributeReference> customerID#0 could run on GPU
              @Expression <Alias> Charge AS kind#126 could run on GPU
                @Expression <Literal> Charge could run on GPU
              @Expression <AttributeReference> value#129 could run on GPU
              @Expression <Alias> add_months(2022-11-15, cast(-(cast(_we0#142 as
 bigint) + last_month#128L) as int)) AS date#149 could run on GPU
                ! <AddMonths> add_months(2022-11-15, cast(-
(cast(_we0#142 as bigint) + last_month#128L) as int)) cannot run
 on GPU because GPU does not currently support the operator class
 org.apache.spark.sql.catalyst.expressions.AddMonths
                  @Expression <Literal> 2022-11-15 could run on GPU
                  @Expression <Cast> cast(-(cast(_we0#142 as bigint) +
 last_month#128L) as int) could run on GPU
                    @Expression <UnaryMinus> -(cast(_we0#142 as bigint) +
 last_month#128L) could run on GPU
                      @Expression <Add> (cast(_we0#142 as bigint) +
 last_month#128L) could run on GPU
                        @Expression <Cast> cast(_we0#142 as bigint) could run on
 GPU
                          @Expression <AttributeReference> _we0#142 could run on
 GPU
                        @Expression <AttributeReference> last_month#128L could run
 on GPU
```

Amazon EMR on EKS での Amazon Redshift integration for Apache Spark の使用

Amazon EMR リリース 6.9.0 以降、リリースイメージには常に Apache Spark と Amazon Redshift をつなぐコネクタが含まれます。そのため、Amazon EMR on EKS で Spark を使用すると、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 on EKS では、Amazon Redshift integration for Apache Spark がネイティブインテグレーションとして含まれています。

トピック

- Amazon Redshift integration for Apache Spark 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

Amazon Redshift integration for Apache Spark を使用した Spark アプリケーションの起動

このインテグレーションを使用するには、Spark ジョブで必要な Spark Redshift の依存関係を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る必要があります。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 on EKS リリース 6.9.0 以降で Amazon Redshift integration for Apache Spark を使用して Spark アプリケーションを起動するには、次のようなコマンドを使用します。なお、--conf spark.jars オプションに指定されているパスは JAR ファイルのデフォルトのパスであることに注意してください。

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

--virtual-cluster-id cluster_id \
--execution-role-arn arn \
--release-label emr-6.9.0-latest\
--job-driver '{
    "sparkSubmitJobDriver": {
        "entryPoint": "s3://script_path",
            "sparkSubmitParameters":
            "--conf spark.kubernetes.file.upload.path=s3://upload_path
             --conf spark.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"
                            }
```

```
            }'
```

Amazon Redshift integration for Apache Spark による認証

以下の各セクションでは、Apache Spark と統合する場合の Amazon Redshift での認証オプションを示します。これらのセクションでは、ログイン認証情報を取得する方法と、IAM 認証での JDBC ドライバーの使用に関する詳細を示します。

AWS Secrets Manager を使用して認証情報を取得し、Amazon Redshift に接続する

Secrets Manager に認証情報を保存すると、Amazon Redshift に対して安全に認証できます。Spark ジョブを使用すると、`GetSecretValue` API を呼び出して認証情報を取得できます。

```
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
```

Amazon EMR on EKS ジョブ実行ロールでの IAM ベースの認証の使用

Amazon EMR on EKS リリース 6.9.0 以降、Amazon Redshift JDBC ドライバーバージョン 2.1 以降が環境にパッケージ化されます。JDBC ドライバー 2.1 以降では、JDBC URL を指定できます。未加工のユーザー名とパスワードを含めることはできません。代わりに、`jdbc:redshift:iam://` スキームを指定できます。このコマンドは、Amazon EMR on EKS 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ています。

詳細については、「Amazon Redshift 管理ガイド」の「Configure a JDBC or ODBC connection to use IAM credentials」を参照してください。

次の URL 例では、jdbc:redshift:iam:// スキームを使用しています。

```
jdbc:redshift:iam://examplecluster.abc123xyz789.us-west-2.redshift.amazonaws.com:5439/
dev
```

指定された条件をジョブ実行ロールが満たすためには、次の権限が必要です。

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
redshift:DescribeCluster	JDBC URL に Amazon Redshift クラスターのほか、エンドポイントではなく AWS リージョン を指定する場合に必要
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
redshift-serverless:GetWorkgroup	Amazon Redshift Serverless を使用していて、URL にワークグループ名とリージョンを含めて指定する場合に必要

ジョブ実行ロールポリシーには、次の権限が必要です。

```
{
            "Effect": "Allow",
            "Action": [
                "redshift:GetClusterCredentials",
                "redshift:DescribeCluster",
                "redshift-serverless:GetCredentials",
                "redshift-serverless:GetWorkgroup"
            ],
            "Resource": [

 "arn:aws:redshift:AWS_REGION:ACCOUNT_ID:dbname:CLUSTER_NAME/DATABASE_NAME",
                "arn:aws:redshift:AWS_REGION:ACCOUNT_ID:dbuser:DATABASE_NAME/USER_NAME"
            ]
        }
```

JDBC ドライバーによる Amazon Redshift の認証

JDBC URL 内にユーザー名とパスワードを設定する

Amazon Redshift クラスターに対して Spark ジョブを認証する場合は、JDBC URL に Amazon Redshift データベース名とパスワードを指定できます。

> **Note**
>
> URL にデータベース認証情報を渡すと、その URL にアクセスできるユーザーなら誰でもその認証情報にアクセスできます。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。

ご使用のアプリケーションでセキュリティが問題にならない場合は、JDBC URL に次の形式を使用してユーザー名とパスワードを設定できます。

```
jdbc:redshift://redshifthost:5439/database?user=username&password=password
```

Amazon Redshift に対する読み書き

次のコード例では、PySpark でデータソース API と SparkSQL を使用して、Amazon Redshift データベースに対してサンプルデータを読み書きします。

Data source API

PySpark でデータソース API を使用して、Amazon Redshift データベースに対してサンプルデータを読み書きします。

```
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::accountID:role/roleName"

df = sql_context.read \
    .format("io.github.spark_redshift_community.spark.redshift") \
```

```
    .option("url", url) \
    .option("dbtable", "tableName") \
    .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", "tableName_copy") \
    .option("tempdir", "s3://path/for/temp/data") \
    .option("aws_iam_role", "aws_iam_role_arn") \
    .mode("error") \
    .save()
```

SparkSQL

PySpark で SparkSQL を使用して、Amazon Redshift データベースに対してサンプルデータを読み書きします。

```
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::accountID:role/roleName"

bucket = "s3://path/for/temp/data"
tableName = "tableName" # Redshift table name

s = f"""CREATE TABLE IF NOT EXISTS {tableName} (country string, data string)
    USING io.github.spark_redshift_community.spark.redshift
    OPTIONS (dbtable '{tableName}', 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(tableName, overwrite=False)
df = spark.sql(f"SELECT * FROM {tableName}")
df.show()
```

Spark コネクタを使用する際の考慮事項と制限事項

Spark コネクタは、認証情報の管理、セキュリティの設定、他の AWS サービスとの接続を行うさまざまな方法をサポートしています。機能的で回復力のある接続を設定するには、このリストの推奨事項について習熟してください。

- Amazon EMR 上の Spark から Amazon Redshift への JDBC 接続に対して SSL をアクティブ化することをお勧めします。
- ベストプラクティスとして、 AWS Secrets Manager 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。例については[AWS Secrets Manager 、「 を使用して Amazon Redshift に接続するための認証情報を取得する]」を参照してください。
- Amazon Redshift 認証パラメータのパラメータ `aws_iam_role` 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。
- `tempdir` URI は Amazon S3 の場所を指します。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 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、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 EMR は、常にオープンソースからイメージにインポートされるコードを検証します。セキュリティのため、Spark から Amazon S3 `tempdir` への認証方法として URI の AWS アクセスキーのエンコードはサポートされていません。

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 on EKS で Apache Spark のカスタムスケジューラとして Volcano を使用する方法

Amazon EMR on EKS では、Spark オペレータまたは spark-submit を使用して、Kubernetes カスタムスケジューラで Spark ジョブを実行できます。このチュートリアルでは、カスタムキューで Volcano スケジューラを使用して、Spark ジョブを実行する方法について説明します。

概要

Volcano では、キュースケジューリング、フェアシェアスケジューリング、リソース予約などの高度な機能を利用して、Spark スケジューリングを管理できます。Volcano の利点の詳細については、The Linux Foundation の CNCF ブログの「Why Spark chooses Volcano as built-in batch scheduler on Kubernetes」を参照してください。

Volcano のインストールとセットアップ

1. アーキテクチャのニーズに応じて、次のいずれかの kubectl コマンドを選択して、Volcano をインストールします。

```
# x86_64
kubectl apply -f https://raw.githubusercontent.com/volcano-sh/volcano/v1.5.1/
installer/volcano-development.yaml
# arm64:
kubectl apply -f https://raw.githubusercontent.com/volcano-sh/volcano/v1.5.1/
installer/volcano-development-arm64.yaml
```

2. サンプルの Volcano キューを用意します。キューは、PodGroups の集まりです。FIFO を採用しており、リソース配分の基礎となるものです。

```
cat << EOF > volcanoQ.yaml
apiVersion: scheduling.volcano.sh/v1beta1
kind: Queue
metadata:
  name: sparkqueue
spec:
  weight: 4
  reclaimable: false
  capability:
    cpu: 10
    memory: 20Gi
EOF

kubectl apply -f volcanoQ.yaml
```

3. サンプルの PodGroup マニフェストを Amazon S3 にアップロードします。PodGroup は、関連性の強いポッドをグループ化したものです。通常、バッチスケジューリングに PodGroup を使用します。前のステップで定義したキューに次のサンプルの PodGroup を送信します。

```
cat << EOF > podGroup.yaml
apiVersion: scheduling.volcano.sh/v1beta1
kind: PodGroup
spec:
  # Set minMember to 1 to make a driver pod
  minMember: 1
  # Specify minResources to support resource reservation.
  # Consider the driver pod resource and executors pod resource.
  # The available resources should meet the minimum requirements of the Spark job
  # to avoid a situation where drivers are scheduled, but they can't schedule
  # sufficient executors to progress.
  minResources:
    cpu: "1"
    memory: "1Gi"
  # Specify the queue. This defines the resource queue that the job should be
 submitted to.
  queue: sparkqueue
EOF

aws s3 mv podGroup.yaml s3://bucket-name
```

Spark オペレータを指定して Volcano スケジューラで Spark アプリケーションを実行する

1. 次のセクションのステップをまだ完了していない場合は完了してセットアップします。

   a. Volcano のインストールとセットアップ

   b. Amazon EMR on EKS での Spark 演算子のセットアップ

   c. Spark 演算子をインストールする

   `helm install spark-operator-demo` コマンドを実行するときは、次の引数を含めます。

   ```
   --set batchScheduler.enable=true
   --set webhook.enable=true
   ```

2. `batchScheduler` を設定して SparkApplication 定義ファイル `spark-pi.yaml` を作成します。

```
apiVersion: "sparkoperator.k8s.io/v1beta2"
kind: SparkApplication
metadata:
  name: spark-pi
  namespace: spark-operator
spec:
  type: Scala
  mode: cluster
  image: "895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-6.10.0:latest"
  imagePullPolicy: Always
  mainClass: org.apache.spark.examples.SparkPi
  mainApplicationFile: "local:///usr/lib/spark/examples/jars/spark-examples.jar"
  sparkVersion: "3.3.1"
  batchScheduler: "volcano"   #Note: You must specify the batch scheduler name as
 'volcano'
  restartPolicy:
    type: Never
  volumes:
    - name: "test-volume"
      hostPath:
        path: "/tmp"
        type: Directory
  driver:
```

```
    cores: 1
    coreLimit: "1200m"
    memory: "512m"
    labels:
      version: 3.3.1
    serviceAccount: emr-containers-sa-spark
    volumeMounts:
      - name: "test-volume"
        mountPath: "/tmp"
  executor:
    cores: 1
    instances: 1
    memory: "512m"
    labels:
      version: 3.3.1
    volumeMounts:
      - name: "test-volume"
        mountPath: "/tmp"
```

3. 次のコマンドで Spark アプリケーションを送信します。その結果、spark-pi という名前の SparkApplication オブジェクトも作成されます。

```
kubectl apply -f spark-pi.yaml
```

4. 次のコマンドで SparkApplication オブジェクトのイベントがないか確認します。

```
kubectl describe pods spark-pi-driver --namespace spark-operator
```

最初のポッドイベントは、Volcano がポッドをスケジュールしたことを示しています。

```
Type    Reason     Age   From               Message
----    ------     ----  ----               -------
Normal  Scheduled  23s   volcano            Successfully assigned default/spark-
pi-driver to integration-worker2
```

spark-submit を指定して Volcano スケジューラで Spark アプリケーションを実行します。

1. まず、「Amazon EMR on EKS での spark-submit のセットアップ」セクションのステップを完了します。Volcano に対応した spark-submit ディストリビューションをビルドする必要

があります。詳細については、Apache Spark ドキュメントの「Using Volcano as Customized Scheduler for Spark on Kubernetes」の「Build」セクションを参照してください。

2. 次の環境変数の値を設定します。

```
export SPARK_HOME=spark-home
export MASTER_URL=k8s://Amazon-EKS-cluster-endpoint
```

3. 次のコマンドで Spark アプリケーションを送信します。

```
$SPARK_HOME/bin/spark-submit \
 --class org.apache.spark.examples.SparkPi \
 --master $MASTER_URL \
 --conf spark.kubernetes.container.image=895885662937.dkr.ecr.us-
west-2.amazonaws.com/spark/emr-6.10.0:latest \
 --conf spark.kubernetes.authenticate.driver.serviceAccountName=spark \
 --deploy-mode cluster \
 --conf spark.kubernetes.namespace=spark-operator \
 --conf spark.kubernetes.scheduler.name=volcano \
 --conf spark.kubernetes.scheduler.volcano.podGroupTemplateFile=/path/to/podgroup-
template.yaml \
 --conf
 spark.kubernetes.driver.pod.featureSteps=org.apache.spark.deploy.k8s.features.VolcanoFeatu
 \
 --conf
 spark.kubernetes.executor.pod.featureSteps=org.apache.spark.deploy.k8s.features.VolcanoFea
 \
 local:///usr/lib/spark/examples/jars/spark-examples.jar 20
```

4. 次のコマンドで SparkApplication オブジェクトのイベントがないか確認します。

```
kubectl describe pod spark-pi --namespace spark-operator
```

最初のポッドイベントは、Volcano がポッドをスケジュールしたことを示しています。

```
Type    Reason     Age   From               Message
----    ------     ----  ----               -------
Normal  Scheduled  23s   volcano            Successfully assigned default/spark-
pi-driver to integration-worker2
```

Amazon EMR on EKS で Apache Spark のカスタムスケジューラとして YuniKorn を使用する方法

Amazon EMR on EKS では、Spark オペレータまたは spark-submit を使用して、Kubernetes カスタムスケジューラで Spark ジョブを実行できます。このチュートリアルでは、カスタムキューとギャングスケジューリングで YuniKorn スケジューラを使用して、Spark ジョブを実行する方法について説明します。

概要

Apache YuniKorn では、アプリ対応のスケジューリングで Spark スケジューリングを管理して、リソースのクォータと優先順位をきめ細かく制御できます。ギャングスケジューリングを備えており、アプリに最小限必要なリソースリクエストを満たすことができるときにのみ、アプリをスケジュールします。詳細については、Apache YuniKorn ドキュメントサイトの「What is gang scheduling」を参照してください。

YuniKorn 用にクラスターを作成してセットアップする

次の手順を使用して、Amazon EKS クラスターをデプロイします。 AWS リージョン (`region`) とアベイラビリティーゾーン (`availabilityZones`) は変更してもかまいません。

1. Amazon EKS クラスターを定義します。

```
cat <<EOF >eks-cluster.yaml
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: emr-eks-cluster
  region: eu-west-1

vpc:
  clusterEndpoints:
    publicAccess: true
    privateAccess: true

iam:
  withOIDC: true
```

```
nodeGroups:
  - name: spark-jobs
    labels: { app: spark }
    instanceType: m5.xlarge
    desiredCapacity: 2
    minSize: 2
    maxSize: 3
    availabilityZones: ["eu-west-1a"]
EOF
```

2. クラスターを作成します。

```
eksctl create cluster -f eks-cluster.yaml
```

3. Spark ジョブの実行先となる名前空間 spark-job を作成します。

```
kubectl create namespace spark-job
```

4. 次に、Kubernetes ロールとロールバインディングを作成します。これは、Spark ジョブ実行が使用するサービスアカウントに必須の操作です。

   a. Spark ジョブのサービスアカウント、ロール、ロールバインディングを定義します。

```
cat <<EOF >emr-job-execution-rbac.yaml
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: spark-sa
  namespace: spark-job
automountServiceAccountToken: false
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: spark-role
  namespace: spark-job
rules:
  - apiGroups: ["", "batch","extensions"]
    resources: ["configmaps","serviceaccounts","events","pods","pods/
exec","pods/log","pods/
portforward","secrets","services","persistentvolumeclaims"]
    verbs: ["create","delete","get","list","patch","update","watch"]
```

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: spark-sa-rb
  namespace: spark-job
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: spark-role
subjects:
  - kind: ServiceAccount
    name: spark-sa
    namespace: spark-job
EOF
```

b. 次のコマンドで Kubernetes ロールとロールバインディング定義を適用します。

```
kubectl apply -f emr-job-execution-rbac.yaml
```

YuniKorn をインストールしてセットアップする

1. 次の kubectl コマンドを使用して、Yunikorn スケジューラのデプロイ先となる名前空間 yunikorn を作成します。

```
kubectl create namespace yunikorn
```

2. このスケジューラをインストールするには、次の Helm コマンドを実行します。

```
helm repo add yunikorn https://apache.github.io/yunikorn-release
```

```
helm repo update
```

```
helm install yunikorn yunikorn/yunikorn --namespace yunikorn
```

Spark オペレータを指定して YuniKorn スケジューラで Spark アプリケーションを実行する

1. 次のセクションのステップをまだ完了していない場合は完了してセットアップします。

   a. YuniKorn 用にクラスターを作成してセットアップする

   b. YuniKorn をインストールしてセットアップする

   c. Amazon EMR on EKS での Spark 演算子のセットアップ

   d. Spark 演算子をインストールする

   `helm install spark-operator-demo` コマンドを実行するときは、次の引数を含めます。

   ```
   --set batchScheduler.enable=true
   --set webhook.enable=true
   ```

2. `SparkApplication` ジョブ定義ファイル `spark-pi.yaml` を作成します。

   YuniKorn をジョブのスケジューラとして使用するには、アプリケーション定義に特定の注釈とラベルを追加する必要があります。注釈とラベルでは、使用するジョブのキューとスケジューリング戦略を指定します。

   次の例では、注釈 `schedulingPolicyParameters` はアプリケーションにギャングスケジューリングをセットアップします。次に、タスクグループ (タスクの「ギャング」) を作成して、ジョブ実行を開始するポッドをスケジュールする前に最小限使用できなければならないキャパシティを指定します。最後に、「YuniKorn 用にクラスターを作成してセットアップする」セクションの定義に従って `"app": "spark"` ラベルを指定して、ノードグループを使用するようにタスクグループ定義を指定します。

   ```
   apiVersion: "sparkoperator.k8s.io/v1beta2"
   kind: SparkApplication
   metadata:
     name: spark-pi
     namespace: spark-job
   spec:
     type: Scala
     mode: cluster
     image: "895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-6.10.0:latest"
     imagePullPolicy: Always
   ```

```
  mainClass: org.apache.spark.examples.SparkPi
  mainApplicationFile: "local:///usr/lib/spark/examples/jars/spark-examples.jar"
  sparkVersion: "3.3.1"
  restartPolicy:
    type: Never
  volumes:
    - name: "test-volume"
      hostPath:
        path: "/tmp"
        type: Directory
  driver:
    cores: 1
    coreLimit: "1200m"
    memory: "512m"
    labels:
      version: 3.3.1
    annotations:
      yunikorn.apache.org/schedulingPolicyParameters: "placeholderTimeoutSeconds=30
gangSchedulingStyle=Hard"
      yunikorn.apache.org/task-group-name: "spark-driver"
      yunikorn.apache.org/task-groups: |-
        [{
            "name": "spark-driver",
            "minMember": 1,
            "minResource": {
              "cpu": "1200m",
              "memory": "1Gi"
            },
            "nodeSelector": {
              "app": "spark"
            }
          },
          {
            "name": "spark-executor",
            "minMember": 1,
            "minResource": {
              "cpu": "1200m",
              "memory": "1Gi"
            },
            "nodeSelector": {
              "app": "spark"
            }
        }]
    serviceAccount: spark-sa
```

```
    volumeMounts:
      - name: "test-volume"
        mountPath: "/tmp"
  executor:
    cores: 1
    instances: 1
    memory: "512m"
    labels:
      version: 3.3.1
    annotations:
      yunikorn.apache.org/task-group-name: "spark-executor"
    volumeMounts:
      - name: "test-volume"
        mountPath: "/tmp"
```

3. 次のコマンドで Spark アプリケーションを送信します。その結果、spark-pi という名前の SparkApplication オブジェクトも作成されます。

```
kubectl apply -f spark-pi.yaml
```

4. 次のコマンドで SparkApplication オブジェクトのイベントがないか確認します。

```
kubectl describe sparkapplication spark-pi --namespace spark-job
```

最初のポッドイベントは、YuniKorn がポッドをスケジュールしたことを示しています。

```
Type    Reason            Age   From                          Message
----    ------            ----  ----                          -------
Normal Scheduling        3m12s yunikorn   spark-operator/org-apache-spark-examples-
sparkpi-2a777a88b98b8a95-driver is queued and waiting for allocation
Normal GangScheduling    3m12s yunikorn   Pod belongs to the taskGroup spark-
driver, it will be scheduled as a gang member
Normal Scheduled         3m10s yunikorn   Successfully assigned spark
Normal PodBindSuccessful 3m10s yunikorn   Pod spark-operator/
Normal TaskCompleted     2m3s  yunikorn   Task spark-operator/
Normal Pulling           3m10s kubelet    Pulling
```

`spark-submit` を指定して YuniKorn スケジューラで Spark アプリケーションを実行します。

1. まず、「Amazon EMR on EKS での spark-submit のセットアップ」セクションのステップを完了します。
2. 次の環境変数の値を設定します。

```
export SPARK_HOME=spark-home
export MASTER_URL=k8s://Amazon-EKS-cluster-endpoint
```

3. 次のコマンドで Spark アプリケーションを送信します。

   次の例では、注釈 schedulingPolicyParameters はアプリケーションにギャングスケジューリングをセットアップします。次に、タスクグループ (タスクの「ギャング」) を作成して、ジョブ実行を開始するポッドをスケジュールする前に最小限使用できなければならないキャパシティを指定します。最後に、「YuniKorn 用にクラスターを作成してセットアップする」セクションの定義に従って "app": "spark" ラベルを指定して、ノードグループを使用するようにタスクグループ定義を指定します。

```
$SPARK_HOME/bin/spark-submit \
 --class org.apache.spark.examples.SparkPi \
 --master $MASTER_URL \
 --conf spark.kubernetes.container.image=895885662937.dkr.ecr.us-
west-2.amazonaws.com/spark/emr-6.10.0:latest \
 --conf spark.kubernetes.authenticate.driver.serviceAccountName=spark-sa \
 --deploy-mode cluster \
 --conf spark.kubernetes.namespace=spark-job \
 --conf spark.kubernetes.scheduler.name=yunikorn \
 --conf spark.kubernetes.driver.annotation.yunikorn.apache.org/
schedulingPolicyParameters="placeholderTimeoutSeconds=30 gangSchedulingStyle=Hard"
 \
 --conf spark.kubernetes.driver.annotation.yunikorn.apache.org/task-group-
name="spark-driver" \
 --conf spark.kubernetes.executor.annotation.yunikorn.apache.org/task-group-
name="spark-executor" \
 --conf spark.kubernetes.driver.annotation.yunikorn.apache.org/task-groups='[{
            "name": "spark-driver",
            "minMember": 1,
            "minResource": {
              "cpu": "1200m",
              "memory": "1Gi"
```

```
            },
            "nodeSelector": {
              "app": "spark"
            }
          },
          {
            "name": "spark-executor",
            "minMember": 1,
            "minResource": {
              "cpu": "1200m",
              "memory": "1Gi"
            },
            "nodeSelector": {
              "app": "spark"
            }
        }]' \
 local:///usr/lib/spark/examples/jars/spark-examples.jar 20
```

4. 次のコマンドで SparkApplication オブジェクトのイベントがないか確認します。

```
kubectl describe pod spark-driver-pod --namespace spark-job
```

最初のポッドイベントは、YuniKorn がポッドをスケジュールしたことを示しています。

```
Type    Reason           Age   From                          Message
----    ------           ----  ----                          -------
Normal Scheduling        3m12s yunikorn   spark-operator/org-apache-spark-examples-
sparkpi-2a777a88b98b8a95-driver is queued and waiting for allocation
Normal GangScheduling    3m12s yunikorn   Pod belongs to the taskGroup spark-
driver, it will be scheduled as a gang member
Normal Scheduled         3m10s yunikorn   Successfully assigned spark
Normal PodBindSuccessful 3m10s yunikorn   Pod spark-operator/
Normal TaskCompleted     2m3s  yunikorn   Task spark-operator/
Normal Pulling           3m10s kubelet    Pulling
```

Amazon EMR on EKS のセキュリティ

のクラウドセキュリティが最優先事項 AWS です。 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、 AWS とお客様の間で共有される責任です。責任共有モデルでは、これをクラウドのセキュリティおよびクラウド内のセキュリティと説明しています。

- クラウドのセキュリティ — AWS クラウドで AWS サービスを実行するインフラストラクチャを保護する AWS 責任があります。 AWS また、 は、お客様が安全に使用できるサービスも提供します。AWS コンプライアンスプログラムコンプライアンスプログラムの一環として、サードパーティーの監査者は定期的にセキュリティの有効性をテストおよび検証。Amazon EMR 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、「コンプライアンスプログラムAWS による対象範囲内の のサービスコンプライアンスプログラム」を参照してください。
- クラウド内のセキュリティ – お客様の責任は、使用する AWS サービスによって決まります。また、ユーザーは、データの機密性、会社の要件、適用される法律や規制など、その他の要因についても責任を負います。

このドキュメントは、Amazon EMR on EKS を使用する際の責任共有モデルの適用について理解するのに役立ちます。以下のトピックで、セキュリティおよびコンプライアンスの目的を満たすように、Amazon EMR on EKS を設定する方法について説明します。また、Amazon EMR on EKS リソースのモニタリングや保護に役立つ他の AWS サービスの使用方法についても説明します。

トピック

- 設定と脆弱性の分析
- インターフェイス VPC エンドポイントを使用して Amazon EMR on EKS に接続する
- Amazon EMR on EKS のクロスアカウントアクセスを設定する

Amazon EMR on EKS でのセキュリティのベストプラクティス

Amazon EMR on EKS には、独自のセキュリティポリシーを策定および実装する際に考慮すべき、さまざまな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適切ではないか、十分ではない場合があるため、これらは指示ではなく、有用な考慮事項と見なしてください。

> **Note**
>
> その他のセキュリティのベストプラクティスについては、「Amazon EMR on EKS でのセキュリティのベストプラクティス」を参照してください。

最小特権の原則を適用する

Amazon EMR on EKS は、実行ロールなどの IAM ロールを使用して、アプリケーションに対してきめ細かいアクセスポリシーを提供します。これらの実行ロールは、IAM ロールの信頼ポリシーを使用して Kubernetes サービスアカウントにマッピングされます。Amazon EMR on EKS は、登録された Amazon EKS 名前空間内に、ユーザー提供のアプリケーションコードを実行するポッドを作成します。アプリケーションコードを実行するジョブポッドは、他の AWS サービスに接続するときに実行ロールを引き受けます。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。

エンドポイントのアクセスコントロールリスト

マネージドエンドポイントは、VPC 内で少なくとも 1 つのプライベートサブネットを使用するように設定されている EKS クラスターに対してのみ作成できます。この設定では、VPC からのみアクセスできるように、マネージドエンドポイントによって作成されたロードバランサーへのアクセスが制限されます。セキュリティをさらに強化するには、これらのロードバランサーを使用してセキュリティグループを設定し、選択した一連の IP アドレスに受信トラフィックを制限できるようにすることをお勧めします。

カスタムイメージの最新のセキュリティ更新プログラムを入手する

Amazon EMR on EKS でカスタムイメージを使用するために、イメージに任意のバイナリとライブラリをインストールできます。イメージに追加するバイナリのセキュリティパッチの適用は、お客様が行います。Amazon EMR on EKS のイメージに、最新のセキュリティパッチを定期的に適用します。最新のイメージを取得するために、Amazon EMR リリースの新しいベースイメージバージョンが存在する場合は常に、カスタムイメージを再構築する必要があります。詳細については、Amazon EMR on EKS リリースおよびベースイメージ URI の選択の詳細を参照してください。

ポッドの認証情報アクセスを制限する

Kubernetes は、ポッドに認証情報を割り当てるための方法をいくつかサポートしています。複数の認証情報プロバイダーをプロビジョニングすると、セキュリティモデルの複雑さが増す可能性があります。Amazon EMR on EKS では、サービスアカウントの IAM ロール (IRSA) の使用を、登録された EKS 名前空間内の標準の認証情報プロバイダーとして採用しています。kube2iam、kiam、クラスターで実行されているインスタンスの EC2 インスタンスプロファイルの使用など、他の方法はサポートされていません。

信頼できないアプリケーションコードを分離する

Amazon EMR on EKS では、システムのユーザーが送信したアプリケーションコードの整合性は検査されません。任意のコードを実行する信頼できないテナントがジョブの送信に使用できる、複数の実行ロールで設定されたマルチテナント仮想クラスターを実行している場合、悪意のあるアプリケーションがその特権をエスカレートするリスクがあります。このような状況では、同様の特権を持つ実行ロールを別の仮想クラスター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) の許可

管理者は、Amazon EMR on EKS 管理の名前空間に対するロールベースのアクセスコントロール (RBAC) の許可を厳密に制御する必要があります。少なくとも、Amazon EMR on EKS 管理の名前空間でのジョブ送信者に、次の許可を付与しないでください。

- configmap を変更するための Kubernetes RBAC 許可 - Amazon EMR on EKS は、Kubernetes の configmaps を使用して、マネージドサービスアカウント名を持つマネージドポッドテンプレートを生成するためです。この属性は変更しないでください。
- Amazon EMR on EKS ポッドに対して実行するための Kubernetes RBAC 許可 - マネージド SA 名を持つマネージドポッドテンプレートへのアクセス権を付与しないようにします。この属性は変更しないでください。また、この許可により、ポッドにマウントされた JWT トークンへのアクセス権が付与され、これを使用して実行ロールの認証情報が取得される可能性があります。

- ポッドを作成するための Kubernetes RBAC アクセス許可 - ユーザーが Kubernetes ServiceAccount を使用してポッドを作成できないようにします。Kubernetes ServiceAccount は、ユーザーよりも多くの AWS 権限を持つ IAM ロールにマッピングできます。
- 変更ウェブフックをデプロイするための Kubernetes RBAC 許可 – ユーザーが変更ウェブフックを使用して、Amazon EMR on EKS によって作成されたポッドの Kubernetes ServiceAccount 名を変更できないようにします。
- Kubernetes シークレットを読み取るための Kubernetes RBAC 許可 - ユーザーがこれらのシークレットに保存されている機密データを読み取ることができないようにします。

ノードグループの IAM ロールまたはインスタンスプロファイルの認証情報へのアクセスを制限する

- ノードグループの IAM ロール (複数可) に最小限の AWS アクセス許可を割り当てることをお勧めします。これにより、EKS ワーカーノードのインスタンスプロファイル認証情報を使用して実行される可能性がある、コードによる特権エスカレーションを回避できます。
- Amazon EMR on EKS マネージドの名前空間で実行されるすべてのポッドに対するインスタンスプロファイル認証情報へのアクセスを完全にブロックするには、EKS ノードで `iptables` コマンドを実行することをお勧めします。詳細については、「Amazon EC2 インスタンスプロファイルの認証情報に対するアクセスの制限」を参照してください。ただし、ポッドが必要なすべての許可を持つように、サービスアカウントの IAM ロールを適切にスコープすることが重要です。例えば、ノードの IAM ロールには、Amazon ECR からコンテナイメージを取得するための許可が割り当てられます。ポッドにこれらの許可が割り当てられていない場合、ポッドは Amazon ECR からコンテナイメージを取得できません。VPC CNI プラグインも更新する必要があります。詳細については、「演習: サービスアカウントの IAM ロールを使用するように VPC CNI プラグインを更新する」を参照してください。

データ保護

Amazon EMR on EKS でのデータ保護には、 AWS 責任共有モデルが適用されます。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。ユーザーには、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制御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 のサービスに対するセキュリティの設定と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のよくある質問」を参照してください。欧州でのデータ保護の詳細については、 AWS セキュリティブログのブログ投稿「 責任 AWS 共有モデルと GDPR」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 (IAM) 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。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。TLS 1.2 以降が推奨されます。
- 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、 AWS サービス内のすべてのデフォルトのセキュリティコントロールを使用します。
- 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 on EKS の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 API AWS を介して にアクセスするときに FIPS 140-2 検証済み暗号化モジュールが必要な場合は、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、「連邦情報処理規格 (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、[Name] (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。これは、コンソール、API、 AWS CLIまたは SDK を使用して Amazon EMR on EKS または他の AWS のサービスを使用する場合も同様です。 AWS SDKs Amazon EMR on EKS や他のサービスに入力したすべてのデータは、診断ログに取り込まれる可能性があります。外部サーバーへの URL を指定するときは、そのサーバーへのリクエストを検証するための認証情報を URL 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から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、暗号化キーの保存と使用には料金が適用されます。詳細については、「AWS KMS 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 内に保管中の EMRFS データの暗号化

Amazon S3 暗号化は、Amazon S3 への読み取りおよび書き込みが行われる EMR ファイルシステム (EMRFS) オブジェクトで使用できます。保管中のデータの暗号化を有効にする場合は、デフォルトの暗号化モードとして、Amazon S3 サーバー側での暗号化 (SSE) またはクライアント側での暗号化 (CSE) を指定します。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 の暗号化が有効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 の間で転送される EMRFS オブジェクトを暗号化します。Amazon S3 の暗号化の詳細については、「Amazon Simple Storage Service 開発者ガイド」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、暗号化キーの保存と使用には料金が適用されます。詳細については、「AWS KMS 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。SSE の詳細については、「Amazon Simple Storage Service 開発者ガイド」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 on EKS で SSE を指定するときに、次の 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 がキーを管理します。
- SSE-KMS - を使用して AWS KMS key 、Amazon EMR on EKS に適したポリシーをセットアップします。

SSE とお客様が用意したキーとの組み合わせ (SSE-C) は、Amazon EMR on EKS では使用できません。

Amazon S3 クライアント側の暗号化

Amazon S3 のクライアント側の暗号化を使用すると、Amazon S3 の暗号化と復号はクラスターの 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 によって提供されるキー (CSE-KMS) か、クライアント側のルートキーを提供するカスタム Java クラス (CSE-C) を使用できます。CSE-KMS と CSE-C では、指定するプロバイダーと、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、暗号化の仕様が少し異なります。これらの差異に関する詳細は、「Amazon Simple Storage Service 開発者ガイド」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。

> **Note**
>
> Amazon S3 CSE では、Amazon S3 と交換される EMRFS データのみが暗号化されます。クラスターインスタンスボリュームのすべてのデータが暗号化されるわけではありません。さらに、Hue は EMRFS を使用しないため、Hue S3 ファイルブラウザが Amazon S3 に書き込むオブジェクトは暗号化されません。

ローカルディスク暗号化

Apache Spark は、ローカルディスクに書き込まれた一時データの暗号化をサポートしています。これには、キャッシュ変数とブロードキャスト変数の両方について、ディスクに保存されているシャッフルファイル、シャッフルスピル、およびデータブロックが含まれます。`saveAsHadoopFile` や `saveAsTable` などの API を使用したアプリケーションによって生成された出力データの暗号化は対象外です。また、ユーザーが明示的に作成した一時ファイルは対象にならない場合があります。詳細については、Spark ドキュメントの「Local Storage Encryption」(ローカルストレージ暗号化) を参照してください。Spark は、データがメモリに収まらない場合、エグゼキュータプロセスによってローカルディスクに書き込まれる中間データなど、ローカルディスク上の暗号化されたデータをサポートしません。ディスクに保持されたデータは、ジョブランタイムにスコープ設定され、データの暗号化に使用されるキーは、ジョブ実行ごとに Spark によって動的に生成されます。Spark ジョブが終了すると、他のプロセスはデータを復号できません。

ドライバーとエグゼキューターポッドの場合、マウントされたボリュームに保持される保管中のデータを暗号化します。Kubernetes で使用できる AWS ネイティブストレージには、EBS、EFS、FSx

for Lustre の 3 つの異なるオプションがあります。3 つのオプションはすべて、サービスマネージドキーまたは AWS KMS keyを使用して保管時の暗号化を行います。詳細については、「EKS ベストプラクティスガイド」を参照してください。この方法では、マウントされたボリュームに保持されるすべてのデータが暗号化されます。

キー管理

KMS キーを自動的にローテーションするように KMS 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、「ローテーション AWS KMS keys」を参照してください。

転送中の暗号化

転送時の暗号化では、複数の暗号化メカニズムが有効になります。これらはオープンソース機能であり、アプリケーション固有のもので、Amazon EMR on EKS リリースによって異なる可能性があります。次のアプリケーション固有の暗号化機能を Amazon EMR on EKS で有効にできます。

- Spark
  - Spark コンポーネント間 (ブロック転送サービスと外部シャッフルサービスなど) での内部 RPC 通信は、Amazon EMR のバージョン 5.9.0 以降では AES-256 暗号を使用して暗号化されます。以前のリリースでは、内部 RPC 通信は SASL と、暗号として DIGEST-MD5 を使用して暗号化されます。
  - Spark 履歴サーバーや HTTPS 対応ファイルサーバーなどのユーザーインターフェイスを使用した HTTP プロトコル通信は、Spark の SSL 設定を使用して暗号化されます。詳細については、Spark ドキュメントの「SSL Configuration」を参照してください。

  詳細については、「Spark のセキュリティ設定」を参照してください。
- Amazon S3 バケット IAM ポリシーで aws:SecureTransport 条件を使用して、HTTPS (TLS) を介した暗号化接続のみを許可してください。
- JDBC または ODBC クライアントにストリーミングされるクエリ結果は、TLS を使用して暗号化されます。

Identity and Access Management

AWS Identity and Access Management (IAM) は、管理者が AWS リソースへのアクセスを安全に制御 AWS のサービス するのに役立つ です。IAM 管理者は、誰を認証 (サインイン) し、誰に Amazon

EMR on EKS リソースの使用を承認する (アクセス許可を付与する) かを制御します。IAM は、追加料金なしで AWS のサービス 使用できる です。

トピック

対象者

AWS Identity and Access Management (IAM) の使用方法は、Amazon EMR on EKS で行う作業によって異なります。

サービスユーザー – ジョブを実行するために Amazon EMR on EKS サービスを使用する場合は、管理者から必要なアクセス許可と認証情報が与えられます。さらに多くの Amazon EMR on EKS 機能を使用して作業を行う場合は、追加のアクセス許可が必要になることがあります。アクセスの管理方法を理解すると、管理者に適切なアクセス許可をリクエストするのに役に立ちます。Amazon EMR on EKS の機能にアクセスできない場合は、「Amazon EMR on EKS の ID とアクセスのトラブルシューティング」を参照してください。

サービス管理者 – 社内の Amazon EMR on EKS リソースを管理しているユーザーには、通常、Amazon EMR on EKS への完全なアクセス権限があります。サービスのユーザーがどの Amazon EMR on EKS 機能やリソースにアクセスするかを決めるのは管理者の仕事です。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権限を変更する必要があります。このページの情報を点検して、IAM の基本概念を理解してください。企業が Amazon EMR on EKS で IAM を利用する方法については、「Amazon EMR on EKS で IAM を使用する方法」を参照してください。

IAM 管理者 – IAM 管理者には、Amazon EMR on EKS へのアクセスを管理するポリシーの作成方法を、詳細に把握する必要が生じる場合があります。IAM で使用可能な、Amazon EMR on EKS アイデンティティベースのポリシーの例を確認するには、「Amazon EMR on EKS の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。

アイデンティティを使用した認証

認証とは、ID 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（ 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 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場合、IAM ロールを使用して、前もって管理者により ID フェデレーションが設定されています。フェデレーションを使用して にアクセスすると、間接的 AWS 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「 にサインインする方法 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 Software Development Kit (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で署名します。 AWS ツールを使用しない場合は、リクエストに自分で署名する必要があります。リクエストに自分で署名する推奨方法の使用については、「IAM ユーザーガイド」の「API リクエストに対するAWS Signature Version 4」を参照してください。

使用する認証方法を問わず、追加セキュリティ情報の提供をリクエストされる場合もあります。例えば、 では、アカウントのセキュリティを強化す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」および「IAM ユーザーガイド」の「IAM のAWS 多要素認証」を参照してください。

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 つのサインインアイデンティティから始めます。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することでアクセスできます。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「ルートユーザー認証情報が必要なタスク」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、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 AWS のサービス を使用して に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を使用することを要求します。

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のユーザー、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 にアクセスするすべてのユーザーです。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、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ことも、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することもできます。IAM Identity Center の詳細については、「AWS IAM Identity Center ユーザーガイド」の「What is IAM Identity Center?」(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 ユーザーは、単一のユーザーまたはアプリケーションに特定のアクセス許可 AWS アカウント を持つ 内のアイデンティティです。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保有する IAM ユーザーを作成する代わりに、一時的な認証情報を使用することをお勧めします。ただし、IAM ユーザーでの長期的な認証情報が必要な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ために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 ユーザーの集団を指定するアイデンティティです。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、IAMAdmins という名前のグループを設定して、そのグループに IAM リソースを管理する許可を与えることができます。

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「IAM ユーザーのユースケース」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 内のアイデンティティです。これは IAM ユーザーに似ていますが、特定のユーザーには関連付けられていません。で IAM ロールを一時的に引き受けるには AWS Management Console、ユーザーから IAM ロールに切り替えることができます (コンソール) 。ロールを引き受けるには、 または AWS API オペレーションを AWS CLI 呼び出すか、カスタム URL を使用します。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ための各種方法」を参照してください。

IAM ロールと一時的な認証情報は、次の状況で役立ちます:

- フェデレーションユーザーアクセス – フェデレーティッド ID に許可を割り当てるには、ロールを作成してそのロールの許可を定義します。フェデレーティッド ID が認証されると、その ID はロールに関連付けられ、ロールで定義されている許可が付与されます。フェデレーションのロールについては、「IAM ユーザーガイド」の「サードパーティー ID プロバイダー (フェデレーション) 用のロールを作成する」を参照してください。IAM Identity Center を使用する場合は、許可セットを設定します。アイデンティティが認証後にアクセスできるものを制御するため、IAM Identity Center は、権限セットを IAM のロールに関連付けます。アクセス許可セットの詳細については、「AWS IAM Identity Center User Guide」の「Permission sets」を参照してください。
- 一時的な IAM ユーザー権限 - IAM ユーザーまたはロールは、特定のタスクに対して複数の異なる権限を一時的に IAM ロールで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、自分のアカウントのリソースにアクセスすることを、別のアカウントの人物 (信頼済みプリンシパル) に許可できます。クロスアカウントアクセス権を付与する主な方法は、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、 (ロールをプロキシとして使用する代わりに) 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における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「IAM でのクロスアカウントのリソースへのアクセス」を参照してください。
- クロスサービスアクセス — 一部の では、他の の機能 AWS のサービス を使用します AWS のサービス。例えば、あるサービスで呼び出しを行うと、通常そのサービスによって Amazon EC2 でアプリケーションが実行されたり、Amazon S3 にオブジェクトが保存されたりします。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リクエストする 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アクセス許可が必要です。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。
- サービスロール - サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除することができます。詳細については、「IAM ユーザーガイド」の「AWS のサービスに許可を委任するロールを作成する」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、サービスリンクロールのアクセス許可を表示できますが、編集することはできません。

- Amazon EC2 で実行されているアプリケーション – 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 リクエストを実行している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 インスタンス内でのアクセスキーの保存に推奨されます。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、EC2 インスタンスで実行されるプログラムは一時的な認証情報を取得できます。詳細については、「IAM ユーザーガイド」の「Amazon EC2 インスタンスで実行されるアプリケーションに IAM ロールを使用して許可を付与する」を参照してください。

ポリシーを使用したアクセスの管理

でアクセスを制御するには AWS 、ポリシーを作成し、ID AWS またはリソースにアタッチします。ポリシーは のオブジェクト AWS であり、アイデンティティまたはリソースに関連付けられると、そのアクセス許可を定義します。 は、プリンシパル (ユーザー、ルートユーザー、またはロールセッション) がリクエストを行うときに、これらのポリシー AWS を評価します。ポリシーでの権限により、リクエストが許可されるか拒否されるかが決まります。ほとんどのポリシーは JSON ドキュ

メント AWS として に保存されます。JSON ポリシードキュメントの構造と内容の詳細については、IAM ユーザーガイドの JSON ポリシー概要を参照してください。

管理者は JSON AWS ポリシーを使用して、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。IAM 管理者は、リソースで必要なアクションを実行するための権限をユーザーに付与する IAM ポリシーを作成できます。その後、管理者はロールに IAM ポリシーを追加し、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方法を問わず、アクションの許可を定義します。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 API からロール情報を取得できます。

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ポリシーは、IAM ユーザーグループ、ユーザーのグループ、ロールなど、アイデンティティにアタッチできる JSON 許可ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件をコントロールします。ID ベースのポリシーを作成する方法については、「IAM ユーザーガイド」の「カスタマー管理ポリシーでカスタム IAM アクセス許可を定義する」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。管理ポリシーは、 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のいずれかを選択する方法については、「IAM ユーザーガイド」の「管理ポリシーとインラインポリシーのいずれかを選択する」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、リソースに添付する 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には例として、IAM ロールの信頼ポリシーや Amazon S3 バケットポリシーがあげられます。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制御できます。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では、IAM の AWS マネージドポリシーを使用できません。

アクセスコントロールリスト (ACL)

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするための許可を持つかを制御します。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

Amazon S3、 AWS WAF、および Amazon VPC は、ACLs。ACL の詳細については、「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) の概要」を参照してください。

その他のポリシータイプ

AWS は、一般的ではない追加のポリシータイプをサポートしています。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ポリシーによって IAM エンティティ (IAM ユーザーまたはロール) に付与できる権限の上限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 AWS アカウント をグループ化して一元管理するためのサービスです。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 (SCP) を一部またはすべてのアカウントに適用できます。SCP 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を制限します AWS アカウントのルートユーザー。Organizations と SCP の詳細については、「AWS Organizations ユーザーガイド」の「サービスコントロールポリシー (SCP)」を参照してください。
- リソースコントロールポリシー (RCP) – RCP は、所有する各リソースにアタッチされた IAM ポリシーを更新することなく、アカウント内のリソースに利用可能な最大数のアクセス許可を設定す

るために使用できる JSON ポリシーです。RCP は、メンバーアカウントのリソースに対するアクセス許可を制限し、組織に属しているかどうかにかかわらず AWS アカウントのルートユーザー、を含む ID に対する有効なアクセス許可に影響を与える可能性があります。RCP をサポートするのリストを含む Organizations と RCP の詳細については、AWS Organizations RCPs「リソースコントロールポリシー (RCPs」を参照してください。 AWS のサービス

- 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、「IAM 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。

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。複数のポリシータイプが関係する場合に が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、「IAM 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。

Amazon EMR on EKS で IAM を使用する方法

IAM を使用して Amazon EMR on EKS へのアクセスを管理する前に、Amazon EMR on EKS で使用できる IAM 機能について理解しておく必要があります。

Amazon EMR on EKS で使用できる IAM の機能

IAM 機能	Amazon EMR on EKS のサポート
アイデンティティベースポリシー	はい
リソースベースのポリシー	いいえ
ポリシーアクション	はい
ポリシーリソース	あり
ポリシー条件キー	Yes
ACL	いいえ

IAM 機能	Amazon EMR on EKS のサポート
ABAC (ポリシー内のタグ)	あり
一時的な認証情報	はい
プリンシパル権限	はい
サービスロール	いいえ
サービスリンクロール	あり

Amazon EMR on EKS およびその他の AWS のサービスがほとんどの IAM 機能と連携する方法の概要を把握するには、「IAM ユーザーガイド」のAWS 「IAM と連携する のサービス」を参照してください。

Amazon EMR on EKS 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

アイデンティティベースポリシーは、IAM ユーザーグループ、ユーザーのグループ、ロールなど、アイデンティティにアタッチできる JSON 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 ベースのポリシーの作成方法については、「IAM ユーザーガイド」の「カスタマー管理ポリシーでカスタム IAM アクセス許可を定義する」を参照してください。

IAM アイデンティティベースのポリシーでは、許可または拒否するアクションとリソース、およびアクションを許可または拒否す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学ぶには、「IAM ユーザーガイド」の「IAM JSON ポリシーの要素のリファレンス」を参照してください。

Amazon EMR on EKS のアイデンティティベースのポリシーの例

Amazon EMR on EKS のアイデンティティベースのポリシーの例は、「Amazon EMR on EKS のアイデンティティベースのポリシーの例」でご確認ください。

Amazon EMR on EKS 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、リソースに添付する 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には例として、IAM ロールの信頼ポリシーや Amazon S3 バケットポリシーがあげられます。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制御できます。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、アカウント全体、または別のアカウントの IAM エンティティをリソースベースのポリシーのプリンシパルとして指定します。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場合 AWS アカウント、信頼されたアカウントの IAM 管理者は、プリンシパルエンティティ (ユーザーまたはロール) にリソースへのアクセス許可も付与する必要があります。IAM 管理者は、アイデンティティベースのポリシーをエンティティにアタッチすることで権限を付与します。ただし、リソースベースのポリシーで、同じアカウントのプリンシパルへのアクセス権が付与されている場合は、アイデンティティベースのポリシーをさらに付与する必要はありません。詳細については、「IAM ユーザーガイド」の「IAM でのクロスアカウントリソースアクセス」を参照してください。

Amazon EMR on EKS 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 JSON AWS ポリシーを使用して、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` 要素に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が記述されます。ポリシーアクションの名前は通常、関連する AWS API オペレーションと同じです。一致する API オペレーションのない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アクセス許可を付与するポリシーで使用されます。

Amazon EMR on EKS のアクションのリストについては、「サービス認可リファレンス」の「Amazon EMR on EKS 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 on EKS 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-containers
```

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-containers:action1",
      "emr-containers:action2"
         ]
```

Amazon EMR on EKS のアイデンティティベースのポリシーの例は、「Amazon EMR on EKS のアイデンティティベースのポリシーの例」でご確認ください。

Amazon EMR on EKS 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 JSON AWS ポリシーを使用して、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、アマゾン 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 on EKS リソースのタイプとその ARN のリストを確認するには、「サービス認可リファレンス」の「Amazon EMR on EKS で定義されるリソース」を参照してください。各リソースの ARN を指定できるアクションについては、「Amazon EMR on EKS 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 on EKS のアイデンティティベースのポリシーの例は、「Amazon EMR on EKS のアイデンティティベースのポリシーの例」でご確認ください。

Amazon EMR on EKS 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: あり

管理者は JSON AWS ポリシーを使用して、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して、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 ユーザーに、IAM ユーザー名がタグ付けされている場合のみリソースにアクセスできる権限を付与することができます。詳細については、「IAM ユーザーガイド」の「IAM ポリシーの要素: 変数およびタグ」を参照してください。

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ています。すべての AWS グローバル条件キーを確認するには、「IAM ユーザーガイド」のAWS 「 グローバル条件コンテキストキー」を参照してください。

Amazon EMR on EKS の条件キーのリスト、および条件キーを使用できるアクションとリソースについては、「サービス認可リファレンス」の「Amazon EMR on EKS 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 on EKS のアイデンティティベースのポリシーの例は、「Amazon EMR on EKS のアイデンティティベースのポリシーの例」でご確認ください。

Amazon EMR on EKS アクセスコントロールリスト (ACL)

ACL のサポート: なし

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするための許可を持つかを制御します。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

Amazon EMR on EKS での属性ベースのアクセスコントロール (ABAC)

ABAC のサポート (ポリシー内のタグ)	はい

属性ベースのアクセス制御 (ABAC) は、属性に基づいてアクセス許可を定義するアクセス許可戦略です。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、ABAC の最初の手順です。その後、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した場合にオペレーションを許可するように ABAC ポリシーをします。

ABAC は、急成長する環境やポリシー管理が煩雑になる状況で役立ちます。

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。

ABAC の詳細については、「IAM ユーザーガイド」の「ABAC 認可でアクセス許可を定義する」を参照してください。ABAC をセットアップする手順を説明するチュートリアルについては、「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 (ABAC) を使用する」を参照してください。

Amazon EMR on EKS での一時認証情報の使用

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 を使用する機能などの詳細については、AWS のサービス「IAM ユーザーガイド」の「IAM と連携する」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は、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。ロールの切り替えに関する詳細については、「IAM ユーザーガイド」の「ユーザーから IAM ロールに切り替える (コンソール)」を参照してください。

一時的な認証情報は、 AWS CLI または AWS API を使用して手動で作成できます。その後、これらの一時的な認証情報を使用してアクセスすることができます 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 AWS を動的に生成することをお勧めします。詳細については、「IAM の一時的セキュリティ認証情報」を参照してください。

Amazon EMR on EKS のクロスサービスプリンシパル許可

転送アクセスセッション (FAS) のサポート: あり

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リクエストする を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アクセス許可が必要です。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。

Amazon EMR on EKS のサービスロール

サービスロールのサポート	いいえ

Amazon EMR on EKS 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	Yes

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、「IAM と提携するAWS のサービス」を参照してください。表の「サービスリンクロール」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、[はい] リンクを選択します。

Amazon EMR on EKS でのサービスにリンクされたロールの使用

Amazon EMR on EKS は AWS Identity and Access Management 、(IAM) サービスにリンクされたロールを使用します。サービスにリンクされたロールは、Amazon EMR on EKS 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、Amazon EMR on EKS によって事前定義されており、ユーザーに代わってサービスから他の AWS のサービスを呼び出すために必要なすべてのアクセス許可が含まれています。

サービスにリンクされたロールを使用すると、必要なアクセス許可を手動で追加する必要がなくなるため、Amazon EMR on EKS の設定が簡単になります。サービスにリンクされたロールのアクセス許可は、Amazon EMR on EKS により定義されます。特に指定されている場合を除き、Amazon EMR on EKS のみがそのロールを引き受けることができます。定義される許可は信頼ポリシーと許可ポリシーに含まれており、その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。

サービスリンク役割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、リソースへのアクセス許可を誤って削除することが防止され、Amazon EMR on EKS 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、「IAM と連携するAWS サービス」を参照して、サービスにリンクされたロール列がはいになっているサービスを見つけてください。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、はいリンクを選択します。

Amazon EMR on EKS でのサービスにリンクされたロールのアクセス許可

Amazon EMR on EKS では、サービスにリンクされたロール AWSServiceRoleForAmazonEMRContainers を使用します。

`AWSServiceRoleForAmazonEMRContainers` サービスリンク役割は役割の引き受けについて以下のサービスを信頼します。

- `emr-containers.amazonaws.com`

ロールのアクセス許可ポリシー `AmazonEMRContainersServiceRolePolicy` は、以下のポリシーステートメントで示すように、指定したリソースに対して一連のアクションを実行することを Amazon EMR on EKS に許可します。

> **Note**
>
> マネージドポリシーの内容は変わるため、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。up-to-dateポリシードキュメントは、「 AWS マネージドポリシーリファレンスガイド」のAmazonEMRContainersServiceRolePolicy」を参照してください。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "eks:DescribeCluster",
                "eks:ListNodeGroups",
                "eks:DescribeNodeGroup",
                "ec2:DescribeRouteTables",
                "ec2:DescribeSubnets",
                "ec2:DescribeSecurityGroups",
                "elasticloadbalancing:DescribeInstanceHealth",
                "elasticloadbalancing:DescribeLoadBalancers",
                "elasticloadbalancing:DescribeTargetGroups",
                "elasticloadbalancing:DescribeTargetHealth",
                "eks:ListPodIdentityAssociations",
                "eks:DescribePodIdentityAssociation"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "acm:ImportCertificate",
                "acm:AddTagsToCertificate"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/emr-container:endpoint:managed-certificate": "true"
                }
            }
        },
        {
```

```
            "Effect": "Allow",
            "Action": [
                "acm:DeleteCertificate"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/emr-container:endpoint:managed-certificate":
 "true"
                }
            }
        }
    ]
}
```

サービスリンク役割の作成、編集、削除を IAM エンティティ (ユーザー、グループ、役割など) に許可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、「IAM ユーザーガイド」の「サービスリンクロールの許可」を参照してください。

Amazon EMR on EKS でのサービスにリンクされたロールの作成

サービスにリンクされたロールを手動で作成する必要はありません。サービスにリンクされたロールは、仮想クラスターの作成時に Amazon EMR on EKS によって自動的に作成されます。

このサービスリンク役割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役割を再作成できます。サービスにリンクされたロールは、仮想クラスターの作成時に Amazon EMR on EKS によって再度作成されます。

IAM コンソールを使用して、Amazon EMR on EKS 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。 AWS CLI または AWS API で、サービス名を使用して`emr-containers.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、「IAM 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」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。

Amazon EMR on EKS でのサービスにリンクされたロールの編集

Amazon EMR on EKS では、`AWSServiceRoleForAmazonEMRContainers` の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。サービスにリンクされた役割を作成すると、多くのエンティティによって役割が参照される可能性があるため、役割名を変更することはできません。ただし、IAM を使用した役割の説明の編集はできます。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。

Amazon EMR on EKS でのサービスにリンクされたロールの削除

サービスリンク役割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役割を削除することをお勧めします。そうすることで、積極的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を排除できます。ただし、手動で削除する前に、サービスリンクロールのリソースをクリーンアップする必要があります。

> **Note**
>
> リソースの削除を試みた際に、対応するロールが Amazon EMR on EKS サービスで使用されている場合、削除が失敗することがあります。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。

`AWSServiceRoleForAmazonEMRContainers` で使用されている Amazon EMR on EKS リソースを削除するには

1. Amazon EMR コンソールを開きます。
2. 仮想クラスターを選択します。
3. `Virtual Cluster` ページで、[削除] を選択します。
4. この手順をアカウント内の他のすべての仮想クラスターに対して繰り返します。

サービスリンクロールを IAM で手動削除するには

IAM コンソール、 AWS CLI、または AWS API を使用して、`AWSServiceRoleForAmazonEMRContainers`サービスにリンクされたロールを削除します。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削除」を参照してください。

Amazon EMR on EKS のサービスにリンクされたロールがサポートされるリージョン

Amazon EMR on EKS では、このサービスを利用できる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がサポートされます。詳細については、「Amazon EMR on EKS サービスエンドポイントとサービスクォータ」を参照してください。

Amazon EMR on EKS の管理ポリシー

2021 年 3 月 1 日以降の Amazon EMR on EKS 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。

変更	説明	日付
AmazonEMRContainersServiceRolePolicy - クラスター内の EKS ポッド ID の関連付けを一覧表示する読み取りアクセス許可と、クラスター内のポッド ID の関連付けに関する説明情報を返す別の読み取りアクセス許可を追加しました。詳細については、「AmazonEMRContainersServiceRolePolicy」を参照してください。	ポリシーにはeks:ListPodIdentityAssociations 、、 のアクセス許可が追加されますeks:DescribePodIdentityAssociation 。	2023 年 2 月 3 日
AmazonEMRContainersServiceRolePolicy - Amazon EKS ノードグループの説明と一覧表示、ロードバランサーのターゲットグループの説明、ロードバランサーのターゲットヘルスの説明を行う権限を追加しました。	以下の権限がポリシーに追加されます: eks:ListNodeGroups 、eks:DescribeNodeGroup 、elasticloadbalancing:DescribeTargetGroups 、elasticloadbalancing:DescribeTargetHealth 。	2023 年 3 月 13 日
AmazonEMRContainersServiceRolePolicy - で証明書をインポートおよび削除するアクセス許可を追加しました AWS Certificate Manager。	以下の権限がポリシーに追加されます: acm:ImportCertificate 、acm:AddTagsToCertificate 、acm:DeleteCertificate 。	2021 年 12 月 3 日
Amazon EMR on EKS が変更の追跡を開始しました	Amazon EMR on EKS が AWS マネージドポリシーの変更の追跡を開始しました。	2021 年 3 月 1 日

Amazon EMR on EKS でのジョブ実行ロールの使用

StartJobRun コマンドを使用して EKS クラスターでのジョブ実行を送信するには、まず、仮想クラスターで使用するジョブ実行ロールをオンボーディングする必要があります。詳細については、「Amazon EMR on EKS のセットアップ」の ジョブ実行ロールを作成する を参照してください。また、Amazon EMR on EKS Workshop の「Create IAM Role for job execution」セクションの指示に従うこともできます。

ジョブ実行ロールの信頼ポリシーに、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",
      "Principal": {
        "Federated": "arn:aws:iam::AWS_ACCOUNT_ID:oidc-provider/OIDC_PROVIDER"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringLike": {
          "OIDC_PROVIDER:sub": "system:serviceaccount:NAMESPACE:emr-containers-sa-*-
*-AWS_ACCOUNT_ID-BASE36_ENCODED_ROLE_NAME"
        }
      }
    }
  ]
}
```

前述の例の信頼ポリシーでは、名前が emr-containers-sa-*-*-*AWS_ACCOUNT_ID*-*BASE36_ENCODED_ROLE_NAME* パターンに一致する Amazon EMR 管理の Kubernetes サービスアカウントにのみ権限が付与されます。このパターンのサービスアカウントは、ジョブの送信時に自動的に作成され、ジョブが送信される名前空間にスコープされます。この信頼ポリシーにより、これらのサービスアカウントは実行ロールを引き受けて、実行ロールの一時認証情報を取得できます。別の Amazon EKS クラスターのサービスアカウントや、同じ EKS クラスター内の別の名前空間のサービスアカウントは、実行ロールを引き受けることができません。

次のコマンドを実行すると、上記の形式で信頼ポリシーを自動的に更新できます。

```
aws emr-containers update-role-trust-policy \
       --cluster-name cluster \
       --namespace namespace \
       --role-name iam_role_name_for_job_execution
```

実行ロールへのアクセスの制御

Amazon EKS クラスターの管理者は、IAM 管理者が複数の実行ロールを追加できるマルチテナントの Amazon EMR on EKS 仮想クラスターを作成できます。信頼できないテナントであっても、こうした実行ロールを使用して、任意のコードを実行するジョブを送信できます。そのため、信頼できないテナントではこうした実行ロールの 1 つ以上に割り当てられた権限を取得するコードを実行できないように制限することをお勧めします。IAM ID にアタッチされている IAM ポリシーを制限する場合、IAM 管理者はオプションの Amazon リソースネーム (ARN) 条件キー `emr-containers:ExecutionRoleArn` を使用できます。この条件は、次の権限ポリシーで示すように、仮想クラスターに対する権限を持つ実行ロール ARN のリストを受け入れます。

```
  {
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "emr-containers:StartJobRun",
      "Resource": "arn:aws:emr-containers:REGION:AWS_ACCOUNT_ID:/
virtualclusters/VIRTUAL_CLUSTER_ID",
      "Condition": {
        "ArnEquals": {
          "emr-containers:ExecutionRoleArn": [
            "execution_role_arn_1",
            "execution_role_arn_2",
            ...
          ]
        }
      }
    }
  ]
}
```

`MyRole` など特定のプレフィックスで始まるすべての実行ロールを許可する場合は、条件演算子 `ArnEquals` を演算子 `ArnLike` に置き換えることができるほか、条件内の

execution_role_arn 値をワイルドカード文字 * に置き換えることができます。例えば、arn:aws:iam::*AWS_ACCOUNT_ID*:role/MyRole* と指定します。その他の ARN 条件キーもすべてサポートされています。

> **Note**
>
> Amazon EMR on EKS では、タグや属性に基づいて実行ロールに権限を付与することはできません。Amazon EMR on EKS は、実行ロールに対してタグベースのアクセスコントロール (TBAC) や属性ベースのアクセスコントロール (ABAC) をサポートしていません。

Amazon EMR on EKS 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、ユーザーおよびロールには Amazon EMR on EKS リソースを作成または変更するアクセス権限はありません。また、、 AWS Command Line Interface （ AWS CLI ） AWS Management Console、または AWS API を使用してタスクを実行することはできません。IAM 管理者は、リソースで必要なアクションを実行するための権限をユーザーに付与する IAM ポリシーを作成できます。その後、管理者はロールに IAM ポリシーを追加し、ユーザーはロールを引き継ぐことができます。

これらサンプルの JSON ポリシードキュメントを使用して、IAM アイデンティティ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 ポリシーを作成する (コンソール)」を参照してください。

Amazon EMR on EKS が定義するアクションとリソースタイプ (リソースタイプごとの ARN の形式を含む) の詳細については、「サービス認可リファレンス」の「Amazon EMR on EKS 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

- ポリシーに関するベストプラクティス
- Amazon EMR on EKS コンソールの使用
- 自分の権限の表示をユーザーに許可する

ポリシーに関するベストプラクティス

アイデンティティベースのポリシーは、アカウント内で誰かが Amazon EMR on EKS リソースを作成、アクセス、または削除できる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 管理ポリシーの使用を開始し、最小特権のアクセス許可に移行する – ユーザーとワークロードにアクセス許可の付与を開始するには、多くの一般的なユースケース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して、アクセス許可をさらに減らすことをお勧めします。詳細については、「IAM ユーザーガイド」の「AWS マネージドポリシー」または「ジョブ機能のAWS マネージドポリシー」を参照してください。
- 最小特権を適用する – IAM ポリシーで許可を設定する場合は、タスクの実行に必要な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「IAM でのポリシーとアクセス許可」を参照してください。
- IAM ポリシーで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、アクションや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 SSL を使用して送信するように指定できます。条件を使用して、サービスアクションが などの特定の を通じて使用される場合に AWS のサービス、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「IAM JSON 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、安全で機能的な権限を確保する - IAM Access Analyzer は、新規および既存のポリシーを検証して、ポリシーが IAM ポリシー言語 (JSON) および IAM のベストプラクティスに準拠するようにします。IAM アクセスアナライザーは 100 を超えるポリシーチェックと実用的な推奨事項を提供し、安全で機能的なポリシーの作成をサポートします。詳細については、「IAM ユーザーガイド」の「IAM Access Analyzer でポリシーを検証する」を参照してください。
- 多要素認証 (MFA) を要求する – 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、セキュリティを強化するために MFA を有効にします。API オペレーションが呼び出されるときに MFA を必須にするには、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 を使用した安全な API アクセス」を参照してください。

IAM で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 IAM でのセキュリティのベストプラクティスを参照してください。

Amazon EMR on EKS コンソールの使用

Amazon EMR on EKS コンソールにアクセスするには、一連の最小限のアクセス許可が必要です。これらのアクセス許可により、 AWS アカウントアカウントの Amazon EMR on EKS リソースの詳細をリストおよび表示することを許可する必要があります。最小限必要な許可よりも制限が厳しいアイデンティティベースのポリシーを作成すると、そのポリシーを持つエンティティ (ユーザーまたはロール) に対してコンソールが意図したとおりに機能しません。

AWS CLI または AWS API のみを呼び出すユーザーには、最小限のコンソールアクセス許可を付与する必要はありません。代わりに、実行しようとしている API オペレーションに一致するアクションのみへのアクセスが許可されます。

ユーザーとロールが引き続き Amazon EMR on EKS コンソールを使用できるようにするには、エンティティに Amazon EMR on EKS `ConsoleAccess`または `ReadOnly` AWS マネージドポリシーもアタッチします。詳細については、「IAM ユーザーガイド」の「ユーザーへのアクセス許可の追加」を参照してください。

自分の権限の表示をユーザーに許可する

この例では、ユーザーアイデンティティにアタッチされたインラインおよびマネージドポリシーの表示を IAM ユーザーに許可するポリシーの作成方法を示します。このポリシーには、コンソールで、または AWS CLI または AWS API 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。

```
{
    "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": "*"
        }
    ]
}
```

タグベースのアクセスコントロールのポリシー

アイデンティティベースのポリシーで条件を使用し、タグに基づいて仮想クラスターおよびジョブ実行へのアクセスを制御できます。タグ付けの詳細については、「Amazon EMR on EKS リソースのタグ付け」を参照してください。

次の例では、Amazon EMR on EKS 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について説明しています。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、本稼働環境で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 ポリシーの計画および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。

以下の例では、Amazon EMR on EKS 仮想クラスターで許可されるアクションを制御するために使用するアイデンティティベースの許可ポリシーを説明しています。

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

次のポリシーの例では、StringEquals 条件演算子は、dev をタグ department の値 と一致させるように試みます。タグ department が仮想クラスターに追加されていない、またはタグ department に値 dev が含まれていない場合は、ポリシーは適用されず、このアクションはポリシーによって許可されません。アクションを許可するポリシーステートメントが他にない場合は、ユーザーはこの値が含まれたこのタグのある仮想クラスターでのみ作業できます。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "emr-containers:DescribeVirtualCluster"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws: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 test が含まれた仮想クラスターでアクションを許可するには、以下のように、前術の例の条件ブロックを置き換えることができます。

```
"Condition": {
        "StringEquals": {
          "aws: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、仮想クラスター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-containers:CreateVirtualCluster"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/department": "dev"
        }
      }
    }
  ]
}
```

次のポリシーステートメントは、任意の値を含めることができる department タグがクラスターにある場合にのみ、ユーザーに仮想クラスターの作成を許可します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "emr-containers:CreateVirtualCluster"
      ],
      "Resource": "*",
      "Condition": {
        "Null": {
          "aws: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

このポリシーの効果は、department 値が含まれる dev タグでタグ付けされる仮想クラスターに任意のタグを追加または削除するアクセス許可を、ユーザーに対して拒否することにあります。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "emr-containers:TagResource",
        "emr-containers:UntagResource"
      ],
      "Resource": "*",
      "Condition": {
        "StringNotEquals": {
          "aws:ResourceTag/department": "dev"
        }
      }
    }
  ]
}
```

Amazon EMR on EKS 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 on EKS と IAM の使用時に発生する可能性がある一般的な問題の診断や修正に役立ちます。

トピック

- Amazon EMR on EKS でアクションを実行する認可がない
- iam:PassRole を実行する権限がない
- AWS アカウント外のユーザーに Amazon EMR on EKS リソースへのアクセスを許可したい

Amazon EMR on EKS でアクションを実行する認可がない

からアクションを実行する権限がないと AWS Management Console 通知された場合は、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-containers:*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-
containers:GetWidget on resource: my-example-widget
```

この場合、Mateo は、emr-containers:*GetWidget* アクションを使用して *my-example-widget* リソースにアクセスできるように、管理者にポリシーの更新を依頼します。

iam:PassRole を実行する権限がない

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、ポリシーを更新して Amazon EMR on EKS にロールを渡せるように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、既存のロールをそのサービスに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。

以下の例のエラーは、marymajor という IAM ユーザーがコンソールを使用して Amazon EMR on EKS でアクションを実行しようとする場合に発生します。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 on EKS 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 (ACL) をサポートするサービスの場合、それらのポリシーを使用して、リソースへのアクセスを付与できます。

詳細については、以下を参照してください:

- Amazon EMR on EKS がこれらの機能をサポートしているかどうかについては、「Amazon EMR on EKS で IAM を使用する方法」を参照してください。
- 所有 AWS アカウント している 全体のリソースへのアクセスを提供する方法については、IAM ユーザーガイドの「所有 AWS アカウント している別の の IAM ユーザーへのアクセスを提供する」を参照してください。
- 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介してアクセスを提供する方法については、「IAM ユーザーガイド」の「外部で認証されたユーザー (ID フェデレーション) へのアクセスの許可」を参照してください。
- クロスアカウントアクセスにおけるロールとリソースベースのポリシーの使用方法の違いについては、「IAM ユーザーガイド」の「IAM でのクロスアカウントのリソースへのアクセス」を参照してください。

AWS Lake Formation での Amazon EMR on EKS を使用したきめ細かなアクセスコントロール

Amazon EMR リリース 7.7 以降では、 AWS Lake Formation を活用して、Amazon S3 バケットでバックアップされた AWS Glue 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、Amazon EMR on EKS 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。

トピック

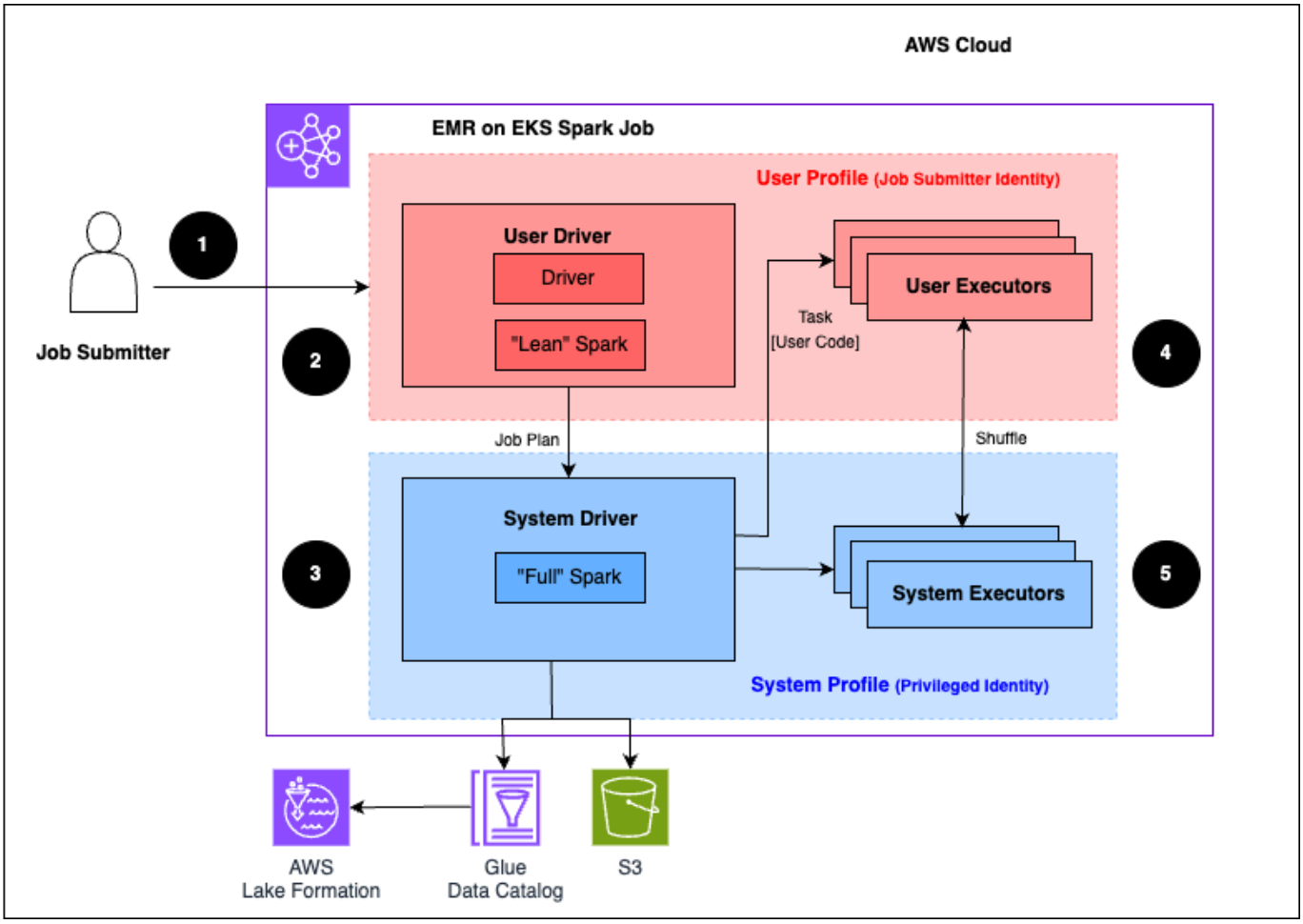
- Amazon EMR on EKS と AWS Lake Formation の連携方法
- Amazon EMR on EKS で Lake Formation を有効にする

Amazon EMR on EKS と AWS Lake Formation の連携方法

Lake Formation で Amazon EMR on EKS 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、Amazon EMR on EKS がジョブを実行するときに Lake Formation アクセス許可コントロールを適用できます。Amazon EMR on EKS 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が指定したコードを実行し、システムプロファイルは Lake Formation ポリシーを適用します。Lake Formation

が有効な各ジョブは、2 つの Spark ドライバーを使用します。1 つはユーザープロファイル用、もう 1 つはシステムプロファイル用です。詳細については、「What is AWS Lake Formation」を参照してください。

以下は、Amazon EMR on EKS 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。



次の手順では、このプロセスについて説明します。

1. ユーザーは、 AWS Lake Formation 対応の Amazon EMR on EKS 仮想クラスターに Spark ジョブを送信します。
2. Amazon EMR on EKS サービスはユーザードライバーを設定し、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、エグゼキューターのリクエスト、Amazon S3 または Glue データカタログ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。ジョブプランのみを構築します。
3. Amazon EMR on EKS サービスは、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。Amazon EKS は、通信用の 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。完全な Spark を実行し、データアクセスのために Amazon S3 およびデータカタログと通信します。エグゼキュターをリクエストし、ジョブプランを実行ステージのシーケンスにコンパイルします。

4. Amazon EMR on EKS サービスは、エグゼキュターでステージを実行します。任意のステージのユーザーコードは、ユーザープロファイルエグゼキュターでのみ実行されます。
5. Lake Formation で保護されたデータカタログテーブルからデータを読み取るステージ、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ターに委任されます。

Amazon EMR on EKS で Lake Formation を有効にする

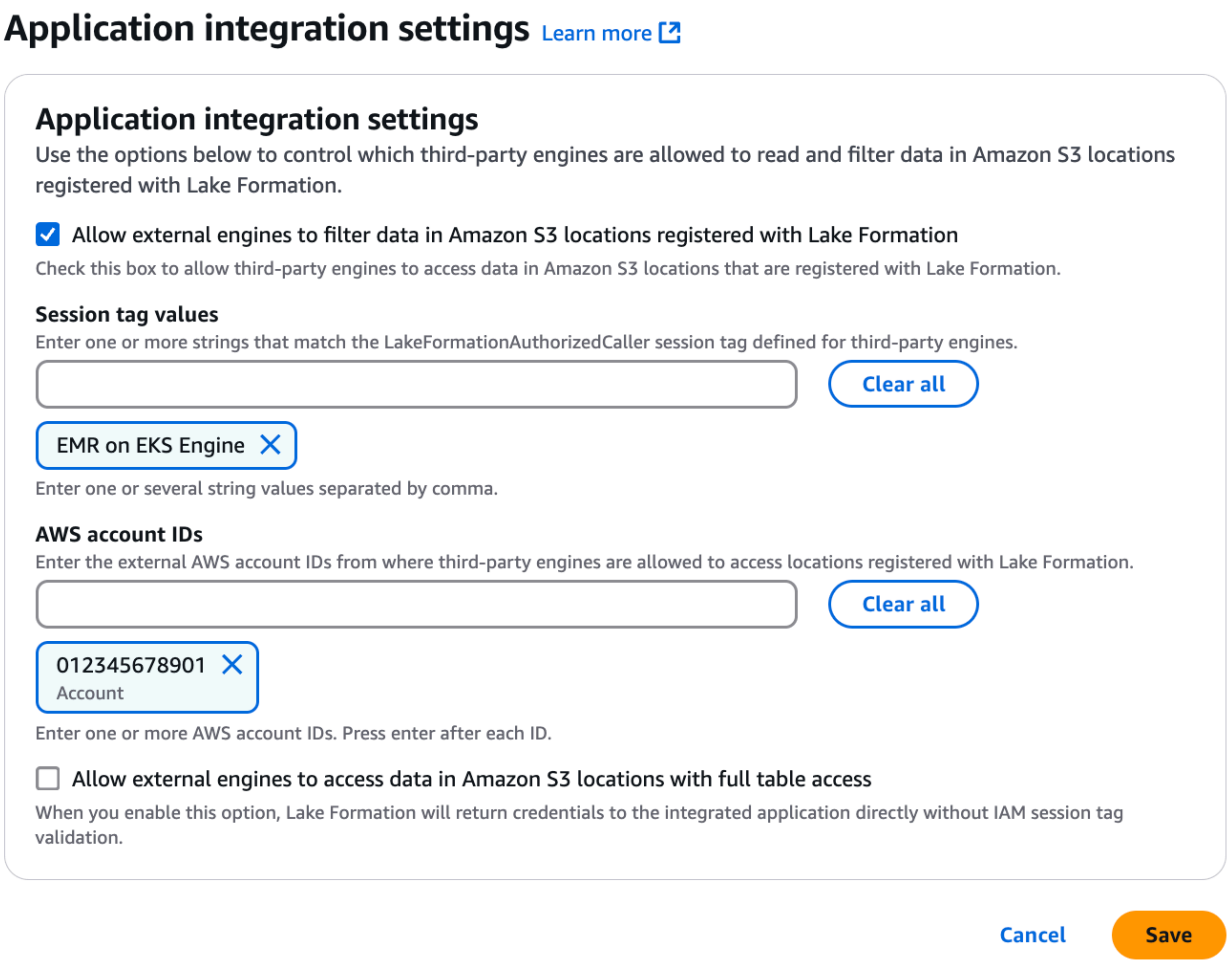
Amazon EMR リリース 7.7 以降では、 AWS Lake Formation を活用して、Amazon S3 でバックアップされた Data Catalog テーブルにきめ細かなアクセスコントロールを適用できます。この機能を使用すると、Amazon EMR on EKS Spark ジョブ内の読み取りクエリのテーブル、行、列、セルレベルのアクセスコントロールを設定できます。

このセクションでは、セキュリティ設定を作成し、Amazon EMR と連携するように Lake Formation を設定する方法について説明します。また、Lake Formation 用に作成したセキュリティ設定を使用して仮想クラスターを作成する方法についても説明します。これらのセクションは順番に完了することを目的としています。

ステップ 1: Lake Formation ベースの列、行、またはセルレベルのアクセス許可を設定する

まず、Lake Formation で行レベルと列レベルのアクセス許可を適用するには、Lake Formation のデータレイク管理者が LakeFormationAuthorizedCaller セッションタグを設定する必要があります。Lake Formation は、このセッションタグを使用して発信者を承認し、データレイクへのアクセス権限を付与します。

AWS Lake Formation コンソールに移動し、サイドバーの管理セクションからアプリケーション統合設定オプションを選択します。次に、Lake Formation に登録されている Amazon S3 ロケーション内のデータを外部エンジンがフィルタリングできるようにするチェックボックスをオンにします。Spark ジョブが実行されているAWS アカウント IDs とセッションタグの値を追加します。



ここで渡された LakeFormationAuthorizedCaller セッションタグは、後でセクション 3 で IAM ロールを設定するときに SecurityConfiguration に渡されることに注意してください。

ステップ 2: EKS RBAC アクセス許可を設定する

次に、ロールベースのアクセスコントロールのアクセス許可を設定します。

Amazon EMR on EKS サービスに EKS クラスターのアクセス許可を付与する

Amazon EMR on EKS Service には EKS クラスターロールのアクセス許可が必要です。これにより、システムドライバーがユーザー名前空間のユーザーエグゼキューターをスピンオフするためのクロス名前空間アクセス許可を作成できます。

クラスターロールを作成する

このサンプルでは、 リソースのコレクションに対するアクセス許可を定義します。

```
vim emr-containers-cluster-role.yaml
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: emr-containers
rules:
  - apiGroups: [""]
    resources: ["namespaces"]
    verbs: ["get"]
  - apiGroups: [""]
    resources: ["serviceaccounts", "services", "configmaps", "events", "pods", "pods/
log"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
 "deletecollection", "annotate", "patch", "label"]
  - apiGroups: [""]
    resources: ["secrets"]
    verbs: ["create", "patch", "delete", "watch"]
  - apiGroups: ["apps"]
    resources: ["statefulsets", "deployments"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete", "annotate",
 "patch", "label"]
  - apiGroups: ["batch"]
    resources: ["jobs"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete", "annotate",
 "patch", "label"]
  - apiGroups: ["extensions", "networking.k8s.io"]
    resources: ["ingresses"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete", "annotate",
 "patch", "label"]
  - apiGroups: ["rbac.authorization.k8s.io"]
    resources: ["clusterroles","clusterrolebindings","roles", "rolebindings"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
 "deletecollection", "annotate", "patch", "label"]
  - apiGroups: [""]
    resources: ["persistentvolumeclaims"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
 "deletecollection", "annotate", "patch", "label"]
---
```

```
kubectl apply -f emr-containers-cluster-role.yaml
```

クラスターロールバインディングを作成する

```
vim emr-containers-cluster-role-binding.yaml
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: emr-containers
subjects:
- kind: User
  name: emr-containers
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: emr-containers
  apiGroup: rbac.authorization.k8s.io
---
```

```
kubectl apply -f emr-containers-cluster-role-binding.yaml
```

Amazon EMR on EKS サービスへの名前空間アクセスを提供する

2 つの Kubernetes 名前空間を作成します。1 つはユーザードライバーとエグゼキュター用、もう 1 つはシステムドライバーとエグゼキュター用で、Amazon EMR on EKS サービスアクセスを有効にしてユーザーとシステムの名前空間の両方でジョブを送信します。既存のガイドに従って、「 を使用してクラスターアクセスを有効にするaws-auth」で利用可能な各名前空間へのアクセスを提供します。

ステップ 3: ユーザーおよびシステムプロファイルコンポーネントの IAM ロールを設定する

3 つ目は、特定のコンポーネントのロールを設定します。Lake Formation 対応の Spark ジョブには、ユーザーとシステムの 2 つのコンポーネントがあります。ユーザードライバーとエグゼキュターはユーザー名前空間で実行され、StartJobRun API で渡される JobExecutionRole に関連付けられます。システムドライバーとエグゼキュターはシステム名前空間で実行され、QueryEngine ロールに関連付けられます。

クエリエンジンロールを設定する

QueryEngine ロールはシステムスペースコンポーネントに関連付けられており、LakeFormationAuthorizedCaller セッションタグを持つ JobExecutionRole を引き受けるアクセス許可を持ちます。クエリエンジンロールの IAM アクセス許可ポリシーは次のとおりです。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AssumeJobRoleWithSessionTagAccessForSystemDriver",
            "Effect": "Allow",
            "Action": [
                "sts:AssumeRole",
                "sts:TagSession"
            ],
            "Resource": "arn:aws:iam::Account:role/JobExecutionRole",
            "Condition": {
                "StringLike": {
                    "aws:RequestTag/LakeFormationAuthorizedCaller": "EMR on EKS Engine"
                }
            }
        },
        {
            "Sid": "AssumeJobRoleWithSessionTagAccessForSystemExecutor",
            "Effect": "Allow",
            "Action": [
                "sts:AssumeRole"
            ],
            "Resource": "arn:aws:iam::Account:role/JobExecutionRole",
        },
        {
            "Sid": "CreateCertificateAccessForTLS",
            "Effect": "Allow",
            "Action": "emr-containers:CreateCertificate",
            "Resource": "*"
        }
    ]
}
```

Kubernetes システム名前空間を信頼するように、クエリエンジンロールの信頼ポリシーを設定します。

```
aws emr-containers update-role-trust-policy \
    --cluster-name eks cluster \
    --namespace eks system namespace \
    --role-name query_engine_iam_role_name
```

詳細については、「ロールの信頼ポリシーの更新」を参照してください。

ジョブ実行ロールを設定する

Lake Formation のアクセス許可は、 Glue Data Catalog AWS リソース、Amazon S3 ロケーション、およびそれらのロケーションの基盤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 および AWS Glue APIsとリソースへのアクセスを制御します。データカタログ (SELECT) 内のテーブルにアクセスする Lake Formation 許可があるかもしれませんが、glue:Get*API オペレーションに対する IAM 許可がない場合、オペレーションは失敗します。

JobExecutionRole の IAM アクセス許可ポリシー: JobExecution ロールには、アクセス許可ポリシーのポリシーステートメントが必要です。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "GlueCatalogAccess",
            "Effect": "Allow",
            "Action": [
                "glue:Get*",
                "glue:Create*",
                "glue:Update*"
            ],
            "Resource": ["*"]
        },
        {
            "Sid": "LakeFormationAccess",
            "Effect": "Allow",
            "Action": [
                "lakeformation:GetDataAccess"
            ],
            "Resource": ["*"]
        },
        {
            "Sid": "CreateCertificateAccessForTLS",
```

```
            "Effect": "Allow",
            "Action": "emr-containers:CreateCertificate",
            "Resource": "*"
        }
    ]
}
```

JobExecutionRole の IAM 信頼ポリシー：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "TrustQueryEngineRoleForSystemDriver",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::your_account:role/QueryExecutionRole"
            },
            "Action": [
                "sts:AssumeRole",
                "sts:TagSession"
            ],
            "Condition": {
                "StringLike": {
                    "aws:RequestTag/LakeFormationAuthorizedCaller": "EMR on EKS Engine"
                }
            }
        },
        {
            "Sid": "TrustQueryEngineRoleForSystemExecutor",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::your_account:role/QueryEngineRole"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Kubernetes ユーザー名前空間を信頼するようにジョブ実行ロールの信頼ポリシーを設定します。

```
aws emr-containers update-role-trust-policy \
    --cluster-name eks cluster \
```

```
    --namespace eks User namespace \
    --role-name job_execution_role_name
```

詳細については、「ジョブ実行ロールの信頼ポリシーを更新する」を参照してください。

ステップ 4: セキュリティ設定をセットアップする

Lake Formation 対応ジョブを実行するには、セキュリティ設定を作成する必要があります。

```
aws emr-containers create-security-configuration \
    --name 'security-configuration-name' \
    --security-configuration '{
        "authorizationConfiguration": {
            "lakeFormationConfiguration": {
                "authorizedSessionTagValue": "SessionTag configured in LakeFormation",
                "secureNamespaceInfo": {
                    "clusterId": "eks-cluster-name",
                    "namespace": "system-namespace-name"
                },
                "queryEngineRoleArn": "query-engine-IAM-role-ARN"
            }
        }
    }'
```

authorizedSessionTagValue フィールドに渡されたセッションタグが Lake Formation を承認できることを確認します。の値を Lake Formation で設定された値に設定しますステップ 1: Lake Formation ベースの列、行、またはセルレベルのアクセス許可を設定する。

ステップ 5: 仮想クラスターを作成する

セキュリティ設定を使用して Amazon EMR on EKS 仮想クラスターを作成します。

```
aws emr-containers create-virtual-cluster \
--name my-lf-enabled-vc \
--container-provider '{
    "id": "eks-cluster",
    "type": "EKS",
    "info": {
        "eksInfo": {
            "namespace": "user-namespace"
        }
    }
```

```
}' \
--security-configuration-id SecurityConfiguraionId
```

Lake Formation 認可設定が仮想クラスターで実行されているすべてのジョブに適用されるように、前のステップの SecurityConfiguration ID が渡されていることを確認します。詳細については、「Amazon EMR で Amazon EKS クラスターを登録する」を参照してください。

ステップ 6: FGAC 対応 VirtualCluster でジョブを送信する

ジョブ送信プロセスは、Lake Formation 以外のジョブと Lake Formation ジョブの両方で同じです。詳細については、「 でジョブ実行を送信するStartJobRun」を参照してください。

システムドライバーの Spark ドライバー、エグゼキュター、イベントログは、デバッグのために AWS サービスアカウントの S3 バケットに保存されます。ジョブ実行でカスタマー管理の KMS キーを設定して、 AWS サービスバケットに保存されているすべてのログを暗号化することをお勧めします。ログ暗号化の有効化の詳細については、「Amazon EMR on EKS ログの暗号化」を参照してください。

ログ記録とモニタリング

インシデントを検出し、インシデントの発生時にアラートを受け取り、それらのアラートに対応するには、以下のオプションを Amazon EMR on EKS で使用します。

- Amazon EMR on EKS を でモニタリングする AWS CloudTrail - Amazon EMR on EKS のユーザー、ロール、または AWS サービスによって実行されたアクションの記録AWS CloudTrailを提供します。Amazon EMR コンソールからの呼び出しと Amazon EMR on EKS API オペレーションへのコード呼び出しがイベントとしてキャプチャされます。これにより、Amazon EMR on EKS に対するリクエスト、リクエスト元の IP アドレス、リクエストの実行者、リクエスト日時などの詳細を把握できます。詳細については、「を使用した Amazon EMR on EKS API コールのログ記録 AWS CloudTrail」を参照してください。
- Amazon EMR on EKS で CloudWatch Events を使用する - CloudWatch Events 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を提供します。CloudWatch Events は、運用上の変更が生じると同時にそれらを認識して対応し、環境に応答するためのメッセージを送信する、機能をアクティブ化する、変更を行う、および状態情報を収集することによって、必要に即した是正措置を講じます。Amazon EMR on EKS で CloudWatch Events を使用するには、CloudTrail 経由で Amazon EMR on EKS API コールでトリガーされるルールを作成します。詳細については、「Amazon CloudWatch Events でジョブをモニタリングする」を参照してください。

マネージドストレージを使用した Amazon EMR on EKS ログの暗号化

以下のセクションでは、 ログの暗号化を設定する方法を示します。

Enable encryption

独自の KMS キーを使用してマネージドストレージのログを暗号化するには、ジョブ実行を送信するときに次の設定を使用します。

```
"monitoringConfiguration": {
            "managedLogs": {
                "allowAWSToRetainLogs":"ENABLED",
                "encryptionKeyArn":"KMS key arn"
            },
            "persistentAppUI": "ENABLED"
        }
```

`allowAWSToRetainLogs` この設定により、 AWS はネイティブ FGAC を使用してジョブを実行するときにシステム名前空間ログを保持できます。`persistentAppUI` この設定により AWS 、 は Spark UI の生成に使用されるイベントログを保存できます。`encryptionKeyArn` は、 によって保存されるログの暗号化に使用する KMS キー ARN を指定するために使用されます AWS。

ログ暗号化に必要なアクセス許可

ジョブを送信するか、Spark UI を表示するユーザーは、`kms:Decrypt`暗号化キーのアクション `kms:DescribeKey`、`kms:GenerateDataKey`、および を許可される必要があります。これらのアクセス許可は、キーの有効性を検証し、ユーザーが KMS キーで暗号化されたログを読み書きするために必要なアクセス許可を持っていることを確認するために使用されます。ジョブを送信するユーザーに必要なキーアクセス許可がない場合、Amazon EMR on EKS はジョブ実行の送信を拒否します。

StartJobRun の呼び出しに使用されるロールの IAM ポリシーの例

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": "emr-containers:StartJobRun",
            "Resource": "*",
            "Effect": "Allow"
        },
```

```
        {
            "Action": [
                "kms:DescribeKey",
                "kms:Decrypt",
                "kms:GenerateDataKey"
            ],
            "Resource": "KMS key ARN",
            "Effect": "Allow"
        }
    ]
}
```

また、 persistentappui.elasticmapreduce.amazonaws.comおよび elasticmapreduce.amazonaws.comサービスプリンシパルに kms:GenerateDataKeyおよびを許可するように KMS キーを設定する必要がありますkms:Decrypt。これにより、EMR は KMS キーで暗号化されたログをマネージドストレージに読み書きできます。

KMS キーポリシーの例

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "IAM role ARN used to call StartJobRun"
            },
            "Action": "kms:DescribeKey",
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "kms:viaService": "emr-containers.region.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "IAM role ARN used to call StartJobRun"
            },
            "Action": [
                "kms:Decrypt",
                "kms:GenerateDataKey"
```

```
            ],
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "kms:viaService": "emr-containers.region.amazonaws.com",
                    "kms:EncryptionContext:aws:emr-containers:virtualClusterId":
 "virtual cluster id"
                }
            }
        },
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "persistentappui.elasticmapreduce.amazonaws.com",
                    "elasticmapreduce.amazonaws.com"
                ]
            },
            "Action": [
                "kms:Decrypt",
                "kms:GenerateDataKey"
            ],
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "kms:EncryptionContext:aws:emr-containers:virtualClusterId":
 "virtual cluster id",
                    "aws:SourceArn": "virtual cluster ARN"
                }
            }
        }
    ]
}
```

セキュリティのベストプラクティスとして、kms:viaService、kms:EncryptionContext、および aws:SourceArn条件を追加することをお勧めします。これらの条件は、キーが Amazon EMR on EKS によってのみ使用され、特定の仮想クラスターで実行されているジョブから生成されたログにのみ使用されるようにするのに役立ちます。

を使用した Amazon EMR on EKS API コールのログ記録 AWS CloudTrail

Amazon EMR on EKS は AWS CloudTrail、Amazon EMR on EKS のユーザー、ロール、または サービスによって実行されたアクションを記録する AWS サービスである と統合されていま

す。CloudTrail は、Amazon EMR on EKS に対するすべての API コールをイベントとしてキャプチャします。キャプチャされる呼び出しには、Amazon EMR on EKS コンソールからの呼び出しと、Amazon EMR on EKS 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、Amazon EMR on EKS のイベントなど、Amazon S3 バケットへの CloudTrail 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。CloudTrail で収集された情報を使用して、Amazon EMR on EKS に対するリクエスト、リクエスト元の IP アドレス、リクエストを行った人、リクエスト日時などの詳細を確認できます。

CloudTrail の詳細については、「AWS CloudTrail ユーザーガイド」を参照してください。

CloudTrail の Amazon EMR on EKS 情報

CloudTrail は、 AWS アカウントの作成時にアカウントで有効になります。Amazon EMR on EKS でアクティビティが発生すると、そのアクティビティはイベント履歴の他の AWS サービスイベントとともに CloudTrail イベントに記録されます。 AWS アカウントで最近のイベントを表示、検索、ダウンロードできます。詳細については、 CloudTrail イベント履歴でのイベントの表示を参照してください。

Amazon EMR on EKS のイベントなど、 AWS アカウントのイベントの継続的な記録については、証跡を作成します。追跡により、CloudTrail はログファイルを Amazon S3 バケットに配信できます。デフォルトでは、コンソールで証跡を作成すると、証跡はすべての AWS リージョンに適用されます。証跡は、 AWS パーティション内のすべてのリージョンからのイベントを記録し、指定した Amazon S3 バケットにログファイルを配信します。さらに、CloudTrail ログで収集されたイベントデータをより詳細に分析し、それに基づいて行動するように、他の AWS サービスを設定できます。詳細については、次を参照してください:

- 追跡を作成するための概要
- 「CloudTrail がサポートされているサービスと統合」
- 「CloudTrail の Amazon SNS 通知の設定」
- 「複数のリージョンから CloudTrail ログファイルを受け取る」および「複数のアカウントから CloudTrail ログファイルを受け取る」

すべての Amazon EMR on EKS アクションは、CloudTrail が記録します。これらの説明については、「Amazon EMR on EKS API リファレンス」を参照してください。例えば、`CreateVirtualCluster`、`StartJobRun`、`ListJobRuns` の各アクションを呼び出すと、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。

- リクエストがルートまたは AWS Identity and Access Management (IAM) ユーザー認証情報を使用して行われたかどうか。
- リクエストがロール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 AWS サービスによって行われたかどうか。

詳細については、「CloudTrail userIdentity エレメント」を参照してください。

Amazon EMR on EKS ログファイルエントリの理解

「トレイル」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するように設定できます。CloudTrail のログファイルは、単一か複数のログエントリを含みます。イベントは任意ソースからの単一リクエストを表し、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の情報を含みます。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。

以下の例は、`ListJobRuns` アクションを示す CloudTrail ログエントリです。

```
{
  "eventVersion": "1.05",
  "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": {
        "mfaAuthenticated": "false",
        "creationDate": "2020-11-04T21:49:36Z"
```

```
      }
    }
  },
  "eventTime": "2020-11-04T21:52:58Z",
  "eventSource": "emr-containers.amazonaws.com",
  "eventName": "ListJobRuns",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "203.0.113.1",
  "userAgent": "aws-cli/1.11.167 Python/2.7.10 Darwin/16.7.0 botocore/1.7.25",
  "requestParameters": {
    "virtualClusterId": "1K48XXXXXXHCB"
  },
  "responseElements": null,
  "requestID": "890b8639-e51f-11e7-b038-EXAMPLE",
  "eventID": "874f89fa-70fc-4798-bc00-EXAMPLE",
  "readOnly": true,
  "eventType": "AwsApiCall",
  "recipientAccountId": "012345678910"
}
```

Amazon EMR on EKS での Amazon S3 Access Grants の使用

Amazon EMR on EKS での S3 Access Grants の概要

Amazon EMR リリース 6.15.0 以降では、Amazon S3 Access Grants によってスケーラブルなアクセスコントロールソリューションが提供され、これを使用して EKS 上の Amazon EMR から Amazon S3 データへのアクセスを強化できます。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 を使用すると、Amazon S3 データへのアクセスを、ランタイムロールや Amazon EMR on EKS クラスターへのアクセス権を持つアイデンティティにアタッチされている IAM ロールによって付与される権限を超えて、Amazon S3 データへのアクセスを強化できます。

詳細については、「Amazon EMR 管理ガイド」の「Amazon EMR の S3 アクセス許可によるアクセスの管理」および「Amazon Simple Storage Service ユーザーガイド」の「S3 アクセス許可によるアクセスの管理」を参照してください。

このページでは、S3 Access Grants 統合による EKS 上の Amazon EMR で Spark ジョブを実行するための要件について説明します。EKS 上の Amazon EMR を使用する場合、S3 Access Grants では、ジョブの実行ロールに IAM ポリシーステートメントを追加し、`StartJobRun` API のオーバー

ライド設定を追加する必要があります。他の Amazon EMR デプロイで S3 Access Grants を設定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 EMR での S3 Access Grants の使用
- EMR Serverless での S3 Access Grants の使用

データ管理用 S3 Access Grants を使用した Amazon EMR on EKS クラスターの起動

EKS 上の Amazon EMR で S3 Access Grants を有効にし、Spark ジョブを起動することができます。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. Amazon EMR on EKS クラスターのジョブ実行ロールを設定します。Spark ジョブの実行に必要な IAM アクセス許可 (`s3:GetDataAccess` および `s3:GetAccessGrantsInstanceForPrefix`) を含めてください。

```
{
    "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**
>
> ジョブ実行用に S3 に直接アクセスするための追加の権限を持つ IAM ロールを指定すると、S3 Access Grants で定義した権限に関係なく、ユーザーはデータにアクセスできる可能性があります。

2. 次の例のように、Amazon EMR リリースラベルが 6.15 以上で、`emrfs-site` 分類が設定された Amazon EMR on EKS クラスターにジョブを送信します。*`red text`* の値を使用シナリオに適した値に置き換えます。

```
{
  "name": "myjob",
  "virtualClusterId": "123456",
  "executionRoleArn": "iam_role_name_for_job_execution",
  "releaseLabel": "emr-7.6.0-latest",
  "jobDriver": {
    "sparkSubmitJobDriver": {
      "entryPoint": "entryPoint_location",
      "entryPointArguments": ["argument1", "argument2"],
       "sparkSubmitParameters": "--class main_class"
    }
  },
  "configurationOverrides": {
    "applicationConfiguration": [
      {
        "classification": "emrfs-site",
        "properties": {
          "fs.s3.s3AccessGrants.enabled": "true",
          "fs.s3.s3AccessGrants.fallbackToIAM": "false"
         }
      }
    ],
  }
}
```

Amazon EMR on EKS での S3 Access Grants の考慮事項

EKS 上の Amazon EMR で Amazon S3 Access Grants を使用する際の重要なサポート、互換性、および動作情報については、「Amazon EMR 管理ガイド」の「Amazon EMR での S3 Access Grants の考慮事項」を参照してください。

Amazon EMR on EKS のコンプライアンス検証

サードパーティーの監査者は、複数の コンプライアンスプログラムの一環として Amazon EMR on EKS のセキュリティと AWS コンプライアンスを評価します。これらのプログラムには、SOC、PCI、FedRAMP、HIPAA などがあります。

Amazon EMR on EKS の耐障害性

AWS グローバルインフラストラクチャは、 AWS リージョンとアベイラビリティーゾーンを中心に構築されています。 AWS リージョンは、低レイテンシー、高スループット、および高度に冗長なネットワークで接続された、物理的に分離および分離された複数のアベイラビリティーゾーンを提供します。アベイラビリティーゾーンでは、ゾーン間で中断することなく自動的にフェイルオーバーするアプリケーションとデータベースを設計および運用することができます。アベイラビリティーゾーンは、従来の単一または複数のデータセンターインフラストラクチャよりも可用性が高く、フォールトトレラントで、スケーラブルです。

AWS リージョンとアベイラビリティーゾーンの詳細については、AWS 「 グローバルインフラストラクチャ」を参照してください。

Amazon EMR on EKS は、 AWS グローバルインフラストラクチャに加えて、EMRFS を介して Amazon S3 との統合を提供し、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 on EKS で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 グローバル AWS ネットワークセキュリティで保護されています。 AWS セキュリティサービスと がインフラストラクチャ AWS を保護する方法については、AWS 「 クラウドセキュリティ」を参照してください。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して AWS 環境を設計するには、「セキュリティの柱 AWS Well-Architected フレームワーク」の「インフラストラクチャの保護」を参照してください。

AWS が公開した API コールを使用して、ネットワーク経由で Amazon EMR にアクセスします。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)。TLS 1.2 が必須で、TLS 1.3 をお勧めします。
- DHE (楕円ディフィー・ヘルマン鍵共有) や ECDHE (楕円曲線ディフィー・ヘルマン鍵共有) などの完全前方秘匿性 (PFS) による暗号スイート。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。

また、リクエストにはアクセスキー ID と、IAM プリンシパルに関連付けられているシークレットアクセスキーを使用して署名する必要があります。または、AWS Security Token Service (AWS STS) を使用して、一時的なセキュリティ認証情報を生成し、リクエストに署名することもできます。

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 on EKS 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)

インターフェイス VPC エンドポイントを使用して Amazon EMR on EKS に接続する

インターネット経由で接続するのではなく、Virtual Private Cloud (VPC) のインターフェイス VPC エンドポイント (AWS PrivateLink) を使用して Amazon EMR on EKS に直接接続できます。インターフェイス VPC エンドポイントを使用する場合、VPC と Amazon EMR on EKS 間の通信は、完全に AWS ネットワーク内で実施されます。各 VPC エンドポイントは、VPC サブネット内のプライベート IP アドレスを持つ 1 つ以上の Elastic Network Interface (ENI) で表されます。

インターフェイス VPC エンドポイントは、インターネットゲートウェイ、NAT デバイス、VPN 接続、または AWS Direct Connect 接続なしで、VPC を Amazon EMR on EKS に直接接続します。VPC のインスタンスは、パブリック IP アドレスがなくても Amazon EMR on EKS API と通信できます。

インターフェイス VPC エンドポイントを作成して、 AWS Management Console または AWS Command Line Interface （ AWS CLI) コマンドを使用して Amazon EMR on EKS に接続できます。詳細については、インターフェイスエンドポイントの作成を参照してください。

インターフェイス VPC エンドポイントを作成した後、エンドポイントのプライベート DNS ホスト名を有効にすると、デフォルトの Amazon EMR on EKS エンドポイントはお客様の VPC エンドポイントに解決されます。Amazon EMR on EKS のデフォルトのサービス名エンドポイントは、次の形式です。

```
emr-containers.Region.amazonaws.com
```

プライベート DNS ホスト名を有効にしない場合は、Amazon VPC が以下の形式で使用できる DNS エンドポイント名を提供します。

```
VPC_Endpoint_ID.emr-containers.Region.vpce.amazonaws.com
```

詳細については、「Amazon VPC ユーザーガイド」の「インターフェイス VPC エンドポイント (AWS PrivateLink)」を参照してください。Amazon EMR on EKS は、VPC 内のすべての API アクションへの呼び出しをサポートしています。

VPC エンドポイントポリシーを VPC エンドポイントにアタッチして、IAM プリンシパルのアクセスを制御できます。また、セキュリティグループを VPC エンドポイントに関連付けて、ネットワークトラフィックの送信元と送信先 (IP アドレスの範囲など) に基づいてインバウンドとアウトバウンドのアクセスを制御することもできます。詳細については、「VPC エンドポイントによるサービスのアクセスコントロール」を参照してください。

Amazon EMR on EKS の VPC エンドポイントポリシーの作成

Amazon EMR on EKS の Amazon VPC エンドポイントに対するポリシーを作成して、以下を指定することができます。

- アクションを実行できるプリンシパルまたは実行できないプリンシパル
- 実行可能なアクション
- アクションを実行できるリソース

詳細については、「Amazon VPC ユーザーガイド」の「VPC エンドポイントでサービスへのアクセスを制御する」を参照してください。

Example 指定された AWS アカウントからのすべてのアクセスを拒否する VPC エンドポイントポリシー

次の VPC エンドポイントポリシーは、 AWS アカウント *123456789012* がエンドポイントを使用して リソースにすべてアクセスすることを拒否します。

```
{
    "Statement": [
        {
            "Action": "*",
            "Effect": "Allow",
            "Resource": "*",
```

```
            "Principal": "*"
        },
        {
            "Action": "*",
            "Effect": "Deny",
            "Resource": "*",
            "Principal": {
                "AWS": [
                    "123456789012"
                ]
            }
        }
    ]
}
```

Example 指定した IAM プリンシパル (ユーザー) への VPC アクセスのみを許可する VPC エンドポイントポリシー

次の VPC エンドポイントポリシーでは、 AWS アカウント *123456789012* の IAM ユーザー *lijuan* にのみフルアクセスを許可します。他のすべての IAM プリンシパルは、エンドポイントを使用したアクセスを拒否されます。

```
{
    "Statement": [
        {
            "Action": "*",
            "Effect": "Allow",
            "Resource": "*",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::123456789012:user/lijuan"
                ]
            }
        }
    ]
}
```

Example 読み取り専用の Amazon EMR on EKS オペレーションを許可する VPC エンドポイントポリシー

次の VPC エンドポイントポリシーでは、 AWS アカウント *123456789012* のみが指定された Amazon EMR on EKS アクションを実行できます。

指定されたアクションは、Amazon EMR on EKS の読み取り専用アクセスに相当します。指定されたアカウントでは、VPC 上の他のすべてのアクションが拒否されます。他のすべてのアカウントは、すべてのアクセスを拒否されます。Amazon EMR on EKS アクションのリストについては、「Amazon EMR on EKS 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。

```
{
    "Statement": [
        {
            "Action": [
                "emr-containers:DescribeJobRun",
                "emr-containers:DescribeVirtualCluster",
                "emr-containers:ListJobRuns",
                "emr-containers:ListTagsForResource",
                "emr-containers:ListVirtualClusters"
            ],
            "Effect": "Allow",
            "Resource": "*",
            "Principal": {
                "AWS": [
                    "123456789012"
                ]
            }
        }
    ]
}
```

Example 指定した仮想クラスターへのアクセスを拒否する VPC エンドポイントポリシー

次の VPC エンドポイントポリシーでは、すべてのアカウントとプリンシパルにフルアクセスを許可しますが、クラスター ID *A1B2CD34EF5G* の仮想クラスターで実行されるアクションへの AWS アカウント *123456789012* のアクセスを拒否します。仮想クラスターのリソースレベルのアクセス許可をサポートしないその他の Amazon EMR on EKS アクションは、引き続き許可されます。Amazon EMR on EKS アクションのリストとそれに対応するリソースタイプについては、「AWS Identity and Access Management ユーザーガイド」の「Amazon EMR on EKS 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。

```
{
    "Statement": [
        {
            "Action": "*",
            "Effect": "Allow",
```

```
            "Resource": "*",
            "Principal": "*"
        },
        {
            "Action": "*",
            "Effect": "Deny",
            "Resource": "arn:aws:emr-containers:us-west-2:123456789012:/
virtualclusters/A1B2CD34EF5G",
            "Principal": {
                "AWS": [
                    "123456789012"
                ]
            }
        }
    ]
}
```

Amazon EMR on EKS のクロスアカウントアクセスを設定する

Amazon EMR on EKS のクロスアカウントアクセスを設定できます。クロスアカウントアクセスにより、ある AWS アカウントのユーザーは Amazon EMR on EKS ジョブを実行し、別の AWS アカウントに属する基盤となるデータにアクセスできます。

前提条件

Amazon EMR on EKS のクロスアカウントアクセスを設定するには、次の AWS アカウントにサインインしてタスクを完了します。

- `AccountA` - Amazon EMR を EKS クラスターの名前空間に登録して Amazon EMR on EKS 仮想クラスターを作成した AWS アカウント。
- `AccountB` - Amazon EMR on EKS ジョブにアクセスさせる Amazon S3 バケットまたは DynamoDB テーブルを含む AWS アカウント。

クロス AWS アカウントアクセスを設定する前に、アカウントで次の準備をしておく必要があります。

- ジョブを実行する、`AccountA` 内の Amazon EMR on EKS 仮想クラスター。

- 仮想クラスターでジョブを実行するために必要な許可を持つ、AccountA 内のジョブ実行ロール。詳細については、ジョブ実行ロールを作成するおよびAmazon EMR on EKS でのジョブ実行ロールの使用を参照してください。

クロスアカウントの Amazon S3 バケットまたは DynamoDB テーブルにアクセスする方法

Amazon EMR on EKS 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。

1. Amazon S3 バケット cross-account-bucket を AccountB に作成します。詳細については、「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、「DynamoDB テーブルの作成」を参照してください。
2. AccountB に、cross-account-bucket にアクセスできる Cross-Account-Role-B IAM ロールを作成します。
   1. IAM コンソールにサインインします。
   2. [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、「IAM ユーザーガイド」の「IAM ロールの作成」を参照してください。
   3. 以下のポリシーステートメントに示すように、cross-account-bucket S3 バケットにアクセスするための Cross-Account-Role-B の許可を指定する IAM ポリシーを作成します。IAM ポリシーを Cross-Account-Role-B にアタッチします。詳細については、「IAM ユーザーガイド」の「新しいポリシーの作成」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、クロスアカウントの DynamoDB テーブルにアクセスするための許可を指定する IAM ポリシーを作成します。IAM ポリシーを Cross-Account-Role-B にアタッチします。詳細については、「IAM ユーザーガイド」の「DynamoDB テーブルの作成」を参照してください。

DynamoDB テーブル CrossAccountTable にアクセスするためのポリシーを以下に示します。

```
{
    "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。

   1. ロールの信頼関係を設定するには、IAM コンソールで、ステップ 2 で作成したロール Cross-Account-Role-B の [信頼関係] タブを選択します。
   2. [信頼関係の編集] を選択します。
   3. 以下のポリシードキュメントを追加します。これにより、AccountA 内の Job-Execution-Role-A にこの Cross-Account-Role-B ロール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"
    }
  ]
```

```
}
```

4. Cross-Account-Role-B を引き受けるように、STS ロール引き受け許可を使用して AccountA 内で Job-Execution-Role-A を付与します。

   1. AWS アカウント の IAM コンソールでAccountA、 を選択しますJob-Execution-Role-A。
   2. 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"
        }
    ]
}
```

5. Amazon S3 アクセスの場合、Amazon EMR on EKS にジョブを送信するときに、以下の spark-submit パラメータ (spark conf) を設定します。

> **Note**
>
> デフォルトでは、EMRFS はジョブ実行ロールを使用して、ジョブから S3 バケットにアクセスします。ただし、customAWSCredentialsProvider が AssumeRoleAWSCredentialsProvider に設定されている場合、EMRFS は、Amazon S3 アクセスに Job-Execution-Role-A ではなく ASSUME_ROLE_CREDENTIALS_ROLE_ARN で指定された対応するロールを使用します。

- --conf spark.hadoop.fs.s3.customAWSCredentialsProvider=com.amazonaws.emr.AssumeRol
- --conf spark.kubernetes.driverEnv.ASSUME_ROLE_CREDENTIALS_ROLE_ARN=arn:aws:iam::Ac
  Cross-Account-Role-B \

- `--conf spark.executorEnv.ASSUME_ROLE_CREDENTIALS_ROLE_ARN=arn:aws:iam::AccountB:ro Cross-Account-Role-B \`

> **Note**
>
> ジョブの Spark 設定で、エグゼキューターとドライバーの両方の env に ASSUME_ROLE_CREDENTIALS_ROLE_ARN を設定する必要があります。

DynamoDB のクロスアカウントアクセスの場合、`--conf spark.dynamodb.customAWSCredentialsProvider=com.amazonaws.emr.AssumeRoleAWSCr` を設定する必要があります。

6. 次の例に示すように、クロスアカウントアクセスを使用して Amazon EMR on EKS ジョブを実行します。

```
aws emr-containers start-job-run \
--virtual-cluster-id 123456 \
--name myjob \
--execution-role-arn execution-role-arn \
--release-label emr-6.2.0-latest \
--job-driver '{"sparkSubmitJobDriver": {"entryPoint": "entryPoint_location",
 "entryPointArguments": ["arguments_list"], "sparkSubmitParameters": "--class
 <main_class> --conf spark.executor.instances=2 --conf spark.executor.memory=2G
 --conf spark.executor.cores=2 --conf spark.driver.cores=1 --conf
 spark.hadoop.fs.s3.customAWSCredentialsProvider=com.amazonaws.emr.AssumeRoleAWSCredentials
 --conf
 spark.kubernetes.driverEnv.ASSUME_ROLE_CREDENTIALS_ROLE_ARN=arn:aws:iam::AccountB:role/
Cross-Account-Role-B --conf
 spark.executorEnv.ASSUME_ROLE_CREDENTIALS_ROLE_ARN=arn:aws:iam::AccountB:role/
Cross-Account-Role-B"}} ' \
--configuration-overrides '{"applicationConfiguration": [{"classification":
 "spark-defaults", "properties": {"spark.driver.memory": "2G"}}],
 "monitoringConfiguration": {"cloudWatchMonitoringConfiguration":
 {"logGroupName": "log_group_name", "logStreamNamePrefix": "log_stream_prefix"},
 "persistentAppUI":"ENABLED",  "s3MonitoringConfiguration": {"logUri": "s3://
my_s3_log_location" }}}'
```

Amazon EMR on EKS リソースのタグ付け

Amazon EMR on EKS リソースを管理しやすくするために、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。このトピックでは、タグ機能の概要とタグの作成方法について説明します。

トピック

タグの基本

タグは、 AWS リソースに割り当てるラベルです。タグはそれぞれ、1 つのキーとオプションの 1 つの値で設定されており、どちらもお客様側が定義します。

タグを使用すると、リソースを目的、所有者、環境などの属性 AWS で分類できます。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 on EKS クラスターに一連のタグを定義して、各クラスターの所有者とスタックレベルを追跡できます。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。追加したタグに基づいてリソースを検索およびフィルタリングできます。

タグは自動的にはリソースに割り当てられません。タグを追加したら、いつでもタグキーと値は編集でき、タグはリソースからいつでも削除できます。リソースを削除すると、リソースのタグも削除されます。

タグには Amazon EMR on EKS に関する意味論的な意味はなく、完全に文字列として解釈されます。

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。タグキーを空の文字列にすることはできません。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。

AWS Identity and Access Management (IAM) を使用すると、タグを管理するアクセス許可を持つ AWS アカウントのユーザーを制御できます。

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

新規または既存の仮想クラスター、およびアクティブな状態のジョブ実行にタグ付けできます。ジョブ実行のアクティブな状態は、PENDING、SUBMITTED、RUNNING、CANCEL_PENDING などです。仮想クラスターのアクティブな状態は、RUNNING、TERMINATING、ARRESTED などです。詳細については、ジョブ実行状態および仮想クラスターの状態を参照してください。

仮想クラスターが終了すると、タグは消去され、アクセスできなくなります。

Amazon EMR on EKS API、、 AWS CLIまたは AWS SDK を使用している場合は、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することもできます。

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を行う場合、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。

次の表には、タグ付け可能な Amazon EMR on EKS リソースが示されています。

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 on EKS API、AWS CLIおよび AWS SDK)	作成時の API (作成時にタグを追加可能)
仮想クラスター	あり	いいえ。仮想クラスターに関連付けられたタグは、その仮想クラスターに送信されたジョブ実行には反映されません。	あり	CreateVirtualCluster

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 on EKS API、AWS CLIおよびAWS SDK)	作成時の API (作成時にタグを追加可能)
ジョブ実行	あり	なし	あり	StartJobRun

タグの制限

タグには以下のような基本制限があります。

- リソースあたりのタグの最大数 - 50 件
- 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 - UTF-8 の 128 Unicode 文字
- 値の最大長 - UTF-8 の 256 Unicode 文字
- タグ付けスキーマが複数の AWS サービスおよびリソースで使用されている場合は、他のサービスで許可される文字に制限がある場合があることに注意してください。一般的に使用が許可される文字は、UTF-8 で表現できる文字、数字、スペース、および +、-、=、.、_、:、/、@。
- タグのキーと値では、大文字と小文字が区別されます。
- 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。タグキーを空の文字列にすることはできません。
- キーまたは値のプレフィックスとして、`aws:`、`AWS:`、またはその大文字と小文字の組み合わせを変えたものは使用しないでください。これらは AWS 専用に予約されています。

AWS CLI と Amazon EMR on EKS API を使用してタグを操作する

次の AWS CLI コマンドまたは Amazon EMR on EKS API オペレーションを使用して、リソースのタグを追加、更新、一覧表示、削除します。

タスク	AWS CLI	API アクション
1 つ以上のタグを追加、または上書きします	タグリソース	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	タグなしリソース	UntagResource

以下の例では、 AWS CLIを使用して、リソースに対してタグ付けまたはタグ削除する方法を示しています。

例 1: 既存仮想クラスターへのタグ付け

次のコマンドは既存の仮想クラスターにタグ付けします。

```
aws emr-containers tag-resource --resource-arn resource_ARN --tags team=devs
```

例 2: 既存仮想クラスターでのタグ削除

次のコマンドは既存の仮想クラスターからタグを削除します。

```
aws emr-containers untag-resource --resource-arn resource_ARN --tag-keys tag_key
```

例 3: リソースのタグのリスト取得

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-containers list-tags-for-resource --resource-arn resource_ARN
```

Amazon EMR on EKS に関するトラブルシューティング

このセクションでは、Amazon EMR on EKS に関する問題をトラブルシューティングする方法について説明します。Amazon EMR に関する一般的な問題をトラブルシューティングする方法については、「Amazon EMR 管理ガイド」の「Troubleshoot a cluster」を参照してください。

トピック

- PersistentVolumeClaims (PVC) を使用するジョブのトラブルシューティング
- Amazon EMR on EKS 垂直自動スケーリングのトラブルシューティング
- Amazon EMR on EKS Spark オペレータのトラブルシューティング

PersistentVolumeClaims (PVC) を使用するジョブのトラブルシューティング

ジョブの PersistentVolumeClaims (PVC) を作成、リスト、または削除する必要があるにもかかわらず、デフォルトの Kubernetes ロール emr-containers に PVC 権限を追加しなかった場合、ジョブの送信が失敗します。PVC 権限がないと、emr-containers ロールでは Spark ドライバーや Spark クライアントに必要なロールを作成できません。エラーメッセージに従って、Spark ドライバーや Spark クライアントのロールに権限を追加するだけでは不十分です。emr-containers プライマリロールにも、必要な権限を含める必要があります。このセクションでは、必要な権限を emr-containers プライマリロールに追加する方法について説明します。

検証

emr-containers ロールに必要な権限があるかどうかを確認するには、NAMESPACE 変数に独自の値を設定してから、次のコマンドを実行します。

```
export NAMESPACE=YOUR_VALUE
kubectl describe role emr-containers -n ${NAMESPACE}
```

また、必要な権限が Spark ロールとクライアントロールにあるかどうかを確認するには、次のコマンドを実行します。

```
kubectl describe role emr-containers-role-spark-driver -n ${NAMESPACE}
kubectl describe role emr-containers-role-spark-client -n ${NAMESPACE}
```

必要な権限がない場合は、次のようにパッチを適用して続行します。

パッチ

1. 必要な権限がないジョブを現在実行している場合は、そのジョブを停止します。
2. 次のように、RBAC_Patch.py という名前でファイルを作成します。

```
import os
import subprocess as sp
import tempfile as temp
import json
import argparse
import uuid

def delete_if_exists(dictionary: dict, key: str):
    if dictionary.get(key, None) is not None:
        del dictionary[key]

def doTerminalCmd(cmd):
    with temp.TemporaryFile() as f:
        process = sp.Popen(cmd, stdout=f, stderr=f)
        process.wait()
        f.seek(0)
        msg = f.read().decode()
    return msg

def patchRole(roleName, namespace, extraRules, skipConfirmation=False):
    cmd = f"kubectl get role {roleName} -n {namespace} --output json".split(" ")
    msg = doTerminalCmd(cmd)
    if "(NotFound)" in msg and "Error" in msg:
        print(msg)
        return False
    role = json.loads(msg)
    rules = role["rules"]
    rulesToAssign = extraRules[::]
    passedRules = []
    for rule in rules:
        apiGroups = set(rule["apiGroups"])
        resources = set(rule["resources"])
        verbs = set(rule["verbs"])
        for extraRule in extraRules:
            passes = 0
            apiGroupsExtra = set(extraRule["apiGroups"])
```

```
            resourcesExtra = set(extraRule["resources"])
            verbsExtra = set(extraRule["verbs"])
            passes += len(apiGroupsExtra.intersection(apiGroups)) >=
len(apiGroupsExtra)
            passes += len(resourcesExtra.intersection(resources)) >=
len(resourcesExtra)
            passes += len(verbsExtra.intersection(verbs)) >= len(verbsExtra)
            if passes >= 3:
                if extraRule not in passedRules:
                    passedRules.append(extraRule)
                    if extraRule in rulesToAssign:
                        rulesToAssign.remove(extraRule)
                break
    prompt_text = "Apply Changes?"
    if len(rulesToAssign) == 0:
        print(f"The role {roleName} seems to already have the necessary
permissions!")
        prompt_text = "Proceed anyways?"
    for ruleToAssign in rulesToAssign:
        role["rules"].append(ruleToAssign)
    delete_if_exists(role, "creationTimestamp")
    delete_if_exists(role, "resourceVersion")
    delete_if_exists(role, "uid")
    new_role = json.dumps(role, indent=3)
    uid = uuid.uuid4()
    filename = f"Role-{roleName}-New_Permissions-{uid}-TemporaryFile.json"
    try:
        with open(filename, "w+") as f:
            f.write(new_role)
            f.flush()
        prompt = "y"
        if not skipConfirmation:
            prompt = input(
                doTerminalCmd(f"kubectl diff -f {filename}".split(" ")) +
f"\n{prompt_text} y/n: "
            ).lower().strip()
            while prompt != "y" and prompt != "n":
                prompt = input("Please make a valid selection. y/n:
").lower().strip()
        if prompt == "y":
            print(doTerminalCmd(f"kubectl apply -f {filename}".split(" ")))
    except Exception as e:
        print(e)
    os.remove(f"./{filename}")
```

```
if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument("-n", "--namespace",
                        help="Namespace of the Role. By default its the
 VirtualCluster's namespace",
                        required=True,
                        dest="namespace"
                        )

    parser.add_argument("-p", "--no-prompt",
                        help="Applies the patches without asking first",
                        dest="no_prompt",
                        default=False,
                        action="store_true"
                        )
    args = parser.parse_args()

    emrRoleRules = [
        {
            "apiGroups": [""],
            "resources": ["persistentvolumeclaims"],
            "verbs": ["list", "create", "delete", "patch"]
         }

    ]

    driverRoleRules = [
        {
            "apiGroups": [""],
            "resources": ["persistentvolumeclaims"],
            "verbs": ["list", "create", "delete", "patch"]
        },
        {
            "apiGroups": [""],
            "resources": ["services"],
            "verbs": ["get", "list", "describe", "create", "delete", "watch"]
        }
    ]

    clientRoleRules = [
        {
            "apiGroups": [""],
            "resources": ["persistentvolumeclaims"],
```

```
            "verbs": ["list", "create", "delete", "patch"]
        }
    ]

    patchRole("emr-containers", args.namespace, emrRoleRules, args.no_prompt)
    patchRole("emr-containers-role-spark-driver", args.namespace, driverRoleRules,
args.no_prompt)
    patchRole("emr-containers-role-spark-client", args.namespace, clientRoleRules,
args.no_prompt)
```

3. Python スクリプトを実行します。

```
python3 RBAC_Patch.py -n ${NAMESPACE}
```

4. kubectl による新しい権限と古い権限の違いが表示されます。y を押すと、ロールにパッチが適用されます。
5. 以下を実行して、3 つのロールに権限が追加されていることを確認します。

```
kubectl describe role -n ${NAMESPACE}
```

6. python スクリプトを実行します。

```
python3 RBAC_Patch.py -n ${NAMESPACE}
```

7. このコマンドを実行すると、kubectl による新しい権限と古い権限の違いが表示されます。y を押すと、ロールにパッチが適用されます。
8. 3 つのロールに権限が追加されていることを確認します。

```
kubectl describe role -n ${NAMESPACE}
```

9. ジョブを再度送信します。

手動パッチ

アプリケーションに必要な権限が PVC ルール以外に適用される場合、必要に応じて Amazon EMR 仮想クラスターに対する Kubernetes 権限を手動で追加できます。

> **Note**
>
> emr-containers ロールはプライマリロールです。つまり、基盤となるドライバーロールやクライアントロールを変更する場合は、その操作に必要なすべての権限をこのプライマリロールで提供する必要があります。

1. 次のコマンドを実行して、現在の権限を yaml ファイルにダウンロードします。

```
kubectl get role -n ${NAMESPACE} emr-containers -o yaml >> emr-containers-role-
patch.yaml
kubectl get role -n ${NAMESPACE} emr-containers-role-spark-driver -o yaml >> driver-
role-patch.yaml
kubectl get role -n ${NAMESPACE} emr-containers-role-spark-client -o yaml >> client-
role-patch.yaml
```

2. アプリケーションに必要な権限に基づいて、各ファイルを編集し、次のようなさらに別のルールを追加します。

   - emr-containers-role-patch.yaml

```
- apiGroups:
  - ""
  resources:
  - persistentvolumeclaims
  verbs:
  - list
  - create
  - delete
  - patch
```

   - driver-role-patch.yaml

```
- apiGroups:
  - ""
  resources:
  - persistentvolumeclaims
  verbs:
  - list
  - create
  - delete
  - patch
```

```
- apiGroups:
  - ""
  resources:
  - services
  verbs:
  - get
  - list
  - describe
  - create
  - delete
  - watch
```

- client-role-patch.yaml

```
- apiGroups:
  - ""
  resources:
  - persistentvolumeclaims
  verbs:
  - list
  - create
  - delete
  - patch
```

3. 次の属性をその値と共に削除します。更新を適用するために必要な操作です。
   - creationTimestamp
   - resourceVersion
   - uid
4. 最後に、パッチを実行します。

```
kubectl apply -f emr-containers-role-patch.yaml
kubectl apply -f driver-role-patch.yaml
kubectl apply -f client-role-patch.yaml
```

Amazon EMR on EKS 垂直自動スケーリングのトラブルシューティング

Operator Lifecycle Manager で Amazon EKS クラスターに Amazon EMR on EKS 垂直自動スケーリングオペレータをセットアップするときに問題が発生した場合は、次のセクションを参照してくださ

い。インストールを完了する手順などの詳細については、「Amazon EMR Spark ジョブで垂直的自動スケーリングを使用する」を参照してください。

403 Forbidden エラー

「Amazon EKS クラスターに Operator Lifecycle Manager (OLM) をインストールする」のステップに従って `olm status` コマンドを実行した結果、次のような `403 Forbidden` エラーが返された場合は、オペレータ用に Amazon ECR リポジトリの認証トークンを取得していない可能性があります。

この問題を解決するには、「Amazon EMR on EKS の垂直的自動スケーリング演算子をインストールする」のステップを繰り返して、トークンを取得します。次に、インストールをもう一度試します。

```
Error: FATA[0002] Failed to run bundle: pull bundle image: error pulling image IMAGE.
error resolving name : unexpected status code [manifests latest]: 403 Forbidden
```

Kubernetes 名前空間が見つからない

Amazon EKS クラスターに Amazon EMR on EKS 垂直自動スケーリングオペレータをセットアップするときに、次のような `namespaces not found` エラーが発生する場合があります。

```
FATA[0020] Failed to run bundle: create catalog: error creating catalog source:
 namespaces "NAME" not found.
```

指定した名前空間が存在しない場合、OLM は垂直自動スケーリングオペレータをインストールしません。この問題を解決するには、次のコマンドを使用して名前空間を作成します。次に、インストールをもう一度試します。

```
kubectl create namespace NAME
```

Docker 認証情報の保存中のエラー

垂直自動スケーリングをセットアップするには、Amazon EMR on EKS 垂直自動スケーリング関連の Docker イメージを認証して取得する必要があります。これを行う際に、Docker が実行されていないと、次のようなエラーが発生する場合があります。

```
aws ecr get-login-password \
```

```
 --region $REGION | docker login \
 --username AWS \
 --password-stdin $ACCOUNT_ID.dkr.ecr.$REGION.amazonaws.com

Error saving credentials: error storing credentials - err: exit status 1
out: 'Post "http://ipc/registry/credstore-updated": dial unix backend.sock: connect: no
 such file or directory'
```

この問題を解決するには、Docker が実行されていることを確認するか、Docker Desktop を開きます。次に、認証情報をもう一度保存してみます。

Amazon EMR on EKS Spark オペレータのトラブルシューティング

Amazon EMR on EKS Spark オペレータに関する問題が発生した場合は、次のセクションを参照してください。インストールを完了する手順などの詳細については、「Spark 演算子を使用して Spark ジョブを実行する」を参照してください。

Helm チャートのインストール時のエラー

「Spark 演算子をインストールする」のステップに従って Helm チャートをインストールまたは検証しようとした結果、次のような `INSTALLATION FAILED` エラーが返された場合は、オペレータ用に Amazon ECR リポジトリの認証トークンを取得していない可能性があります。

この問題を解決するには、「Spark 演算子をインストールする」のステップを繰り返して、Amazon ECR レジストリに照らして Helm クライアントを認証します。次に、インストール手順をもう一度試します。

```
Error: INSTALLATION FAILED: Kubernetes cluster unreachable: the server has asked for
 the client to provide credentials
```

UnsupportedFileSystemException: No FileSystem for scheme "s3"

スレッド「main」で次の例外が発生することがあります。

```
org.apache.hadoop.fs.UnsupportedFileSystemException: No FileSystem for scheme "s3"
```

このような場合は、`SparkApplication` 仕様に次の例外を追加します。

```
 hadoopConf:
```

```
    # EMRFS filesystem
    fs.s3.customAWSCredentialsProvider:
 com.amazonaws.auth.WebIdentityTokenCredentialsProvider
    fs.s3.impl: com.amazon.ws.emr.hadoop.fs.EmrFileSystem
    fs.AbstractFileSystem.s3.impl: org.apache.hadoop.fs.s3.EMRFSDelegate
    fs.s3.buffer.dir: /mnt/s3
    fs.s3.getObject.initialSocketTimeoutMilliseconds: "2000"
    mapreduce.fileoutputcommitter.algorithm.version.emr_internal_use_only.EmrFileSystem:
 "2"
    mapreduce.fileoutputcommitter.cleanup-
failures.ignored.emr_internal_use_only.EmrFileSystem: "true"
 sparkConf:
    # Required for EMR Runtime
    spark.driver.extraClassPath: /usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/hadoop-
aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/emr/emrfs/conf:/usr/share/aws/
emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/auxlib/*:/usr/share/aws/emr/security/conf:/
usr/share/aws/emr/security/lib/*:/usr/share/aws/hmclient/lib/aws-glue-datacatalog-
spark-client.jar:/usr/share/java/Hive-JSON-Serde/hive-openx-serde.jar:/usr/share/aws/
sagemaker-spark-sdk/lib/sagemaker-spark-sdk.jar:/home/hadoop/extrajars/*
    spark.driver.extraLibraryPath: /usr/lib/hadoop/lib/native:/usr/lib/hadoop-lzo/lib/
native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/lib/native
    spark.executor.extraClassPath: /usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/hadoop-
aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/emr/emrfs/conf:/usr/share/aws/
emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/auxlib/*:/usr/share/aws/emr/security/conf:/
usr/share/aws/emr/security/lib/*:/usr/share/aws/hmclient/lib/aws-glue-datacatalog-
spark-client.jar:/usr/share/java/Hive-JSON-Serde/hive-openx-serde.jar:/usr/share/aws/
sagemaker-spark-sdk/lib/sagemaker-spark-sdk.jar:/home/hadoop/extrajars/*
    spark.executor.extraLibraryPath: /usr/lib/hadoop/lib/native:/usr/lib/hadoop-lzo/lib/
native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/lib/native
```

Amazon EMR on EKS サービスエンドポイントとサービスクォータ

Amazon EMR on EKS のサービスエンドポイントおよび Service Quotas を以下に示します。 AWS サービスにプログラムで接続するには、 エンドポイントを使用します。標準 AWS エンドポイントに加えて、一部の AWS サービスは、選択したリージョンで FIPS エンドポイントを提供します。詳細については、AWS サービスエンドポイントを参照してください。サービスクォータ (制限とも呼ばれます) は、 AWS アカウントのサービスリソースまたはオペレーションの最大数です。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。

サービスエンドポイント

AWS リージョン 名前	コード	エンドポイント	プロトコル
米国東部 (バージニア北部)	us-east-1	emr-containers.us-east-1.amazonaws.com	HTTPS
米国東部 (オハイオ)	us-east-2	emr-containers.us-east-2.amazonaws.com	HTTPS
米国西部 (北カリフォルニア)	us-west-1	emr-containers.us-west-1.amazonaws.com	HTTPS
米国西部 (オレゴン)	us-west-2	emr-containers.us-west-2.amazonaws.com	HTTPS
アジアパシフィック (東京)	ap-northeast-1	emr-containers.ap-northeast-1.amazonaws.com	HTTPS
アジアパシフィック (ソウル)	ap-northeast-2	emr-containers.ap-northeast-2.amazonaws.com	HTTPS
アジアパシフィック (大阪)	ap-northeast-3	emr-containers.ap-northeast-3.amazonaws.com	HTTPS
アジアパシフィック (ムンバイ)	ap-south-1	emr-containers.ap-south-1.amazonaws.com	HTTPS

AWS リージョン 名前	コード	エンドポイント	プロトコル
アジアパシフィック (ハイデラバード)	ap-south-2	emr-containers.ap-south-2.amazonaws.com	HTTPS
アジアパシフィック (シンガポール)	ap-southeast-1	emr-containers.ap-southeast-1.amazonaws.com	HTTPS
アジアパシフィック (シドニー)	ap-southeast-2	emr-containers.ap-southeast-2.amazonaws.com	HTTPS
アジアパシフィック (ジャカルタ)	ap-southeast-3	emr-containers.ap-southeast-3.amazonaws.com	HTTPS
アジアパシフィック (香港)	ap-east-1	emr-containers.ap-east-1.amazonaws.com	HTTPS
アフリカ (ケープタウン)	af-south-1	emr-containers.af-south-1.amazonaws.com	HTTPS
カナダ (中部)	ca-central-1	emr-containers.ca-central-1.amazonaws.com	HTTPS
中国 (寧夏)	cn-northwest-1	emr-containers.cn-northwest-1.amazonaws.com.cn	HTTPS
中国 (北京)	cn-north-1	emr-containers.cn-north-1.amazonaws.com.cn	HTTPS
欧州 (フランクフルト)	eu-central-1	emr-containers.eu-central-1.amazonaws.com	HTTPS
欧州 (チューリッヒ)	eu-central-2	emr-containers.eu-central-2.amazonaws.com	HTTPS
欧州 (アイルランド)	eu-west-1	emr-containers.eu-west-1.amazonaws.com	HTTPS

AWS リージョン 名前	コード	エンドポイント	プロトコル
欧州 (ロンドン)	eu-west-2	emr-containers.eu-west-2.amazonaws.com	HTTPS
欧州 (パリ)	eu-west-3	emr-containers.eu-west-3.amazonaws.com	HTTPS
欧州 (ストックホルム)	eu-north-1	emr-containers.eu-north-1.amazonaws.com	HTTPS
欧州 (ミラノ)	eu-south-1	emr-containers.eu-south-1.amazonaws.com	HTTPS
欧州 (スペイン)	eu-south-2	emr-containers.eu-south-2.amazonaws.com	HTTPS
イスラエル (テルアビブ)	il-central-1	emr-containers.il-central-1.amazonaws.com	HTTPS
南米 (サンパウロ)	sa-east-1	emr-containers.sa-east-1.amazonaws.com	HTTPS
中東 (UAE)	me-central-1	emr-containers.me-central-1.amazonaws.com	HTTPS
中東 (バーレーン)	me-south-1	emr-containers.me-south-1.amazonaws.com	HTTPS
AWS GovCloud (米国東部)	us-gov-east-1	emr-containers.us-gov-east-1.amazonaws.com	HTTPS
AWS GovCloud (米国西部)	us-gov-west-1	emr-containers.us-gov-west-1.amazonaws.com	HTTPS

Service Quotas

Amazon EMR on EKS は、リージョンごとに AWS アカウントごとに次の API リクエストを調整します。スロットルの適用方法の詳細については、Amazon EC2 API リファレンスの API リクエスト

のスロットリング を参照してください。 AWS アカウントの API スロットリングクォータの引き上げをリクエストできます。

API アクション	バケットの最大容量	バケットの補充レート/秒
CancelJobRun	25	1
CreateManagedEndpoint	25	1
CreateVirtualCluster	25	1
DeleteManagedEndpoint	25	1
DeleteVirtualCluster	25	1
DescribeJobRun	100	20
DescribeManagedEndpoint	100	5
DescribeVirtualCluster	100	5
ListJobRun	100	5
ListManagedEndpoint	25	1
ListVirtualCluster	100	5
StartJobRun	25	1
At the AWS account level, the bucket maximum capacity and refill rate for the sum of all API actions listed in this table	200	20

Amazon EMR on EKS リリース

Amazon EMR リリースは、ビッグデータエコシステムの一連のオープンソースアプリケーションです。各リリースは異なるビッグデータアプリケーション、コンポーネント、および機能で構成され、ジョブを行うときに Amazon EMR on EKS でデプロイして設定することを選択します。

Amazon EMR リリース 5.32.0 および 6.2.0 以降で、Amazon EMR on EKS 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に、サポートされているリリースバージョンを指定する必要があります。

Amazon EMR on EKSでは、次の形式のリリースラベルを使用します : `emr-x.x.x-latest` または `emr-x.x.x-yyyymmdd` 特定のリリース日を指定します。例えば、`emr-7.6.0-latest`、`emr-7.6.0-20210129` です。`-latest` サフィックスを使用すると、Amazon EMR のバージョンに常に最新のセキュリティアップデートが含まれるようになります。

> **Note**
>
> Amazon EMR on EKS と EC2 で実行されている Amazon EMR の比較については、 AWS ウェブサイトの「Amazon EMR に関するFAQs」を参照してください。

トピック

- Amazon EMR on EKS 6.11.0 リリース
- Amazon EMR on EKS 6.10.0 リリース
- Amazon EMR on EKS 6.9.0 リリース
- Amazon EMR on EKS 6.8.0 リリース
- Amazon EMR on EKS 6.7.0 リリース
- Amazon EMR on EKS 6.6.0 リリース
- Amazon EMR on EKS 6.5.0 リリース
- Amazon EMR on EKS 6.4.0 リリース
- Amazon EMR on EKS 6.3.0 リリース
- Amazon EMR on EKS 6.2.0 リリース
- Amazon EMR on EKS 5.36.0 リリース
- Amazon EMR on EKS 5.35.0 リリース
- Amazon EMR on EKS 5.34.0 リリース
- Amazon EMR on EKS 5.33.0 リリース
- Amazon EMR on EKS 5.32.0 リリース

Amazon EMR on EKS 7.7.0 リリース

このページでは、Amazon EMR on EKS デプロイに固有の Amazon EMR の新しい機能と更新された機能について説明します。Amazon EC2 で実行されている Amazon EMR と Amazon EMR 7.7.0 リリース全般の詳細については、「Amazon EMR リリースガイド」の「Amazon EMR 7.7.0」を参照してください。

Amazon EMR on EKS 7.7 リリース

Amazon EMR on EKS では、次の Amazon EMR 7.7.0 リリースを利用できます。特定の emr-7.7.0-XXXX リリースを選択すると、関連するコンテナイメージタグなどの詳細が表示されます。

Flink releases

Flink アプリケーションを実行する場合、Amazon EMR on EKS では次の Amazon EMR 7.7.0 リリースを使用できます。

- emr-7.7.0-flink-latest

- emr-7.7.0-flink-20250131

Spark releases

Spark アプリケーションを実行する場合、Amazon EMR on EKS では次の Amazon EMR 7.7.0 リリースを使用できます。

- emr-7.7.0-最新
- emr-7.7.0-20250131
- emr-7.7.0-spark-rapids-latest
- emr-7.7.0-spark-rapids-20250131
- emr-7.7.0-java11-latest
- emr-7.7.0-java11-20250131
- emr-7.7.0-java8-latest
- emr-7.7.0-java8-20250131
- emr-7.7.0-spark-rapids-java8-latest
- emr-7.7.0-spark-rapids-java8-20250131
- notebook-spark/emr-7.7.0-latest
- notebook-spark/emr-7.7.0-20250131
- notebook-spark/emr-7.7.0-spark-rapids-latest
- notebook-spark/emr-7.7.0-spark-rapids-20250131
- notebook-spark/emr-7.7.0-java11-latest
- notebook-spark/emr-7.7.0-java11-20250131
- notebook-spark/emr-7.7.0-java8-latest
- notebook-spark/emr-7.7.0-java8-20250131
- notebook-spark/emr-7.7.0-spark-rapids-java8-latest
- notebook-spark/emr-7.7.0-spark-rapids-java8-20250131
- notebook-python/emr-7.7.0-latest
- notebook-python/emr-7.7.0-20250131
- notebook-python/emr-7.7.0-spark-rapids-latest
- notebook-python/emr-7.7.0-spark-rapids-20250131
- notebook-python/emr-7.7.0-java11-latest

- notebook-python/emr-7.7.0-java11-20250131
- notebook-python/emr-7.7.0-java8-latest
- notebook-python/emr-7.7.0-java8-20250131
- notebook-python/emr-7.7.0-spark-rapids-java8-latest
- notebook-python/emr-7.7.0-spark-rapids-java8-20250131
- livy/emr-7.7.0-latest
- livy/emr-7.7.0-20250131
- livy/emr-7.7.0-java11-latest
- livy/emr-7.7.0-java11-20250131
- livy/emr-7.7.0-java8-latest
- livy/emr-7.7.0-java8-20250131

リリースノート

Amazon EMR on EKS 7.7.0 のリリースノート

- サポートされているアプリケーション - AWS SDK for Java 2.29.25 and 1.12.779, Apache Spark 3.5.3-amzn-0, Apache Hudi 0.15.0-amzn-3, Apache Iceberg 1.6.1-amzn-2, Delta 3.2.1-amzn-1, Apache Spark RAPIDS 24.10.1-amzn-0, Jupyter Enterprise Gateway 2.6.0, Apache Flink 1.20.0-amzn-0, Flink Operator 1.10.0-amzn-0
- サポートされているコンポーネント - `emr-ddb`、`emr-goodies`、、`emr-s3-select`、`emrfshadoop-client`、、`hudi`、`hudi-sparkiceberg`、`spark-kubernetes`。
- サポートされている設定分類

  StartJobRun API と CreateManagedEndpoint API で使用する場合:

分類	説明
`core-site`	`core-site.xml` Hadoop ファイル内の値を変更します。
`emrfs-site`	EMRFS の設定を変更します。
`spark-metrics`	`metrics.properties` Spark ファイル内の値を変更します。

分類	説明
spark-defaults	spark-defaults.conf Spark ファイル内の値を変更します。
spark-env	Spark 環境の値を変更します。
spark-hive-site	hive-site.xml Spark ファイル内の値を変更します。
spark-log4j2	log4j2.properties Spark ファイル内の値を変更します。
emr-job-submitter	ジョブ送信者ポッドの設定。

特に CreateManagedEndpoint API で使用する場合:

分類	説明
jeg-config	Jupyter Enterprise Gateway jupyter_enterprise_gateway_config.py ファイルの値を変更します。
jupyter-kernel-overrides	Jupyter カーネル仕様ファイル内のカーネルイメージの値を変更します。

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。これらは多くの場合、spark-hive-site.xml などのアプリケーションの構成 XML ファイルに対応します。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。

変更

Amazon EMR on EKS の 7.7.0 リリースには、次の変更が含まれています。

- EMR 7.7.0 で使用されている Iceberg バージョンは、Java 8 をサポートしなくなりました。さらに、Iceberg は および の Java 8 イメージから除外emr-7.7.0-java8-latestされますemr-7.7.0-spark-rapids-java8-latest。

emr-7.7.0-最新

リリースノート : 現在 emr-7.7.0-latest が指しているのは emr-7.7.0-20250131 です。

リージョン: emr-7.7.0-latest は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-7.7.0:latest

emr-7.7.0-20250131

リリースノート: は 2025 年 2 月にリリースemr-7.7.0-20250131されました。これは Amazon EMR 7.7.0 (Spark) の初期リリースです。

リージョン: emr-emr-7.7.0-20250131 は、Amazon EMR で EKS でサポートされているすべてのリージョンで使用可能です。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-7.7.0-20250131

emr-7.7.0-flink-latest

リリースノート: emr-7.7.0-flink-latest は現在 を指しています emr-7.7.0-flink-20250131

リージョン: emr-7.7.0-flink-latest は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-7.7.0-flink:latest

emr-7.7.0-flink-20250131

リリースノート: は 2025 年 2 月にリリース7.7.0-flink-20250131されました。これは Amazon EMR 7.7.0 (Flink) の初期リリースです。

リージョン: emr-7.7.0-flink-20250131 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-7.7.0-flink:20250131

Amazon EMR on EKS 7.6.0 リリース

このページでは、Amazon EMR on EKS デプロイに固有の Amazon EMR の新しい機能と更新された機能について説明します。Amazon EC2 で実行されている Amazon EMR と Amazon EMR 7.6.0 リリース全般の詳細については、「Amazon EMR リリースガイド」の「Amazon EMR 7.6.0」を参照してください。

Amazon EMR on EKS 7.6 リリース

Amazon EMR on EKS では、次の Amazon EMR 7.6.0 リリースを利用できます。特定の emr-7.6.0-XXXX リリースを選択すると、関連するコンテナイメージタグなどの詳細が表示されます。

Flink releases

Flink アプリケーションを実行する場合、Amazon EMR on EKS では次の Amazon EMR 7.6.0 リリースを使用できます。

- emr-7.6.0-flink-latest
- emr-7.6.0-flink-20241213

Spark releases

Spark アプリケーションを実行する場合、Amazon EMR on EKS では次の Amazon EMR 7.6.0 リリースを使用できます。

- emr-7.6.0-最新
- emr-7.6.0-20241213
- emr-7.6.0-spark-rapids-latest
- emr-7.6.0-spark-rapids-20241213
- emr-7.6.0-java11-latest
- emr-7.6.0-java11-20241213
- emr-7.6.0-java8-latest
- emr-7.6.0-java8-20241213
- emr-7.6.0-spark-rapids-java8-latest
- emr-7.6.0-spark-rapids-java8-20241213
- notebook-spark/emr-7.6.0-latest

- notebook-spark/emr-7.6.0-20241213
- notebook-spark/emr-7.6.0-spark-rapids-latest
- notebook-spark/emr-7.6.0-spark-rapids-20241213
- notebook-spark/emr-7.6.0-java11-latest
- notebook-spark/emr-7.6.0-java11-20241213
- notebook-spark/emr-7.6.0-java8-latest
- notebook-spark/emr-7.6.0-java8-20241213
- notebook-spark/emr-7.6.0-spark-rapids-java8-latest
- notebook-spark/emr-7.6.0-spark-rapids-java8-20241213
- notebook-python/emr-7.6.0-latest
- notebook-python/emr-7.6.0-20241213
- notebook-python/emr-7.6.0-spark-rapids-latest
- notebook-python/emr-7.6.0-spark-rapids-20241213
- notebook-python/emr-7.6.0-java11-latest
- notebook-python/emr-7.6.0-java11-20241213
- notebook-python/emr-7.6.0-java8-latest
- notebook-python/emr-7.6.0-java8-20241213
- notebook-python/emr-7.6.0-spark-rapids-java8-latest
- notebook-python/emr-7.6.0-spark-rapids-java8-20241213
- livy/emr-7.6.0-latest
- livy/emr-7.6.0-20241213
- livy/emr-7.6.0-java11-latest
- livy/emr-7.6.0-java11-20241213
- livy/emr-7.6.0-java8-latest
- livy/emr-7.6.0-java8-20241213

リリースノート

Amazon EMR on EKS 7.6.0 のリリースノート

- サポートされているアプリケーション - AWS SDK for Java 2.29.25 and 1.12.779, Apache Spark 3.5.3-amzn-0, Apache Hudi 0.15.0-amzn-3, Apache Iceberg 1.6.1-amzn-2, Delta 3.2.1-amzn-1,

Apache Spark RAPIDS 24.10.1-amzn-0, Jupyter Enterprise Gateway 2.6.0, Apache Flink 1.20.0-amzn-0, Flink Operator 1.10.0-amzn-0

- サポートされているコンポーネント - `aws-sagemaker-spark-sdk`、`emr-ddb`、`emr-goodies`、`emr-s3-select`、`emrfs`、`hadoop-client`、`hudi`、`hudi-spark`、`iceberg`、`spark-kubernetes`。
- サポートされている設定分類

  StartJobRun API と CreateManagedEndpoint API で使用する場合:

分類	説明
`core-site`	`core-site.xml` Hadoop ファイル内の値を変更します。
`emrfs-site`	EMRFS の設定を変更します。
`spark-metrics`	`metrics.properties` Spark ファイル内の値を変更します。
`spark-defaults`	`spark-defaults.conf` Spark ファイル内の値を変更します。
`spark-env`	Spark 環境の値を変更します。
`spark-hive-site`	`hive-site.xml` Spark ファイル内の値を変更します。
`spark-log4j2`	`log4j2.properties` Spark ファイル内の値を変更します。
`emr-job-submitter`	ジョブ送信者ポッドの設定。

  特に CreateManagedEndpoint API で使用する場合:

分類	説明
jeg-config	Jupyter Enterprise Gateway jupyter_enterprise_gateway_config.py ファイルの値を変更します。
jupyter-kernel-overrides	Jupyter カーネル仕様ファイル内のカーネルイメージの値を変更します。

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。これらは多くの場合、spark-hive-site.xml などのアプリケーションの構成 XML ファイルに対応します。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。

注目すべき機能

Amazon EMR on EKS の 7.6.0 リリースには、次の機能が含まれています。

- Apache Spark Operator の設定サポートのモニタリング – 設定のモニタリングにより、Spark アプリケーションとオペレーターログの Amazon S3 または Amazon CloudWatch へのログアーカイブを簡単に設定できます。1 つまたは両方を選択できます。これにより、Spark オペレータポッド、ドライバー、エグゼキュターポッドにログエージェントサイドカーが追加され、その後、これらのコンポーネントのログが設定されたシンクに転送されます。詳細については、「モニタリング設定を使用して Spark Kubernetes オペレータと Spark ジョブをモニタリングする」を参照してください。

変更

Amazon EMR on EKS の 7.6.0 リリースには、次の変更が含まれています。

- リリースに変更はありません。

emr-7.6.0-最新

リリースノート：現在 emr-7.6.0-latest が指しているのは emr-7.6.0-20241213 です。

リージョン: emr-7.6.0-latest は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-7.6.0:latest

emr-7.6.0-20241213

リリースノート: は 2024 年 1 月にリリース7.6.0-20241213されました。これは Amazon EMR 7.6.0 (Spark) の初期リリースです。

リージョン: emr-7.6.0-20241213 は、Amazon EMR で EKS でサポートされているすべてのリージョンで使用可能です。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-7.6.0:20241213

emr-7.6.0-flink-latest

リリースノート: emr-7.6.0-flink-latest は現在 を指しています emr-7.6.0-flink-20241213

リージョン: emr-7.6.0-flink-latest は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-7.6.0-flink:latest

emr-7.6.0-flink-20241213

リリースノート: は 2024 年 1 月にリリース7.6.0-flink-20241213されました。これは Amazon EMR 7.6.0 (Flink) の初期リリースです。

リージョン: emr-7.6.0-flink-20241213 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-7.6.0-flink:20241213

Amazon EMR on EKS 7.5.0 リリース

このページでは、Amazon EMR on EKS デプロイに固有の Amazon EMR の新しい機能と更新された機能について説明します。Amazon EC2 で実行されている Amazon EMR と Amazon EMR 7.5.0 リリース全般の詳細については、「Amazon EMR リリースガイド」の「Amazon EMR 7.5.0」を参照してください。

Amazon EMR on EKS 7.5 リリース

Amazon EMR on EKS では、次の Amazon EMR 7.5.0 リリースを利用できます。特定の emr-7.5.0-XXXX リリースを選択すると、関連するコンテナイメージタグなどの詳細が表示されます。

リリースノート

Amazon EMR on EKS 7.5.0 のリリースノート

- サポートされているアプリケーション - AWS SDK for Java 2.28.8 and 1.12.772, Apache Spark 3.5.2-amzn-1, Apache Hudi 0.15.0-amzn-1, Apache Iceberg 1.6.1-amzn-0, Delta 3.2.0-amzn-1, Apache Spark RAPIDS 24.08.1-amzn-1, Jupyter Enterprise Gateway 2.6.0, Apache Flink 1.19.1-amzn-1, Flink Operator 1.9.0-amzn-0
- サポートされているコンポーネント - `aws-sagemaker-spark-sdk`、`emr-ddb`、`emr-goodies`、`emr-s3-select`、`emrfs`、`hadoop-client`、`hudi`、`hudi-spark`、`iceberg`、`spark-kubernetes`。

Amazon EMR on EKS 7.4.0 リリース

このページでは、Amazon EMR on EKS デプロイに固有の Amazon EMR の新しい機能と更新された機能について説明します。Amazon EC2 で実行されている Amazon EMR と Amazon EMR 7.4.0 リリース全般の詳細については、「Amazon EMR リリースガイド」の「Amazon EMR 7.4.0」を参照してください。

Amazon EMR on EKS 7.4 リリース

Amazon EMR on EKS では、次の Amazon EMR 7.4.0 リリースを利用できます。特定の emr-7.4.0-XXXX リリースを選択すると、関連するコンテナイメージタグなどの詳細が表示されます。

リリースノート

Amazon EMR on EKS 7.4.0 のリリースノート

- サポートされているアプリケーション - AWS SDK for Java 2.25.70 and 1.12.772, Apache Spark 3.5.2-amzn-0, Apache Hudi 0.15.0-amzn-1, Apache Iceberg 1.6.1-amzn-0, Delta 3.2.0-amzn-1, Apache Spark RAPIDS 24.08.1-amzn-0, Jupyter Enterprise Gateway 2.6.0, Apache Flink 1.19.1-amzn-0, Flink Operator 1.9.0-amzn-1
- サポートされているコンポーネント - `aws-sagemaker-spark-sdk`、`emr-ddb`、`emr-goodies`、`emr-s3-select`、`emrfs`、`hadoop-client`、`hudi`、`hudi-spark`、`iceberg`、`spark-kubernetes`。

Amazon EMR on EKS 7.3.0 リリース

このページでは、Amazon EMR on EKS デプロイに固有の Amazon EMR の新しい機能と更新された機能について説明します。Amazon EC2 で実行されている Amazon EMR の詳細と Amazon EMR 7.3.0 リリース全般の詳細については、「Amazon EMR リリースガイド」の「Amazon EMR 7.3.0」を参照してください。

Amazon EMR on EKS 7.3 リリース

Amazon EMR on EKS では、次の Amazon EMR 7.3.0 リリースを利用できます。特定の emr-7.3.0-XXXX リリースを選択すると、関連するコンテナイメージタグなどの詳細が表示されます。

Flink releases

Flink アプリケーションを実行する場合は、Amazon EMR on EKS で次の Amazon EMR 7.3.0 リリースを利用できます。

- emr-7.3.0-flink-latest
- emr-7.3.0-flink-29240920

Spark releases

Spark アプリケーションを実行する場合は、Amazon EMR on EKS で次の Amazon EMR 7.3.0 リリースを利用できます。

- emr-7.3.0-latest
- emr-7.3.0-29240920
- emr-7.3.0-spark-rapids-latest
- emr-7.3.0-spark-rapids-29240920

- emr-7.3.0-java11-latest
- emr-7.3.0-java11-29240920
- emr-7.3.0-java8-latest
- emr-7.3.0-java8-29240920
- emr-7.3.0-spark-rapids-java8-latest
- emr-7.3.0-spark-rapids-java8-29240920
- notebook-spark/emr-7.3.0-latest
- notebook-spark/emr-7.3.0-29240920
- notebook-spark/emr-7.3.0-spark-rapids-latest
- notebook-spark/emr-7.3.0-spark-rapids-29240920
- notebook-spark/emr-7.3.0-java11-latest
- notebook-spark/emr-7.3.0-java11-29240920
- notebook-spark/emr-7.3.0-java8-latest
- notebook-spark/emr-7.3.0-java8-29240920
- notebook-spark/emr-7.3.0-spark-rapids-java8-latest
- notebook-spark/emr-7.3.0-spark-rapids-java8-29240920
- notebook-python/emr-7.3.0-latest
- notebook-python/emr-7.3.0-29240920
- notebook-python/emr-7.3.0-spark-rapids-latest
- notebook-python/emr-7.3.0-spark-rapids-29240920
- notebook-python/emr-7.3.0-java11-latest
- notebook-python/emr-7.3.0-java11-29240920
- notebook-python/emr-7.3.0-java8-latest
- notebook-python/emr-7.3.0-java8-29240920
- notebook-python/emr-7.3.0-spark-rapids-java8-latest
- notebook-python/emr-7.3.0-spark-rapids-java8-29240920
- livy/emr-7.3.0-latest
- livy/emr-7.3.0-29240920
- livy/emr-7.3.0-java11-latest
- livy/emr-7.3.0-java11-29240920

- livy/emr-7.3.0-java8-latest
- livy/emr-7.3.0-java8-29240920

リリースノート

Amazon EMR on EKS 7.3.0 のリリースノート

- サポートされているアプリケーション - AWS SDK for Java 2.25.70 and 1.12.747, Apache Spark 3.5.1-amzn-1, Apache Hudi 0.15.0-amzn-0, Apache Iceberg 1.5.2-amzn-0, Delta 3.2.0-amzn-0, Apache Spark RAPIDS 24.06.1-amzn-0, Jupyter Enterprise Gateway 2.6.0, Apache Flink 1.18.1-amzn-2, Flink Operator 1.9.0-amzn-0
- サポートされているコンポーネント - `aws-sagemaker-spark-sdk`、`emr-ddb`、`emr-goodies`、`emr-s3-select`、`emrfs`、`hadoop-client`、`hudi`、`hudi-spark`、`iceberg`、`spark-kubernetes`。
- サポートされている設定分類

  StartJobRun API と CreateManagedEndpoint API で使用する場合:

分類	説明
`core-site`	`core-site.xml` Hadoop ファイル内の値を変更します。
`emrfs-site`	EMRFS の設定を変更します。
`spark-metrics`	`metrics.properties` Spark ファイル内の値を変更します。
`spark-defaults`	`spark-defaults.conf` Spark ファイル内の値を変更します。
`spark-env`	Spark 環境の値を変更します。
`spark-hive-site`	`hive-site.xml` Spark ファイル内の値を変更します。
`spark-log4j2`	`log4j2.properties` Spark ファイル内の値を変更します。

分類	説明
emr-job-submitter	ジョブ送信者ポッドの設定。

特に CreateManagedEndpoint API で使用する場合:

分類	説明
jeg-config	Jupyter Enterprise Gateway jupyter_enterprise_gateway_config.py ファイルの値を変更します。
jupyter-kernel-overrides	Jupyter カーネル仕様ファイル内のカーネルイメージの値を変更します。

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。これらは多くの場合、spark-hive-site.xml などのアプリケーションの構成 XML ファイルに対応します。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。

注目すべき機能

Amazon EMR on EKS の 7.3.0 リリースには、以下の機能が含まれています。

- アプリケーションのアップグレード – Amazon EMR on EKS に Flink Operator 1.9.0 が追加されました。Flink Kubernetes では、他の機能に加えて、オートスケーラーの CPU クォータとメモリクォータを設定できるようになりました。
- Apache Iceberg が Apache Flink をサポート – Apache Iceberg はオープンソースの高性能フォーマットの巨大な分析テーブルです。Amazon EMR 7.3.0 以降では、Amazon EMR on EKS で Apache Flink を実行するときに Apache Iceberg テーブルを使用できます。詳細については、Amazon EMR on EKS で「Amazon EMR on EKS での Apache Iceberg の使用」を参照してください。
- Delta Lake が Apache Flink をサポート - Delta Lake は、レイクハウスアーキテクチャのストレージレイヤーフレームワークであり、一般的に、Amazon S3 上に構築されます。Amazon EMR 7.3.0 以降では、Amazon EMR on EKS で Apache Flink を実行するときに Delta テーブルを使用

できます。詳細については、「Amazon EMR on EKS での Delta Lake の使用」を参照してください。

変更

Amazon EMR on EKS の 7.3.0 リリースでは、以下の変更が行われています。

- Amazon EMR on EKS 7.3.0 以降では、Apache Flink がデフォルトで Java 17 ランタイムを使用するようになりました。

emr-7.3.0-latest

リリースノート : 現在 `emr-7.3.0-latest` が指しているのは `emr-7.3.0-29240920` です。

リージョン: `emr-7.3.0-latest` は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: `emr-7.3.0:latest`

emr-7.3.0-29240920

リリースノート: `7.3.0-29240920` は、2023 年 12 月にリリースされました。これは Amazon EMR 7.3.0 (Spark) の初期リリースです。

リージョン: `emr-7.3.0-29240920` は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: `emr-7.3.0:29240920`

emr-7.3.0-flink-latest

リリースノート : 現在 `emr-7.3.0-flink-latest` が指しているのは `emr-7.3.0-flink-29240920` です。

リージョン: `emr-7.3.0-flink-latest` は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: `emr-7.3.0-flink:latest`

emr-7.3.0-flink-29240920

リリースノート: `7.3.0-flink-29240920` は、2023 年 12 月にリリースされました。これは Amazon EMR 7.3.0 (Flink) の初期リリースです。

リージョン: `emr-7.3.0-flink-29240920` は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: `emr-7.3.0-flink:29240920`

Amazon EMR on EKS 7.2.0 リリース

このページでは、Amazon EMR on EKS デプロイに固有の Amazon EMR の新しい機能と更新された機能について説明します。Amazon EC2 で実行されている Amazon EMR の詳細と Amazon EMR 7.2.0 リリース全般の詳細については、「Amazon EMR リリースガイド」の「Amazon EMR 7.2.0」を参照してください。

Amazon EMR on EKS 7.2 リリース

Amazon EMR on EKS では、次の Amazon EMR 7.2.0 リリースが利用可能です。特定の emr-7.2.0-XXXX リリースを選択すると、関連するコンテナイメージタグなどの詳細が表示されます。

Flink releases

Flink アプリケーションを実行する場合は、Amazon EMR on EKS で次の Amazon EMR 7.2.0 リリースを利用できます。

- emr-7.2.0-flink-latest
- emr-7.2.0-flink-20240610

Spark releases

Spark アプリケーションを実行する場合は、Amazon EMR on EKS で次の Amazon EMR 7.2.0 リリースを利用できます。

- emr-7.2.0-latest
- emr-7.2.0-20240610

- emr-7.2.0-spark-rapids-latest
- emr-7.2.0-spark-rapids-20240610
- emr-7.2.0-java11-latest
- emr-7.2.0-java11-20240610
- emr-7.2.0-java8-latest
- emr-7.2.0-java8-20240610
- emr-7.2.0-spark-rapids-java8-latest
- emr-7.2.0-spark-rapids-java8-20240610
- notebook-spark/emr-7.2.0-latest
- notebook-spark/emr-7.2.0-20240610
- notebook-spark/emr-7.2.0-spark-rapids-latest
- notebook-spark/emr-7.2.0-spark-rapids-20240610
- notebook-spark/emr-7.2.0-java11-latest
- notebook-spark/emr-7.2.0-java11-20240610
- notebook-spark/emr-7.2.0-java8-latest
- notebook-spark/emr-7.2.0-java8-20240610
- notebook-spark/emr-7.2.0-spark-rapids-java8-latest
- notebook-spark/emr-7.2.0-spark-rapids-java8-20240610
- notebook-python/emr-7.2.0-latest
- notebook-python/emr-7.2.0-20240610
- notebook-python/emr-7.2.0-spark-rapids-latest
- notebook-python/emr-7.2.0-spark-rapids-20240610
- notebook-python/emr-7.2.0-java11-latest
- notebook-python/emr-7.2.0-java11-20240610
- notebook-python/emr-7.2.0-java8-latest
- notebook-python/emr-7.2.0-java8-20240610
- notebook-python/emr-7.2.0-spark-rapids-java8-latest
- notebook-python/emr-7.2.0-spark-rapids-java8-20240610
- livy/emr-7.2.0-latest
- livy/emr-7.2.0-20240610

- livy/emr-7.2.0-java11-latest
- livy/emr-7.2.0-java11-20240610
- livy/emr-7.2.0-java8-latest
- livy/emr-7.2.0-java8-20240610

リリースノート

Amazon EMR on EKS 7.2.0 のリリースノート

- サポートされているアプリケーション - AWS SDK for Java 2.23.18 and 1.12.705, Apache Spark 3.5.1-amzn-1, Apache Hudi 0.14.1-amzn-0, Apache Iceberg 1.5.0-amzn-0, Delta 3.1.0, Apache Spark RAPIDS 24.02.0-amzn-1, Jupyter Enterprise Gateway 2.6.0, Apache Flink 1.18.1-amzn-0, Flink Operator 1.8.0-amzn-1
- サポートされているコンポーネント - `aws-sagemaker-spark-sdk`、`emr-ddb`、`emr-goodies`、`emr-s3-select`、`emrfs`、`hadoop-client`、`hudi`、`hudi-spark`、`iceberg`、`spark-kubernetes`。
- サポートされている設定分類

  StartJobRun API と CreateManagedEndpoint API で使用する場合:

分類	説明
`core-site`	`core-site.xml` Hadoop ファイル内の値を変更します。
`emrfs-site`	EMRFS の設定を変更します。
`spark-metrics`	`metrics.properties` Spark ファイル内の値を変更します。
`spark-defaults`	`spark-defaults.conf` Spark ファイル内の値を変更します。
`spark-env`	Spark 環境の値を変更します。
`spark-hive-site`	`hive-site.xml` Spark ファイル内の値を変更します。

分類	説明
spark-log4j2	log4j2.properties Spark ファイル内の値を変更します。
emr-job-submitter	ジョブ送信者ポッドの設定。

特に CreateManagedEndpoint API で使用する場合:

分類	説明
jeg-config	Jupyter Enterprise Gateway jupyter_enterprise_gateway_config.py ファイルの値を変更します。
jupyter-kernel-overrides	Jupyter カーネル仕様ファイル内のカーネルイメージの値を変更します。

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。これらは多くの場合、spark-hive-site.xml などのアプリケーションの構成 XML ファイルに対応します。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。

注目すべき機能

Amazon EMR on EKS の 7.2.0 リリースには、以下の機能が含まれています。

- アプリケーションのアップグレード - Amazon EMR on EKS 7.2.0 アプリケーションのアップグレードには、Spark 3.5.1、Flink 1.18.1、および Flink Operator 1.8.0 が含まれます。
- Flink 更新用の Autoscaler – 7.2.0 リリースでは、オープンソース設定 job.autoscaler.restart.time-tracking.enabled を使用して再スケーリング時間の推定を有効にするため、再起動時間に経験値を手動で割り当てる必要がなくなります。7.1.0 以前を実行する場合は、引き続き Amazon EMR 自動スケーリングを使用できます。
- Amazon EMR on EKS での Apache Hudi 統合 Apache Flink – このリリースでは、Apache Hudi と Apache Flink 間の統合が追加されているため、Flink Kubernetes 演算子を使用して Hudi ジョブを

実行できます。Hudi では、データ管理とデータパイプライン開発を簡素化するために使用できるレコードレベルのオペレーションを使用できます。

- Amazon S3 Express One Zone と Amazon EMR on EKS との統合 – 7.2.0 以降では、Amazon EMR on EKS を使用して S3 Express One Zone にデータをアップロードできます。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 1 桁のミリ秒単位で一貫したデータアクセスを提供する、高パフォーマンスの単一ゾーンの 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。
- Spark 演算子でのデフォルト設定のサポート – Amazon EKS の Spark 演算子は、7.2.0 以降で Amazon EMR on EKS の開始ジョブ実行モデルと同じデフォルト設定をサポートするようになりました。つまり、Amazon S3 や EMRFS などの機能では、yaml ファイルでの手動設定が不要になります。

emr-7.2.0-latest

リリースノート : 現在 `emr-7.2.0-latest` が指しているのは `emr-7.2.0-20240610` です。

リージョン: `emr-7.2.0-latest` は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: `emr-7.2.0:latest`

emr-7.2.0-20240610

リリースノート: `7.2.0-20240610` は、2023 年 12 月にリリースされました。これは Amazon EMR 7.2.0 (Spark) の初期リリースです。

リージョン: `emr-7.2.0-20240610` は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: `emr-7.2.0:20240610`

emr-7.2.0-flink-latest

リリースノート : 現在 `emr-7.2.0-flink-latest` が指しているのは `emr-7.2.0-flink-20240610` です。

リージョン: `emr-7.2.0-flink-latest` は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: `emr-7.2.0-flink:latest`

emr-7.2.0-flink-20240610

リリースノート: `7.2.0-flink-20240610` は、2023 年 12 月にリリースされました。これは Amazon EMR 7.2.0 (Flink) の初期リリースです。

リージョン: `emr-7.2.0-flink-20240610` は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: `emr-7.2.0-flink:20240610`

Amazon EMR on EKS 7.1.0 リリース

このページでは、Amazon EMR on EKS デプロイに固有の Amazon EMR の新しい機能と更新された機能について説明します。Amazon EC2 で実行されている Amazon EMR の詳細と Amazon EMR 7.1.0 リリース全般の詳細については、「Amazon EMR リリースガイド」の「Amazon EMR 7.1.0」を参照してください。

Amazon EMR on EKS 7.1 リリース

Amazon EMR on EKS では、次の Amazon EMR 7.1.0 リリースを利用できます。特定の emr-7.1.0-XXXX リリースを選択すると、関連するコンテナイメージタグなどの詳細が表示されます。

Flink releases

Flink アプリケーションを実行する場合は、Amazon EMR on EKS で次の Amazon EMR 7.1.0 リリースを利用できます。

- emr-7.1.0-flink-latest
- emr-7.1.0-flink-20240321

Spark releases

Spark アプリケーションを実行する場合は、Amazon EMR on EKS で次の Amazon EMR 7.1.0 リリースを利用できます。

- emr-7.1.0-latest
- emr-7.1.0-20240321
- emr-7.1.0-spark-rapids-latest
- emr-7.1.0-spark-rapids-20240321
- emr-7.1.0-java11-latest
- emr-7.1.0-java11-20240321
- emr-7.1.0-java8-latest
- emr-7.1.0-java8-20240321
- emr-7.1.0-spark-rapids-java8-latest
- emr-7.1.0-spark-rapids-java8-20240321
- notebook-spark/emr-7.1.0-latest
- notebook-spark/emr-7.1.0-20240321
- notebook-spark/emr-7.1.0-spark-rapids-latest
- notebook-spark/emr-7.1.0-spark-rapids-20240321
- notebook-spark/emr-7.1.0-java11-latest
- notebook-spark/emr-7.1.0-java11-20240321
- notebook-spark/emr-7.1.0-java8-latest
- notebook-spark/emr-7.1.0-java8-20240321
- notebook-spark/emr-7.1.0-spark-rapids-java8-latest
- notebook-spark/emr-7.1.0-spark-rapids-java8-20240321
- notebook-python/emr-7.1.0-latest
- notebook-python/emr-7.1.0-20240321
- notebook-python/emr-7.1.0-spark-rapids-latest
- notebook-python/emr-7.1.0-spark-rapids-20240321
- notebook-python/emr-7.1.0-java11-latest
- notebook-python/emr-7.1.0-java11-20240321

- notebook-python/emr-7.1.0-java8-latest
- notebook-python/emr-7.1.0-java8-20240321
- notebook-python/emr-7.1.0-spark-rapids-java8-latest
- notebook-python/emr-7.1.0-spark-rapids-java8-20240321
- livy/emr-7.1.0-latest
- livy/emr-7.1.0-20240321
- livy/emr-7.1.0-java11-latest
- livy/emr-7.1.0-java11-20240321
- livy/emr-7.1.0-java8-latest
- livy/emr-7.1.0-java8-20240321

リリースノート

Amazon EMR on EKS 7.1.0 のリリースノート

- サポートされているアプリケーション - AWS SDK for Java 2.23.18 and 1.12.656, Apache Spark 3.5.0-amzn-1, Apache Hudi 0.14.1-amzn-0, Apache Iceberg 1.4.3-amzn-0, Delta 3.0.0, Apache Spark RAPIDS 23.10.0-amzn-1, Jupyter Enterprise Gateway 2.6.0, Apache Flink 1.18.1-amzn-0, Flink Operator 1.6.1-amzn-1
- サポートされているコンポーネント - `aws-sagemaker-spark-sdk`、`emr-ddb`、`emr-goodies`、`emr-s3-select`、`emrfs`、`hadoop-client`、`hudi`、`hudi-spark`、`iceberg`、`spark-kubernetes`。
- サポートされている設定分類

  StartJobRun API と CreateManagedEndpoint API で使用する場合:

分類	説明
`core-site`	`core-site.xml` Hadoop ファイル内の値を変更します。
`emrfs-site`	EMRFS の設定を変更します。
`spark-metrics`	`metrics.properties` Spark ファイル内の値を変更します。

分類	説明
spark-defaults	spark-defaults.conf Spark ファイル内の値を変更します。
spark-env	Spark 環境の値を変更します。
spark-hive-site	hive-site.xml Spark ファイル内の値を変更します。
spark-log4j2	log4j2.properties Spark ファイル内の値を変更します。
emr-job-submitter	ジョブ送信者ポッドの設定。

特に CreateManagedEndpoint API で使用する場合:

分類	説明
jeg-config	Jupyter Enterprise Gateway jupyter_enterprise_gateway_config.py ファイルの値を変更します。
jupyter-kernel-overrides	Jupyter カーネル仕様ファイル内のカーネルイメージの値を変更します。

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。これらは多くの場合、spark-hive-site.xml などのアプリケーションの構成 XML ファイルに対応します。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。

注目すべき機能

Amazon EMR on EKS の 7.1.0 リリースには、以下の機能が含まれています。

- Amazon EMR on EKS での Apache Livy のサポート – Amazon EMR on EKS リリース 7.1.0 以降では、Amazon EKS クラスターで Apache Livy を使用して Apache Livy REST インターフェイスを作成し、Spark ジョブまたは Spark コードのスニペットを送信できます。これにより、Amazon

EMR に最適化された Spark ランタイム、SSL 対応の Livy エンドポイント、プログラムによるセットアップエクスペリエンスなど、Amazon EMR on EKS の利点を引き続き活用しながら、結果を同期的および非同期的に取得できます。

emr-7.1.0-latest

リリースノート : 現在 `emr-7.1.0-latest` が指しているのは `emr-7.1.0-20240321` です。

リージョン: `emr-7.1.0-latest` は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: `emr-7.1.0:latest`

emr-7.1.0-20240321

リリースノート: `7.1.0-20240321` は、2023 年 12 月にリリースされました。これは Amazon EMR 7.1.0 (Spark) の初期リリースです。

リージョン: `emr-7.1.0-20240321` は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: `emr-7.1.0:20240321`

emr-7.1.0-flink-latest

リリースノート : 現在 `emr-7.1.0-flink-latest` が指しているのは `emr-7.1.0-flink-20240321` です。

リージョン: `emr-7.1.0-flink-latest` は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: `emr-7.1.0-flink:latest`

emr-7.1.0-flink-20240321

リリースノート: `7.1.0-flink-20240321` は、2023 年 12 月にリリースされました。これは Amazon EMR 7.1.0 (Flink) の初期リリースです。

リージョン: `emr-7.1.0-flink-20240321` は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: `emr-7.1.0-flink:20240321`

Amazon EMR on EKS 7.0.0 リリース

このページでは、Amazon EMR on EKS デプロイに固有の Amazon EMR の新しい機能と更新された機能について説明します。Amazon EC2 で実行されている Amazon EMR の詳細と Amazon EMR 7.0.0 リリース全般の詳細については、「Amazon EMR リリースガイド」の「Amazon EMR 7.0.0」を参照してください。

Amazon EMR on EKS 7.0 リリース

Amazon EMR on EKS では、次の Amazon EMR 7.0.0 リリースが利用可能です。特定の emr-7.0.0-XXXX リリースを選択すると、関連するコンテナイメージタグなどの詳細が表示されます。

Flink releases

Flink アプリケーションを実行する場合、EKS 上の Amazon EMR で次の Amazon EMR 7.0.0 リリースを利用できます。

- emr-7.0.0-flink-latest
- emr-7.0.0-flink-2024321
- emr-7.0.0-flink-20231211

Spark releases

Spark アプリケーションを実行する場合、EKS 上の Amazon EMR で次の Amazon EMR 7.0.0 リリースを利用できます。

- emr-7.0.0-latest
- emr-7.0.0-20231211
- emr-7.0.0-spark-rapids-latest
- emr-7.0.0-spark-rapids-20231211
- emr-7.0.0-java11-latest

- emr-7.0.0-java11-20231211
- emr-7.0.0-java8-latest
- emr-7.0.0-java8-20231211
- emr-7.0.0-spark-rapids-java8-latest
- emr-7.0.0-spark-rapids-java8-20231211
- notebook-spark/emr-7.0.0-latest
- notebook-spark/emr-7.0.0-20231211
- notebook-spark/emr-7.0.0-spark-rapids-latest
- notebook-spark/emr-7.0.0-spark-rapids-20231211
- notebook-spark/emr-7.0.0-java11-latest
- notebook-spark/emr-7.0.0-java11-20231211
- notebook-spark/emr-7.0.0-java8-latest
- notebook-spark/emr-7.0.0-java8-20231211
- notebook-spark/emr-7.0.0-spark-rapids-java8-latest
- notebook-spark/emr-7.0.0-spark-rapids-java8-20231211
- notebook-python/emr-7.0.0-latest
- notebook-python/emr-7.0.0-20231211
- notebook-python/emr-7.0.0-spark-rapids-latest
- notebook-python/emr-7.0.0-spark-rapids-20231211
- notebook-python/emr-7.0.0-java11-latest
- notebook-python/emr-7.0.0-java11-20231211
- notebook-python/emr-7.0.0-java8-latest
- notebook-python/emr-7.0.0-java8-20231211
- notebook-python/emr-7.0.0-spark-rapids-java8-latest
- notebook-python/emr-7.0.0-spark-rapids-java8-20231211

リリースノート

Amazon EMR on EKS 7.0.0 リリースノート

- サポートされているアプリケーション - AWS SDK for Java 2.20.160-amzn-0 and 1.12.595, Apache Spark 3.5.0-amzn-0, Apache Flink 1.18.0-amzn-0, Flink Operator 1.6.1, Apache Hudi

0.14.0-amzn-1, Apache Iceberg 1.4.2-amzn-0, Delta 3.0.0, Apache Spark RAPIDS 23.10.0-amzn-0, Jupyter Enterprise Gateway 2.6.0

- サポートされているコンポーネント - `aws-sagemaker-spark-sdk`、`emr-ddb`、`emr-goodies`、`emr-s3-select`、`emrfs`、`hadoop-client`、`hudi`、`hudi-spark`、`iceberg`、`spark-kubernetes`。
- サポートされている設定分類

  StartJobRun API と CreateManagedEndpoint API で使用する場合:

分類	説明
`core-site`	`core-site.xml` Hadoop ファイル内の値を変更します。
`emrfs-site`	EMRFS の設定を変更します。
`spark-metrics`	`metrics.properties` Spark ファイル内の値を変更します。
`spark-defaults`	`spark-defaults.conf` Spark ファイル内の値を変更します。
`spark-env`	Spark 環境の値を変更します。
`spark-hive-site`	`hive-site.xml` Spark ファイル内の値を変更します。
`spark-log4j`	`log4j2.properties` Spark ファイル内の値を変更します。
`emr-job-submitter`	ジョブ送信者ポッドの設定。

  特に CreateManagedEndpoint API で使用する場合:

分類	説明
jeg-config	Jupyter Enterprise Gateway jupyter_enterprise_gateway_config.py ファイルの値を変更します。
jupyter-kernel-overrides	Jupyter カーネル仕様ファイル内のカーネルイメージの値を変更します。

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。これらは多くの場合、spark-hive-site.xml などのアプリケーションの構成 XML ファイルに対応します。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。

注目すべき機能

Amazon EMR on EKS の 7.0 リリースには、以下の機能が含まれています。

- アプリケーションのアップグレード - EKS 7.0.0 上の Amazon EMR アプリケーションのアップグレードには、Spark 3.5、Flink 1.18、および Flink Operator 1.6.1 が含まれます。
- Flink Autoscaler パラメータの自動調整 - Flink Autoscaler がスケーリング計算に使用するデフォルトのパラメータは、特定のジョブに最適な値ではない場合があります。EKS 7.0.0 の Amazon EMR は、キャプチャされた特定のメトリックスの履歴傾向を使用して、ジョブに合わせた最適なパラメータを計算します。

変更

Amazon EMR on EKS の 7.0 リリースには、以下の変更が含まれています。

- Amazon Linux 2023 - EKS 7.0.0 以降の Amazon EMR では、すべてのコンテナイメージが Amazon Linux 2023 をベースにしています。
- Spark は Java 17 をデフォルトのランタイムとして使用します - EKS 7.0.0 Spark 上の Amazon EMR は Java 17 をデフォルトのランタイムとして使用します。必要に応じて、Amazon EMR on EKS 7.0 リリース リストに記載されている対応するリリースラベルの付いた Java 8 または Java 11 を使用するように切り替えることができます。

emr-7.0.0-latest

リリースノート : 現在 `emr-7.0.0-latest` が指しているのは `emr-7.0.0-2024321` です。

リージョン: `emr-7.0.0-latest` は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: `emr-7.0.0:latest`

emr-7.0.0-2024321

リリースノート: `7.0.0-2024321` は、2024 年 3 月 11 日にリリースされました。以前のリリースと比較すると、このリリースは先ごろ更新された Amazon Linux パッケージと重大な修正で更新されています。

リージョン: `emr-7.0.0-2024321` は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: `emr-7.0.0:2024321`

emr-7.0.0-20231211

リリースノート: `7.0.0-20231211` は、2023 年 12 月にリリースされました。これは Amazon EMR 7.0.0 (Spark) の初期リリースです。

リージョン: `emr-7.0.0-20231211` は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: `emr-7.0.0:20231211`

emr-7.0.0-flink-latest

リリースノート : 現在 `emr-7.0.0-flink-latest` が指しているのは `emr-7.0.0-flink-2024321` です。

リージョン: `emr-7.0.0-flink-latest` は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: `emr-7.0.0-flink:latest`

emr-7.0.0-flink-2024321

リリースノート: `7.0.0-flink-2024321` は、2024 年 3 月 11 日にリリースされました。以前のリリースと比較すると、このリリースは先ごろ更新された Amazon Linux パッケージと重大な修正で更新されています。

リージョン: `emr-7.0.0-flink-2024321` は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: `emr-7.0.0-flink:2024321`

emr-7.0.0-flink-20231211

リリースノート: `7.0.0-flink-20231211` は、2023 年 12 月にリリースされました。これは Amazon EMR 7.0.0 (Flink) の初期リリースです。

リージョン: `emr-7.0.0-flink-20231211` は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: `emr-7.0.0-flink:20231211`

Amazon EMR on EKS 6.15.0 リリース

このページでは、Amazon EMR on EKS デプロイに固有の Amazon EMR の新しい機能と更新された機能について説明します。Amazon EC2 で実行されている Amazon EMR の詳細と Amazon EMR 6.15.0 リリース全般の詳細については、「Amazon EMR リリースガイド」の「Amazon EMR 6.15.0」を参照してください。

Amazon EMR on EKS 6.15 リリース

Amazon EMR on EKS では、次の Amazon EMR 6.15.0 リリースが利用可能です。特定の emr-6.15.0-XXXX リリースを選択すると、関連するコンテナイメージタグなどの詳細が表示されます。

Flink releases

Flink アプリケーションを実行する場合、EKS 上の Amazon EMR で次の Amazon EMR 6.15.0 リリースを利用できます。

- emr-6.15.0-flink-latest
- emr-6.15.0-flink-20240105
- emr-6.15.0-flink-20231109

Spark releases

Spark アプリケーションを実行する場合、EKS 上の Amazon EMR で次の Amazon EMR 6.15.0 リリースを利用できます。

- emr-6.15.0-latest
- emr-6.15.0-20231109
- emr-6.15.0-spark-rapids-latest
- emr-6.15.0-spark-rapids-20231109
- emr-6.15.0-java11-latest
- emr-6.15.0-java11-20231109
- emr-6.15.0-java17-latest
- emr-6.15.0-java17-20231109
- emr-6.15.0-java17-al2023-latest
- emr-6.15.0-java17-al2023-20231109
- emr-6.15.0-spark-rapids-java17-latest
- emr-6.15.0-spark-rapids-java17-20231109
- emr-6.15.0-spark-rapids-java17-al2023-latest
- emr-6.15.0-spark-rapids-java17-al2023-20231109
- notebook-spark/emr-6.15.0-latest
- notebook-spark/emr-6.15.0-20231109
- notebook-spark/emr-6.15.0-spark-rapids-latest
- notebook-spark/emr-6.15.0-spark-rapids-20231109
- notebook-spark/emr-6.15.0-java11-latest
- notebook-spark/emr-6.15.0-java11-20231109

- notebook-spark/emr-6.15.0-java17-latest
- notebook-spark/emr-6.15.0-java17-20231109
- notebook-spark/emr-6.15.0-java17-al2023-latest
- notebook-spark/emr-6.15.0-java17-al2023-20231109
- notebook-python/emr-6.15.0-latest
- notebook-python/emr-6.15.0-20231109
- notebook-python/emr-6.15.0-spark-rapids-latest
- notebook-python/emr-6.15.0-spark-rapids-20231109
- notebook-python/emr-6.15.0-java11-latest
- notebook-python/emr-6.15.0-java11-20231109
- notebook-python/emr-6.15.0-java17-latest
- notebook-python/emr-6.15.0-java17-20231109
- notebook-python/emr-6.15.0-java17-al2023-latest
- notebook-python/emr-6.15.0-java17-al2023-20231109

リリースノート

Amazon EMR on EKS 6.15.0 リリースノート

- サポートされているアプリケーション - AWS SDK for Java 1.12.569, Apache Spark 3.4.1-amzn-2, Apache Flink 1.17.1-amzn-1, Apache Hudi 0.14.0-amzn-0, Apache Iceberg 1.4.0-amzn-0, Delta 2.4.0, Apache Spark RAPIDS 23.08.01-amzn-0, Jupyter Enterprise Gateway 2.6.0
- サポートされているコンポーネント - `aws-sagemaker-spark-sdk`、`emr-ddb`、`emr-goodies`、`emr-s3-select`、`emrfs`、`hadoop-client`、`hudi`、`hudi-spark`、`iceberg`、`spark-kubernetes`。
- サポートされている設定分類

  StartJobRun API と CreateManagedEndpoint API で使用する場合:

分類	説明
`core-site`	`core-site.xml` Hadoop ファイル内の値を変更します。

分類	説明
emrfs-site	EMRFS の設定を変更します。
spark-metrics	metrics.properties Spark ファイル内の値を変更します。
spark-defaults	spark-defaults.conf Spark ファイル内の値を変更します。
spark-env	Spark 環境の値を変更します。
spark-hive-site	hive-site.xml Spark ファイル内の値を変更します。
spark-log4j	log4j2.properties Spark ファイル内の値を変更します。
emr-job-submitter	ジョブ送信者ポッドの設定。

特に CreateManagedEndpoint API で使用する場合:

分類	説明
jeg-config	Jupyter Enterprise Gateway jupyter_enterprise_gateway_config.py ファイルの値を変更します。
jupyter-kernel-overrides	Jupyter カーネル仕様ファイル内のカーネルイメージの値を変更します。

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。これらは多くの場合、spark-hive-site.xml などのアプリケーションの構成 XML ファイルに対応します。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。

注目すべき機能

Amazon EMR on EKS の 6.15 リリースには、以下の機能が含まれています。

- Apache Flink を使用した EKS 上の Amazon EMR - EKS 6.15.0 上の Amazon EMR を使用すると、Apache Flink ベースのアプリケーションを同じ Amazon EKS クラスター上の他のタイプのアプリケーションとともに実行できます。これにより、リソース使用率が向上し、インフラストラクチャ管理が簡素化されます。Flink アプリケーションのスポットインスタンスで適切にデコミッションを行い、Amazon EBS によるきめ細かなリカバリとタスクローカルリカバリにより再起動時間を短縮できます。アクセシビリティとモニタリング機能には、Amazon S3 に保存されている jar で Flink アプリケーションを起動する機能、 AWS Glue データカタログへのアクセス、Amazon S3 と Amazon CloudWatch との統合のモニタリング、コンテナログのローテーションなどがあります。

emr-6.15.0-latest

リリースノート : 現在 `emr-6.15.0-latest` が指しているのは `emr-6.15.0-20240105` です。

リージョン: `emr-6.15.0-latest` は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: `emr-6.15.0:latest`

emr-6.15.0-20240105

リリースノート: `6.15.0-20240105` は、2024 年 1 月 17 日にリリースされました。以前のリリースと比較すると、このリリースは先ごろ更新された Amazon Linux パッケージと重大な修正で更新されています。

リージョン: `emr-6.15.0-20240105` は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: `emr-6.15.0:20240105`

emr-6.15.0-20231109

リリースノート: `6.15.0-20231109` は、2023 年 11 月 17 日にリリースされました。これは Amazon EMR 6.15.0 の初回リリースです。

リージョン: emr-6.15.0-20231109 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-6.15.0:20231109

emr-6.15.0-flink-latest

リリースノート : 現在 emr-6.15.0-flink-latest が指しているのは emr-6.15.0-flink-20240105 です。

リージョン: emr-6.15.0-flink-latest は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-6.15.0-flink:latest

emr-6.15.0-flink-20240105

リリースノート: 6.15.0-flink-20240105 は、2024 年 1 月 17 日にリリースされました。以前のリリースと比較すると、このリリースは先ごろ更新された Amazon Linux パッケージと重大な修正で更新されています。

リージョン: emr-6.15.0-flink-20240105 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-6.15.0-flink:20240105

emr-6.15.0-flink-20231109

リリースノート: 6.15.0-flink-20231109 は、2023 年 11 月 17 日にリリースされました。これは Amazon EMR 6.15.0 の初回リリースです。

リージョン: emr-6.15.0-flink-20231109 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-6.15.0-flink:20231109

Amazon EMR on EKS 6.14.0 リリース

このページでは、Amazon EMR on EKS デプロイに固有の Amazon EMR の新しい機能と更新された機能について説明します。Amazon EC2 で実行されている Amazon EMR の詳細と Amazon EMR 6.14.0 リリース全般の詳細については、「Amazon EMR リリースガイド」の「Amazon EMR 6.14.0」を参照してください。

Amazon EMR on EKS 6.14 リリース

Amazon EMR on EKS では、次の Amazon EMR 6.14.0 リリースが利用可能です。特定の emr-6.14.0-XXXX リリースを選択すると、関連するコンテナイメージタグなどの詳細が表示されます。

- emr-6.14.0-latest
- emr-6.14.0-20231005
- emr-6.14.0-spark-rapids-latest
- emr-6.14.0-spark-rapids-20231005
- emr-6.14.0-java11-latest
- emr-6.14.0-java11-20231005
- emr-6.14.0-java17-latest
- emr-6.14.0-java17-20231005
- emr-6.14.0-java17-al2023-latest
- emr-6.14.0-java17-al2023-20231005
- emr-6.14.0-spark-rapids-java17-latest
- emr-6.14.0-spark-rapids-java17-20231005
- emr-6.14.0-spark-rapids-java17-al2023-latest
- emr-6.14.0-spark-rapids-java17-al2023-20231005
- notebook-spark/emr-6.14.0-latest
- notebook-spark/emr-6.14.0-20231005
- notebook-spark/emr-6.14.0-spark-rapids-latest
- notebook-spark/emr-6.14.0-spark-rapids-20231005
- notebook-spark/emr-6.14.0-java11-latest
- notebook-spark/emr-6.14.0-java11-20231005

- notebook-spark/emr-6.14.0-java17-latest
- notebook-spark/emr-6.14.0-java17-20231005
- notebook-spark/emr-6.14.0-java17-al2023-latest
- notebook-spark/emr-6.14.0-java17-al2023-20231005
- notebook-python/emr-6.14.0-latest
- notebook-python/emr-6.14.0-20231005
- notebook-python/emr-6.14.0-spark-rapids-latest
- notebook-python/emr-6.14.0-spark-rapids-20231005
- notebook-python/emr-6.14.0-java11-latest
- notebook-python/emr-6.14.0-java11-20231005
- notebook-python/emr-6.14.0-java17-latest
- notebook-python/emr-6.14.0-java17-20231005
- notebook-python/emr-6.14.0-java17-al2023-latest
- notebook-python/emr-6.14.0-java17-al2023-20231005

リリースノート

Amazon EMR on EKS 6.14.0 のリリースノート

- サポートされているアプリケーション - AWS SDK for Java 1.12.543、Apache Spark 3.4.1-amzn-1、Apache Hudi 0.13.1-amzn-2、Apache Iceberg 1.3.0-amzn-0、Delta 2.4.0、Apache Spark RAPIDS 23.06.0-amzn-2、Jupyter Enterprise Gateway 2.7.0
- サポートされているコンポーネント - `aws-sagemaker-spark-sdk`、`emr-ddb`、`emr-goodies`、`emr-s3-select`、`emrfs`、`hadoop-client`、`hudi`、`hudi-spark`、`iceberg`、`spark-kubernetes`。
- サポートされている設定分類

  StartJobRun API と CreateManagedEndpoint API で使用する場合:

分類	説明
`core-site`	`core-site.xml` Hadoop ファイル内の値を変更します。

分類	説明
emrfs-site	EMRFS の設定を変更します。
spark-metrics	metrics.properties Spark ファイル内の値を変更します。
spark-defaults	spark-defaults.conf Spark ファイル内の値を変更します。
spark-env	Spark 環境の値を変更します。
spark-hive-site	hive-site.xml Spark ファイル内の値を変更します。
spark-log4j	log4j2.properties Spark ファイル内の値を変更します。
emr-job-submitter	ジョブ送信者ポッドの設定。

特に CreateManagedEndpoint API で使用する場合:

分類	説明
jeg-config	Jupyter Enterprise Gateway jupyter_enterprise_gateway_config.py ファイルの値を変更します。
jupyter-kernel-overrides	Jupyter カーネル仕様ファイル内のカーネルイメージの値を変更します。

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。これらは多くの場合、spark-hive-site.xml などのアプリケーションの構成 XML ファイルに対応します。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。

注目すべき機能

Amazon EMR on EKS の 6.14 リリースには、以下の機能が含まれています。

- Apache Livy サポート - Amazon EMR on EKS では、`spark-submit` で Apache Livy をサポートするようになりました。

emr-6.14.0-latest

リリースノート : 現在 `emr-6.14.0-latest` が指しているのは `emr-6.14.0-20231005` です。

リージョン: `emr-6.14.0-latest` は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: `emr-6.14.0:latest`

emr-6.14.0-20231005

リリースノート: `6.14.0-20231005` は、2023 年 10 月 17 日にリリースされました。これは Amazon EMR 6.14.0 の初期リリースです。

リージョン: `emr-6.14.0-20231005` は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: `emr-6.14.0:20231005`

Amazon EMR on EKS 6.13.0 リリース

このページでは、Amazon EMR on EKS デプロイに固有の Amazon EMR の新しい機能と更新された機能について説明します。Amazon EC2 で実行されている Amazon EMR の詳細と Amazon EMR 6.13.0 リリース全般の詳細については、「Amazon EMR リリースガイド」の「Amazon EMR 6.13.0」を参照してください。

Amazon EMR on EKS 6.13 リリース

Amazon EMR on EKS では、次の Amazon EMR 6.13.0 リリースが利用可能です。特定の emr-6.13.0-XXXX リリースを選択すると、関連するコンテナイメージタグなどの詳細が表示されます。

- emr-6.13.0-latest
- emr-6.13.0-20230814
- emr-6.13.0-spark-rapids-latest
- emr-6.13.0-spark-rapids-20230814
- emr-6.13.0-java11-latest
- emr-6.13.0-java11-20230814
- emr-6.13.0-java17-latest
- emr-6.13.0-java17-20230814
- emr-6.13.0-java17-al2023-latest
- emr-6.13.0-java17-al2023-20230814
- emr-6.13.0-spark-rapids-java17-latest
- emr-6.13.0-spark-rapids-java17-20230814
- emr-6.13.0-spark-rapids-java17-al2023-latest
- emr-6.13.0-spark-rapids-java17-al2023-20230814
- notebook-spark/emr-6.13.0-latest
- notebook-spark/emr-6.13.0-20230814
- notebook-spark/emr-6.13.0-spark-rapids-latest
- notebook-spark/emr-6.13.0-spark-rapids-20230814
- notebook-spark/emr-6.13.0-java11-latest
- notebook-spark/emr-6.13.0-java11-20230814
- notebook-spark/emr-6.13.0-java17-latest
- notebook-spark/emr-6.13.0-java17-20230814
- notebook-spark/emr-6.13.0-java17-al2023-latest
- notebook-spark/emr-6.13.0-java17-al2023-20230814
- notebook-python/emr-6.13.0-latest
- notebook-python/emr-6.13.0-20230814
- notebook-python/emr-6.13.0-spark-rapids-latest
- notebook-python/emr-6.13.0-spark-rapids-20230814
- notebook-python/emr-6.13.0-java11-latest

- notebook-python/emr-6.13.0-java11-20230814
- notebook-python/emr-6.13.0-java17-latest
- notebook-python/emr-6.13.0-java17-20230814
- notebook-python/emr-6.13.0-java17-al2023-latest
- notebook-python/emr-6.13.0-java17-al2023-20230814

リリースノート

Amazon EMR on EKS 6.13.0 のリリースノート

- サポートされているアプリケーション - AWS SDK for Java 1.12.513、Apache Spark 3.4.1-amzn-0、Apache Hudi 0.13.1-amzn-0、Apache Iceberg 1.3.0-amzn-0、Delta 2.4.0、Apache Spark RAPIDS 23.06.0-amzn-1、Jupyter Enterprise Gateway 2.6.0.amzn
- サポートされているコンポーネント - `aws-sagemaker-spark-sdk`、`emr-ddb`、`emr-goodies`、`emr-s3-select`、`emrfs`、`hadoop-client`、`hudi`、`hudi-spark`、`iceberg`、`spark-kubernetes`。
- サポートされている設定分類

  StartJobRun API と CreateManagedEndpoint API で使用する場合:

分類	説明
`core-site`	`core-site.xml` Hadoop ファイル内の値を変更します。
`emrfs-site`	EMRFS の設定を変更します。
`spark-metrics`	`metrics.properties` Spark ファイル内の値を変更します。
`spark-defaults`	`spark-defaults.conf` Spark ファイル内の値を変更します。
`spark-env`	Spark 環境の値を変更します。
`spark-hive-site`	`hive-site.xml` Spark ファイル内の値を変更します。

分類	説明
spark-log4j	log4j2.properties Spark ファイル内の値を変更します。
emr-job-submitter	ジョブ送信者ポッドの設定。

特に CreateManagedEndpoint API で使用する場合:

分類	説明
jeg-config	Jupyter Enterprise Gateway jupyter_enterprise_gateway_config.py ファイルの値を変更します。
jupyter-kernel-overrides	Jupyter カーネル仕様ファイル内のカーネルイメージの値を変更します。

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。これらは多くの場合、spark-hive-site.xml などのアプリケーションの構成 XML ファイルに対応します。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。

注目すべき機能

Amazon EMR on EKS の 6.13 リリースには、以下の機能が含まれています。

- Amazon Linux 2023 - Amazon EMR on EKS 6.13 以降では、AL2023 で Java 17 ランタイムと共に Spark をオペレーティングシステムとして起動できます。そのためには、名前に al2023 を含めたリリースラベルを使用します。例: emr-6.13.0-java17-al2023-latest。本番稼働用ワークロードを AL2023 と Java 17 に移行する前に、パフォーマンステストを検証して実行することをお勧めします。
- Amazon EMR on EKS での Apache Flink の使用 (パブリックプレビュー) - Amazon EMR on EKS は、リリース 6.13 以降、Apache Flink をサポートしており、パブリックプレビューで使用可能です。このリリースでは、Apache Flink ベースのアプリケーションを他のタイプのアプリケーションと共に同じ Amazon EKS クラスターで実行できます。これにより、リソース使用率が向上し、

インフラストラクチャ管理が簡素化されます。Amazon EKS でビッグデータフレームワークを既に実行している場合は、Amazon EMR を使用してプロビジョニングと管理を自動化できるようになりました。

emr-6.13.0-latest

リリースノート : 現在 `emr-6.13.0-latest` が指しているのは `emr-6.13.0-20230814` です。

リージョン: `emr-6.13.0-latest` は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: `emr-6.13.0:latest`

emr-6.13.0-20230814

リリースノート: `6.13.0-20230814` は、2023 年 9 月 7 日にリリースされました。これは Amazon EMR 6.13.0 の初期リリースです。

リージョン: `emr-6.13.0-20230814` は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: `emr-6.13.0:20230814`

Amazon EMR on EKS 6.12.0 リリース

このページでは、Amazon EMR on EKS デプロイに固有の Amazon EMR の新しい機能と更新された機能について説明します。Amazon EC2 で実行されている Amazon EMR の詳細と Amazon EMR 6.12.0 リリース全般の詳細については、「Amazon EMR リリースガイド」の「Amazon EMR 6.12.0」を参照してください。

Amazon EMR on EKS 6.12 リリース

Amazon EMR on EKS では、次の Amazon EMR 6.12.0 リリースが利用可能です。特定の emr-6.12.0-XXXX リリースを選択すると、関連するコンテナイメージタグなどの詳細が表示されます。

- emr-6.12.0-latest

- emr-6.12.0-20240321
- emr-6.12.0-20230701
- emr-6.12.0-spark-rapids-latest
- emr-6.12.0-spark-rapids-20230701
- emr-6.12.0-java11-latest
- emr-6.12.0-java11-20230701
- emr-6.12.0-java17-latest
- emr-6.12.0-java17-20230701
- emr-6.12.0-spark-rapids-java17-latest
- emr-6.12.0-spark-rapids-java17-20230701
- notebook-spark/emr-6.12.0-latest
- notebook-spark/emr-6.12.0-20230701
- notebook-spark/emr-6.12.0-spark-rapids-latest
- notebook-spark/emr-6.12.0-spark-rapids-20230701
- notebook-python/emr-6.12.0-latest
- notebook-python/emr-6.12.0-20230701
- notebook-python/emr-6.12.0-spark-rapids-latest
- notebook-python/emr-6.12.0-spark-rapids-20230701

リリースノート

Amazon EMR on EKS 6.12.0 のリリースノート

- サポートされているアプリケーション - AWS SDK for Java 1.12.490、Apache Spark 3.4.0-amzn-0、Apache Hudi 0.13.1-amzn-0、Apache Iceberg 1.3.0-amzn-0、Delta 2.4.0、Apache Spark RAPIDS 23.06.0-amzn-0、Jupyter Enterprise Gateway 2.6.0
- サポートされているコンポーネント - `aws-sagemaker-spark-sdk`、`emr-ddb`、`emr-goodies`、`emr-s3-select`、`emrfs`、`hadoop-client`、`hudi`、`hudi-spark`、`iceberg`、`spark-kubernetes`。
- サポートされている設定分類

  StartJobRun API と CreateManagedEndpoint API で使用する場合:

分類	説明
core-site	core-site.xml Hadoop ファイル内の値を変更します。
emrfs-site	EMRFS の設定を変更します。
spark-metrics	metrics.properties Spark ファイル内の値を変更します。
spark-defaults	spark-defaults.conf Spark ファイル内の値を変更します。
spark-env	Spark 環境の値を変更します。
spark-hive-site	hive-site.xml Spark ファイル内の値を変更します。
spark-log4j	log4j2.properties Spark ファイル内の値を変更します。
emr-job-submitter	ジョブ送信者ポッドの設定。

特に CreateManagedEndpoint API で使用する場合:

分類	説明
jeg-config	Jupyter Enterprise Gateway jupyter_enterprise_gateway_config.py ファイルの値を変更します。
jupyter-kernel-overrides	Jupyter カーネル仕様ファイル内のカーネルイメージの値を変更します。

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。これらは多くの場合、spark-hive-site.xml などのアプリケーションの構成 XML ファイルに対応します。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。

注目すべき機能

Amazon EMR on EKS の 6.12 リリースには、以下の機能が含まれています。

- Java 17 - Amazon EMR on EKS 6.12 以降では、Spark を Java 17 ランタイムと共に起動できます。このためには、emr-6.12.0-java17-latest をリリースラベルとして渡します。本番稼働用ワークロードを以前のバージョンの Java イメージから Java 17 イメージに移行する前に、パフォーマンステストを検証して実行することをお勧めします。

emr-6.12.0-latest

リリースノート : 現在 emr-6.12.0-latest が指しているのは emr-6.12.0-20240321 です。

リージョン: emr-6.12.0-latest は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-6.12.0:latest

emr-6.12.0-20240321

リリースノート: 6.12.0-20240321 は、2024 年 3 月 11 日にリリースされました。以前のリリースと比較すると、このリリースは先ごろ更新された Amazon Linux パッケージと重大な修正で更新されています。

リージョン: emr-6.12.0-20240321 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-6.12.0:20240321

emr-6.12.0-20230701

リリースノート: `6.12.0-20230701` は、2023 年 7 月 1 日にリリースされました。これは Amazon EMR 6.12.0 の初期リリースです。

リージョン: `emr-6.12.0-20230701` は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: `emr-6.12.0:20230701`

Amazon EMR on EKS 6.11.0 リリース

このページでは、Amazon EMR on EKS デプロイに固有の Amazon EMR の新しい機能と更新された機能について説明します。Amazon EC2 で実行されている Amazon EMR の詳細と Amazon EMR 6.11.0 リリース全般の詳細については、「Amazon EMR リリースガイド」の「Amazon EMR 6.11.0」を参照してください。

Amazon EMR on EKS 6.11 リリース

Amazon EMR on EKS では、次の Amazon EMR 6.11.0 リリースが利用可能です。特定の emr-6.11.0-XXXX リリースを選択すると、関連するコンテナイメージタグなどの詳細が表示されます。

- emr-6.11.0-latest
- emr-6.11.0-20230905
- emr-6.11.0-20230509

- emr-6.11.0-spark-rapids-latest
- emr-6.11.0-spark-rapids-20230509
- emr-6.11.0-java11-latest
- emr-6.11.0-java11-20230509
- notebook-spark/emr-6.11.0-latest
- notebook-spark/emr-6.11.0-20230509
- notebook-python/emr-6.11.0-latest
- notebook-python/emr-6.11.0-20230509

リリースノート

Amazon EMR on EKS 6.11.0 のリリースノート

- サポートされているアプリケーション - AWS SDK for Java 1.12.446、Apache Spark 3.3.2-amzn-0、Apache Hudi 0.13.0-amzn-0、Apache Iceberg 1.2.0-amzn-0、Delta 2.2.0、Apache Spark RAPIDS 23.02.0-amzn-0、Jupyter Enterprise Gateway 2.6.0
- サポートされているコンポーネント - `aws-sagemaker-spark-sdk`、`emr-ddb`、`emr-goodies`、`emr-s3-select`、`emrfs`、`hadoop-client`、`hudi`、`hudi-spark`、`iceberg`、`spark-kubernetes`。
- サポートされている設定分類

  StartJobRun API と CreateManagedEndpoint API で使用する場合:

分類	説明
`core-site`	`core-site.xml` Hadoop ファイル内の値を変更します。
`emrfs-site`	EMRFS の設定を変更します。
`spark-metrics`	`metrics.properties` Spark ファイル内の値を変更します。
`spark-defaults`	`spark-defaults.conf` Spark ファイル内の値を変更します。
`spark-env`	Spark 環境の値を変更します。
`spark-hive-site`	`hive-site.xml` Spark ファイル内の値を変更します。
`spark-log4j`	`log4j.properties` Spark ファイル内の値を変更します。

  特に CreateManagedEndpoint API で使用する場合:

分類	説明
jeg-config	Jupyter Enterprise Gateway jupyter_enterprise_gateway_config.py ファイルの値を変更します。
jupyter-kernel-overrides	Jupyter カーネル仕様ファイル内のカーネルイメージの値を変更します。

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。これらは多くの場合、spark-hive-site.xml などのアプリケーションの構成 XML ファイルに対応します。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。

注目すべき機能

Amazon EMR on EKS の 6.11 リリースには、以下の機能が含まれています。

- Amazon ECR Public Gallery 内の Amazon EMR on EKS ベースイメージ – カスタムイメージ機能を使用する場合は、ベースイメージに用意されている必須の jar、設定、ライブラリを利用して、Amazon EMR on EKS とやり取りできます。ベースイメージは、Amazon ECR Public Gallery に収められるようになりました。
- Spark コンテナログのローテーション – Amazon EMR on EKS 6.11 は、Spark コンテナログのローテーションをサポートしています。この機能を有効にするには、StartJobRun API の MonitoringConfiguration オペレーション内で containerLogRotationConfiguration を使用します。rotationSize と maxFilestoKeep を設定すると、Amazon EMR on EKS の Spark ドライバーポッドとエグゼキューターポッドに保持されるログファイルの数とサイズを指定できます。詳細については、「Spark コンテナログのローテーションを使用する」を参照してください。
- Spark オペレータと spark-submit での Volcano サポート – Amazon EMR on EKS 6.11 では、Spark オペレータと spark-submit で Volcano を Kubernetes カスタムスケジューラとして使用して、Spark ジョブを実行できます。ギャングスケジューリング、キュー管理、プリエンプション、フェアシェアスケジューリングなどの機能を使用すると、スケジューリングのスループットを高め、キャパシティを最適化できます。詳細については、「Amazon EMR on EKS で Apache Spark のカスタムスケジューラとして Volcano を使用する方法」を参照してください。

emr-6.11.0-latest

リリースノート : 現在 emr-6.11.0-latest が指しているのは emr-20230905 です。

リージョン: emr-6.11.0-latest は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-6.11.0:latest

emr-6.11.0-20230905

リリースノート: 6.11.0-20230905 は、2023 年 9 月 29 日にリリースされました。以前のリリースと比較すると、このリリースは先ごろ更新された Amazon Linux パッケージと重大な修正で更新されています。

リージョン: emr-6.11.0-20230509 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-6.11.0:20230509

emr-6.11.0-20230509

リリースノート: 6.11.0-20230509 は、2023 年 5 月 9 日にリリースされました。これは Amazon EMR 6.11.0 の初期リリースです。

リージョン: emr-6.11.0-20230509 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-6.11.0:20230509

Amazon EMR on EKS 6.10.0 リリース

Amazon EMR on EKS では、次の Amazon EMR 6.10.0 リリースが利用可能です。特定の emr-6.10.0-XXXX リリースを選択すると、関連するコンテナイメージタグなどの詳細が表示されます。

- emr-6.10.0-latest

- emr-6.10.0-20230905
- emr-6.10.0-20230624
- emr-6.10.0-20230421
- emr-6.10.0-20230403
- emr-6.10.0-20230220
- emr-6.10.0-spark-rapids-latest
- emr-6.10.0-spark-rapids-20230624
- emr-6.10.0-spark-rapids-20230220
- emr-6.10.0-java11-latest
- emr-6.10.0-java11-20230624
- emr-6.10.0-java11-20230220
- notebook-spark/emr-6.10.0-latest
- notebook-spark/emr-6.10.0-20230624
- notebook-spark/emr-6.10.0-20230220
- notebook-python/emr-6.10.0-latest
- notebook-python/emr-6.10.0-20230624
- notebook-python/emr-6.10.0-20230220

Amazon EMR 6.10.0 のリリースノート

- サポートされているアプリケーション - AWS SDK for Java 1.12.397、Spark 3.3.1-amzn-0、Hudi 0.12.2-amzn-0、Iceberg 1.1.0-amzn-0、Delta 2.2.0。
- サポートされているコンポーネント - `aws-sagemaker-spark-sdk`、`emr-ddb`、`emr-goodies`、`emr-s3-select`、`emrfs`、`hadoop-client`、`hudi`、`hudi-spark`、`iceberg`、`spark-kubernetes`。
- サポートされている設定分類 :

  StartJobRun API と CreateManagedEndpoint API で使用する場合:

分類	説明
`core-site`	Hadoop の `core-site.xml` ファイル内の値を変更します。

分類	説明
emrfs-site	EMRFS の設定を変更します。
spark-metrics	Spark の metrics.properties ファイル内の値を変更します
spark-defaults	Spark の spark-defaults.conf ファイル内の値を変更します
spark-env	Spark 環境の値を変更します。
spark-hive-site	Spark の hive-site.xml ファイル内の値を変更します
spark-log4j	Spark の log4j.properties ファイル内の値を変更します

特に CreateManagedEndpoint API で使用する場合:

分類	説明
jeg-config	Jupyter Enterprise Gateway jupyter_enterprise_gateway_config.py ファイルの値を変更します。
jupyter-kernel-overrides	Jupyter カーネル仕様ファイル内のカーネルイメージの値を変更します。

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。これらは多くの場合、spark-hive-site.xml などのアプリケーションの構成 XML ファイルに対応します。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。

注目すべき機能

- Spark オペレータ - Amazon EMR on EKS 6.10.0 以降では、Apache Spark の Kubernetes オペレータ (Spark オペレータ) を使用して、Spark アプリケーションを Amazon EMR リリースラン

タイムと共に独自の Amazon EKS クラスターにデプロイして管理できます。詳細については、「Spark 演算子を使用して Spark ジョブを実行する」を参照してください。

- Java 11 - Amazon EMR on EKS 6.10 以降では、Spark を Java 11 ランタイムと共に起動できます。このためには、`emr-6.10.0-java11-latest` をリリースラベルとして渡します。本番稼働用ワークロードを Java 8 イメージから Java 11 イメージに移行する前に、パフォーマンステストを検証して実行することをお勧めします。
- Amazon Redshift integration for Apache Spark の場合、Amazon EMR on EKS 6.10.0 は `minimal-json.jar` への依存をなくし、Spark に必要な `spark-redshift` 関連の jar (`spark-redshift.jar`、`spark-avro.jar`、`RedshiftJDBC.jar`) をエグゼキュタークラスパスに自動的に追加します。

変更

- EMRFS S3 に最適化されたコミッターを parquet、ORC、テキストベースの形式 (CSV や JSON など) でデフォルトで使用できるようになりました。

emr-6.10.0-latest

リリースノート : 現在 `emr-6.10.0-latest` が指しているのは `emr-6.10.0-20230905` です。

リージョン: `emr-6.10.0-latest` は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: `emr-6.10.0:latest`

emr-6.10.0-20230905

リリースノート: `6.10.0-20230905` は、2023 年 9 月 29 日にリリースされました。前回のリリースと比較すると、このバージョンは先ごろ更新された Amazon Linux パッケージと重要な修正で更新されています。

リージョン: `emr-6.10.0-20230905` は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: `emr-6.10.0:20230905`

emr-6.10.0-20230624

リリースノート: 6.10.0-20230624 は、2023 年 7 月 7 日にリリースされました。前回のリリースと比較すると、このバージョンは先ごろ更新された Amazon Linux パッケージと重要な修正で更新されています。

リージョン: emr-6.10.0-20230624 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-6.10.0:20230624

emr-6.10.0-20230421

リリースノート: 6.10.0-20230421 は、2023 年 4 月 28 日にリリースされました。前回のリリースと比較すると、このバージョンは先ごろ更新された Amazon Linux パッケージと重要な修正で更新されています。

リージョン: emr-6.10.0-20230421 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-6.10.0:20230421

emr-6.10.0-20230403

リリースノート: 6.10.0-20230403 は、2023 年 4 月 12 日にリリースされました。前回のリリースと比較すると、このバージョンは先ごろ更新された Amazon Linux パッケージと重要な修正で更新されています。

リージョン: emr-6.10.0-20230403 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-6.10.0:20230403

emr-6.10.0-20230220

リリースノート: emr-6.10.0-20230220 は、2023 年 2 月 20 日にリリースされました。これは Amazon EMR 6.10.0 の初期リリースです。

リージョン: emr-6.10.0-20230220 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-6.10.0:20230220

Amazon EMR on EKS 6.9.0 リリース

Amazon EMR on EKS では、次の Amazon EMR 6.9.0 リリースが利用可能です。特定の emr-6.9.0-XXXX リリースを選択すると、関連するコンテナイメージタグなどの詳細が表示されます。

- emr-6.9.0-latest
- emr-6.9.0-20230905
- emr-6.9.0-20230624
- emr-6.9.0-20221108
- emr-6.9.0-spark-rapids-latest
- emr-6.9.0-spark-rapids-20230624
- emr-6.9.0-spark-rapids-20221108
- notebook-spark/emr-6.9.0-latest
- notebook-spark/emr-6.9.0-20230624
- notebook-spark/emr-6.9.0-20221108
- notebook-python/emr-6.9.0-latest
- notebook-python/emr-6.9.0-20230624
- notebook-python/emr-6.9.0-20221108

Amazon EMR 6.9.0 のリリースノート

- サポートされているアプリケーション - AWS SDK for Java 1.12.331、Spark 3.3.0-amzn-1、Hudi 0.12.1-amzn-0、Iceberg 0.14.1-amzn-0、Delta 2.1.0。
- サポートされているコンポーネント - aws-sagemaker-spark-sdk、emr-ddb、emr-goodies、emr-s3-select、emrfs、hadoop-client、hudi、hudi-spark、iceberg、spark-kubernetes。
- サポートされている設定分類 :

  StartJobRun API と CreateManagedEndpoint API で使用する場合:

分類	説明
core-site	Hadoop の core-site.xml ファイルの値を変更します。
emrfs-site	EMRFS の設定を変更します。
spark-metrics	Spark の metrics.properties ファイルの値を変更します。
spark-defaults	Spark の spark-defaults.conf ファイルの値を変更します。
spark-env	Spark 環境の値を変更します。
spark-hive-site	Spark の hive-site.xml ファイルの値を変更します
spark-log4j	Spark の log4j.properties ファイルの値を変更します。

特に CreateManagedEndpoint API で使用する場合:

分類	説明
jeg-config	Jupyter Enterprise Gateway jupyter_enterprise_gateway_config.py ファイルの値を変更します。
jupyter-kernel-overrides	Jupyter カーネル仕様ファイル内のカーネルイメージの値を変更します。

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。これらは多くの場合、spark-hive-site.xml などのアプリケーションの構成 XML ファイルに対応します。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。

注目すべき機能

- Nvidia RAPIDS Accelerator for Apache Spark - Amazon EMR on EKS では、EC2 グラフィック処理ユニット (GPU) インスタンスタイプを使用して、Spark を加速化できます。RAPIDS Accelerator で Spark イメージを使用するには、リリースラベルとして emr-6.9.0-spark-rapids-latest を指定します。詳細については、ドキュメントページを参照してください。
- Spark-Redshift コネクタ - Amazon Redshift integration for Apache Spark 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 on EKS での Amazon Redshift integration for Apache Spark の使用」を参照してください。
- Delta Lake - Delta Lake は、オープンソースのストレージ形式であり、一貫性のあるトランザクション、一貫性のあるデータセット定義、スキーマ進化の変更、データミューテーションのサポートを備えたデータレイクを構築できます。詳細については、「Using Delta Lake」を参照してください。
- PySpark パラメータの変更 - インタラクティブエンドポイントでは、EMR Studio Jupyter Notebook で PySpark セッションに関連付けられている Spark パラメータを変更できるようになりました。詳細については、「Modifying PySpark session parameters」を参照してください。

解決された問題

- Spark on Amazon EMR バージョン 6.6.0、6.7.0、6.8.0 で DynamoDB コネクタを使用すると、テーブルから何を読み込んでも空の結果が返されます。この状況は、入力分割が空でないデータを参照している場合でも変わりません。Amazon EMR リリース 6.9.0 では、この問題が修正されています。
- Amazon EMR on EKS 6.8.0 では、Apache Spark を使用して生成された Parquet ファイルのメタデータにビルドハッシュが誤って入力されます。この問題のため、Amazon EMR on EKS 6.8.0 が生成した Parquet ファイルのメタデータバージョン文字列をツールで解析しようとすると、ツールが失敗する場合があります。

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-6.9.0-latest

リリースノート : 現在 emr-6.9.0-latest が指しているのは emr-6.9.0-20230905 です。

リージョン: emr-6.9.0-latest は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-6.9.0:latest

emr-6.9.0-20230905

リリースノート: emr-6.9.0-20230905。以前のリリースと比較すると、このリリースは先ごろ更新された Amazon Linux パッケージと重大な修正で更新されています。

リージョン: emr-6.9.0-20230905 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-6.9.0:20230905

emr-6.9.0-20230624

リリースノート: emr-6.9.0-20230624 は、2023 年 7 月 7 日にリリースされました。

リージョン: emr-6.9.0-20230624 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-6.9.0:20230624

emr-6.9.0-20221108

リリースノート: emr-6.9.0-20221108 は、2022 年 12 月 8 日にリリースされました。これは Amazon EMR 6.9.0 の初期リリースです。

リージョン: emr-6.9.0-20221108 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-6.9.0:20221108

Amazon EMR on EKS 6.8.0 リリース

Amazon EMR on EKS では、次の Amazon EMR 6.8.0 リリースが利用可能です。特定の emr-6.8.0-XXXX リリースを選択すると、関連するコンテナイメージタグなどの詳細が表示されます。

- emr-6.8.0-latest
- emr-6.8.0-20230905
- emr-6.8.0-20230624
- emr-6.8.0-20221219
- emr-6.8.0-20220802

Amazon EMR 6.8.0 のリリースノート

- サポートされているアプリケーション - AWS SDK for Java 1.12.170、Spark 3.3.0-amzn-0、Hudi 0.11.1-amzn-0、Iceberg 0.14.0-amzn-0。
- サポートされているコンポーネント - aws-sagemaker-spark-sdk、emr-ddb、emr-goodies、emr-s3-select、emrfs、hadoop-client、hudi、hudi-spark、iceberg、spark-kubernetes。
- サポートされている設定分類 :

分類	説明
core-site	Hadoop の core-site.xml ファイルの値を変更します。
emrfs-site	EMRFS の設定を変更します。
spark-metrics	Spark の metrics.properties ファイルの値を変更します。

分類	説明
spark-defaults	Spark の spark-defaults.conf ファイルの値を変更します。
spark-env	Spark 環境の値を変更します。
spark-hive-site	Spark の hive-site.xml ファイルの値を変更します
spark-log4j	Spark の log4j.properties 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。これらは多くの場合、spark-hive-site.xml などのアプリケーションの構成 XML ファイルに対応します。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。

注目すべき機能

- Spark3.3.0 - Amazon EMR on EKS 6.8 には、Spark 3.3.0 が含まれています。これにより、Spark ドライバーエグゼキューターポッドに個別のノードセレクターラベルを使用できるようになりました。こうした新しいラベルを使用すると、ポッドテンプレートを使用しなくても StartJobRun API でドライバーポッドとエグゼキューターポッドのノードタイプを個別に定義できます。
  - ドライバーノードセレクタープロパティ: spark.kubernetes.driver.node.selector.[ラベルキー]
  - エグゼキューターノードセレクタープロパティ: spark.kubernetes.executor.node.selector.[ラベルキー]
- ジョブ失敗メッセージの充実 - このリリースでは、設定 spark.stage.extraDetailsOnFetchFailures.enabled と spark.stage.extraDetailsOnFetchFailures.maxFailuresToInclude が導入され、ユーザーコードによるタスク失敗を追跡できるようになりました。こうした設定の詳しい情報を使用すると、シャッフルフェッチの失敗によってステージが中止されたときにドライバーログに表示される失敗メッセージを充実させることができます。

プロパティ名	デフォルト値	意味	バージョン以降
spark.stage.extraDetailsOnFetchFailures.enabled	false	true に設定すると、このプロパティを使用して、シャッフルフェッチの失敗によってステージが中止されたときにドライバーログに表示されるジョブ失敗メッセージを充実させることができます。デフォルトでは、ユーザーコードによるタスク失敗のうち直近 5 件が追跡され、失敗エラーメッセージがドライバーログに追加されます。 ユーザー例外で追跡できるタスク失敗の数を増やすには、設定 spark.stage.extraDetailsOnFetchFailures.maxFailuresToInclude を参照してください。	emr-6.8

プロパティ名	デフォルト値	意味	バージョン以降
spark.stage.extraDetailsOnFetchFailures.maxFailuresToInclude	5	ステージおよび試行ごとに追跡できるタスク失敗の数。このプロパティを使用すると、シャッフルフェッチの失敗によってステージが中止されたときにドライバーログに表示されるジョブ失敗メッセージをユーザー例外で充実させることができます。 このプロパティは、設定 spark.stage.extraDetailsOnFetchFailures.enabled が true に設定されている場合にのみ機能します。	emr-6.8

詳細については、「Apache Spark の設定ドキュメント」を参照してください。

既知の問題

- Amazon EMR on EKS 6.8.0 では、Apache Spark を使用して生成された Parquet ファイルのメタデータにビルドハッシュが誤って入力されます。この問題のため、Amazon EMR on EKS 6.8.0 が生成した Parquet ファイルのメタデータバージョン文字列をツールで解析しようとすると、ツールが失敗する場合があります。Parquet メタデータからバージョン文字列を解析し、ビルドハッシュに依存するお客様は、別の Amazon EMR バージョンに切り替えて、Parquet ファイルを書き換える必要があります。

解決された問題

- pySpark カーネルのカーネル割り込み機能 - ノートブックでセルを実行することでトリガーされるインタラクティブワークロードが進行中の場合、Interrupt Kernel 機能を使用して停止できます。pySpark カーネルに対してこの機能が動作するように修正が施されました。これは、「PySpark Kubernetes カーネルの割り込み処理に関する変更 #1115」でオープンソースでも使用可能です。

emr-6.8.0-latest

リリースノート : 現在 emr-6.8.0-latest が指しているのは emr-6.8.0-20230624 です。

リージョン: emr-6.8.0-latest は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-6.8.0:latest

emr-6.8.0-20230905

リリースノート: emr-6.8.0-20230905 は、2023 年 9 月 29 日にリリースされました。以前のリリースと比較すると、このリリースは先ごろ更新された Amazon Linux パッケージと重大な修正で更新されています。

リージョン: emr-6.8.0-20230905 は、Amazon EMR で EKS でサポートされているすべてのリージョンで使用可能です。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-6.8.0:20230905

emr-6.8.0-20230624

リリースノート: emr-6.8.0-20230624 は、2023 年 7 月 7 日にリリースされました。以前のリリースと比較すると、このリリースは先ごろ更新された Amazon Linux パッケージと重大な修正で更新されています。

リージョン: emr-6.8.0-20230624 は、Amazon EMR で EKS でサポートされているすべてのリージョンで使用可能です。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-6.8.0:20230624

emr-6.8.0-20221219

リリースノート: emr-6.8.0-20221219 は、2023 年 1 月 19 日にリリースされました。以前のリリースと比較すると、このリリースは先ごろ更新された Amazon Linux パッケージと重大な修正で更新されています。

リージョン: emr-6.8.0-20221219 は、Amazon EMR で EKS でサポートされているすべてのリージョンで使用可能です。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-6.8.0:20221219

emr-6.8.0-20220802

リリースノート: emr-6.8.0-20220802 は、2022 年 9 月 27 日にリリースされました。これは Amazon EMR 6.8.0 の初期リリースです。

リージョン: emr-6.8.0-20220802 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-6.8.0:20220802

Amazon EMR on EKS 6.7.0 リリース

Amazon EMR on EKS では、次の Amazon EMR 6.7.0 リリースが利用可能です。特定の emr-6.7.0-XXXX リリースを選択すると、関連するコンテナイメージタグなどの詳細が表示されます。

- emr-6.7.0-latest
- emr-6.7.0-20240321
- emr-6.7.0-20230624
- emr-6.7.0-20221219
- emr-6.7.0-20220630

Amazon EMR 6.7.0 のリリースノート

- サポートされているアプリケーション - Spark 3.2.1-amzn-0、Jupyter Enterprise Gateway 2.6、Hudi 0.11-amzn-0、Iceberg 0.13.1。

- サポートされるコンポーネント-aws-hm-client (Glue コネクタ) 、aws-sagemaker-spark-sdk、emr-s3-select、emrfs、emr-ddb、hudi-spark。
- JEG 2.6 へのアップグレードにより、カーネル管理が非同期になりました。つまり、カーネルの起動中、JEG はトランザクションをブロックしません。この結果、以下の機能が利用できるようになり、ユーザーエクスペリエンスが大幅に向上しています。
  - 他のカーネルが起動中のときに現在実行中のノートブックでコマンドを実行できる機能
  - 既に実行中のカーネルに影響を与えることなく複数のカーネルを同時に起動できる機能
- サポートされている設定分類 :

分類	説明
core-site	Hadoop core-site.xml ファイル内の値を変更します。
emrfs-site	EMRFS の設定を変更します。
spark-metrics	Spark metrics.properties ファイル内の値を変更します。
spark-defaults	Spark spark-defaults.conf ファイル内の値を変更します。
spark-env	Spark 環境の値を変更します。
spark-hive-site	Spark hive-site.xml ファイル内の値を変更します。
spark-log4j	Spark log4j.properties ファイル内の値を変更します。

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。これらは多くの場合、spark-hive-site.xml などのアプリケーションの構成 XML ファイルに対応します。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。

解決された問題

- Amazon EMR on EKS 6.7 では、インタラクティブエンドポイントで Apache Spark のポッドテンプレート機能を使用すると発生していた 6.6 の問題を修正しています。Amazon EMR on EKS リリース 6.4、6.5、6.6 でも、同じ問題が発生していました。インタラクティブエンドポイントを使用してインタラクティブな分析を実行するときに、ポッドテンプレートを使用して Spark ドライバーポッドとエグゼキュターポッドをどのように起動するかを定義できるようになりました。
- 以前の Amazon EMR on EKS リリースでは、カーネルの起動中は Jupyter Enterprise Gateway がトランザクションをブロックしていたため、現在実行中のノートブックセッションが阻害されていました。他のカーネルが起動中のときに、現在実行中のノートブックでコマンドを実行できるようになりました。また、既に実行中のカーネルとの接続を失うことなく、複数のカーネルを同時に起動することもできます。

emr-6.7.0-latest

リリースノート : 現在 `emr-6.7.0-latest` が指しているのは `emr-6.7.0-20240321` です。

リージョン: `emr-6.7.0-latest` は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: `emr-6.7.0:latest`

emr-6.7.0-20240321

リリースノート: `emr-6.7.0-20240321` は、2024 年 3 月 11 日にリリースされました。以前のリリースと比較すると、このリリースは先ごろ更新された Amazon Linux パッケージと重大な修正で更新されています。

リージョン: `emr-6.7.0-20240321` は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: `emr-6.7.0:20240321`

emr-6.7.0-20230624

リリースノート: `emr-6.7.0-20230624` は、2023 年 7 月 7 日にリリースされました。以前のバージョンと比較すると、このバージョンは先ごろ更新された Amazon Linux パッケージと重大な修正で更新されています。

リージョン: emr-6.7.0-20230624 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-6.7.0:20230624

emr-6.7.0-20221219

リリースノート: emr-6.7.0-20221219 は、2023 年 1 月 19 日にリリースされました。以前のバージョンと比較すると、このバージョンは先ごろ更新された Amazon Linux パッケージと重大な修正で更新されています。

リージョン: emr-6.7.0-20221219 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-6.7.0:20221219

emr-6.7.0-20220630

リリースノート: emr-6.7.0-20220630 は、2022 年 7 月 12 日にリリースされました。これは Amazon EMR 6.7.0 の初期リリースです。

リージョン: emr-6.7.0-20220630 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-6.7.0:20220630

Amazon EMR on EKS 6.6.0 リリース

Amazon EMR on EKS では、次の Amazon EMR 6.6.0 リリースが利用可能です。特定の emr-6.6.0-XXXX リリースを選択すると、関連するコンテナイメージタグなどの詳細が表示されます。

- emr-6.6.0-latest
- emr-6.6.0-20240321
- emr-6.6.0-20230624
- emr-6.6.0-20221219
- emr-6.6.0-20220411

Amazon EMR 6.6.0 のリリースノート

- サポートされているアプリケーション - Spark 3.2.0-amzn-0、Jupyter Enterprise Gateway (エンドポイント、パブリックプレビュー)、Hudi 0.10.1-amzn-0、Iceberg 0.13.1。
- サポートされるコンポーネント`-aws-hm-client` (Glue コネクタ) 、`aws-sagemaker-spark-sdk`、`emr-s3-select`、`emrfs`、`emr-ddb`、`hudi-spark`。
- サポートされている設定分類 :

分類	説明
`core-site`	Hadoop の core-site.xml ファイルの値を変更します。
`emrfs-site`	EMRFS の設定を変更します。
`spark-metrics`	Spark の metrics.properties ファイルの値を変更します。
`spark-defaults`	Spark の spark-defaults.conf ファイルの値を変更します。
`spark-env`	Spark 環境の値を変更します。
`spark-hive-site`	Spark の hive-site.xml ファイルの値を変更します
`spark-log4j`	Spark の log4j.properties 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。これらは多くの場合、spark-hive-site.xml などのアプリケーションの構成 XML ファイルに対応します。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。

既知の問題

- Amazon EMR on EKS リリース 6.4、6.5、6.6 では、インタラクティブエンドポイントで Spark ポッドテンプレート機能を使用できません。

解決された問題

- インタラクティブエンドポイントログが、Cloudwatch と S3 にアップロードされます。

emr-6.6.0-latest

リリースノート : 現在 `emr-6.6.0-latest` が指しているのは `emr-6.6.0-20240321` です。

リージョン: `emr-6.6.0-latest` は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: `emr-6.6.0:latest`

emr-6.6.0-20240321

リリースノート: `emr-6.6.0-20240321` は、2024 年 3 月 11 日にリリースされました。以前のリリースと比較すると、このリリースは先ごろ更新された Amazon Linux パッケージと重大な修正で更新されています。

リージョン: `emr-6.6.0-20240321` は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: `emr-6.6.0:20240321`

emr-6.6.0-20230624

リリースノート: `emr-6.6.0-20230624` は、2023 年 1 月 27 日にリリースされました。以前のバージョンと比較すると、このバージョンは先ごろ更新された Amazon Linux パッケージと重大な修正で更新されています。

リージョン: `emr-6.6.0-20230624` は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: `emr-6.6.0:20230624`

emr-6.6.0-20221219

リリースノート: emr-6.6.0-20221219 は、2023 年 1 月 27 日にリリースされました。以前のバージョンと比較すると、このバージョンは先ごろ更新された Amazon Linux パッケージと重大な修正で更新されています。

リージョン: emr-6.6.0-20221219 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-6.6.0:20221219

emr-6.6.0-20220411

リリースノート: emr-6.6.0-20220411 は、2022 年 5 月 20 日にリリースされました。これは Amazon EMR 6.6.0 の初期リリースです。

リージョン: emr-6.6.0-20220411 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-6.6.0:20220411

Amazon EMR on EKS 6.5.0 リリース

Amazon EMR on EKS では、次の Amazon EMR 6.5.0 リリースが利用可能です。特定の emr-6.5.0-XXXX リリースを選択すると、関連するコンテナイメージタグなどの詳細が表示されます。

- emr-6.5.0-latest
- emr-6.5.0-20240321
- emr-6.5.0-20221219
- emr-6.5.0-20220802
- emr-6.5.0-20211119

Amazon EMR 6.5.0 のリリースノート

- サポートされているアプリケーション - Spark 3.1.2-amzn-1、Jupyter Enterprise Gateway (エンドポイント、パブリックプレビュー)。

- サポートされるコンポーネント-aws-hm-client (Glue コネクタ) 、aws-sagemaker-spark-sdk、emr-s3-select、emrfs、emr-ddb、hudi-spark。
- サポートされている設定分類 :

分類	説明
core-site	Hadoop の core-site.xml ファイルの値を変更します。
emrfs-site	EMRFS の設定を変更します。
spark-metrics	Spark の metrics.properties ファイルの値を変更します。
spark-defaults	Spark の spark-defaults.conf ファイルの値を変更します。
spark-env	Spark 環境の値を変更します。
spark-hive-site	Spark の hive-site.xml ファイルの値を変更します
spark-log4j	Spark の log4j.properties 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。これらは多くの場合、spark-hive-site.xml などのアプリケーションの構成 XML ファイルに対応します。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。

既知の問題

- Amazon EMR on EKS リリース 6.4 と 6.5 では、インタラクティブエンドポイントで Spark ポッドテンプレート機能を使用できません。

emr-6.5.0-latest

リリースノート : 現在 emr-6.5.0-latest が指しているのは emr-6.5.0-20240321 です。

リージョン: emr-6.5.0-latest は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-6.5.0:latest

emr-6.5.0-20240321

リリースノート: emr-6.5.0-20240321 は、2024 年 3 月 11 日にリリースされました。以前のバージョンと比較すると、このバージョンは先ごろ更新された Amazon Linux パッケージと重大な修正で更新されています。

リージョン: emr-6.5.0-20240321 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-6.5.0:20240321

emr-6.5.0-20221219

リリースノート: emr-6.5.0-20221219 は、2023 年 1 月 19 日にリリースされました。以前のバージョンと比較すると、このバージョンは先ごろ更新された Amazon Linux パッケージと重大な修正で更新されています。

リージョン: emr-6.5.0-20221219 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-6.5.0:20221219

emr-6.5.0-20220802

リリースノート: emr-6.5.0-20220802 は、2022 年 8 月 24 日にリリースされました。以前のバージョンと比較すると、このバージョンは先ごろ更新された Amazon Linux パッケージで更新されています。

リージョン: emr-6.5.0-20220802 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-6.5.0:20220802

emr-6.5.0-20211119

リリースノート: emr-6.5.0-20211119 は、2022 年 1 月 20 日にリリースされました。これは Amazon EMR 6.5.0 の初期リリースです。

リージョン: emr-6.5.0-20211119 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-6.5.0:20211119

Amazon EMR on EKS 6.4.0 リリース

Amazon EMR on EKS では、次の Amazon EMR 6.4.0 リリースが利用可能です。特定の emr-6.4.0-XXXX リリースを選択すると、関連するコンテナイメージタグなどの詳細が表示されます。

- emr-6.4.0-latest
- emr-6.4.0-20240321
- emr-6.4.0-20221219
- emr-6.4.0-20210830

Amazon EMR 6.4.0 のリリースノート

- サポートされているアプリケーション - Spark 3.1.2-amzn-0、Jupyter Enterprise Gateway (エンドポイント、パブリックプレビュー)。
- サポートされるコンポーネント-aws-hm-client (Glue コネクタ) 、aws-sagemaker-spark-sdk、emr-s3-select、emrfs、emr-ddb、hudi-spark。
- サポートされている設定分類 :

分類	説明
core-site	Hadoop の core-site.xml ファイルの値を変更します。
emrfs-site	EMRFS の設定を変更します。

分類	説明
spark-metrics	Spark の metrics.properties ファイルの値を変更します。
spark-defaults	Spark の spark-defaults.conf ファイルの値を変更します。
spark-env	Spark 環境の値を変更します。
spark-hive-site	Spark の hive-site.xml ファイルの値を変更します
spark-log4j	Spark の log4j.properties 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。これらは多くの場合、spark-hive-site.xml などのアプリケーションの構成 XML ファイルに対応します。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。

既知の問題

- Amazon EMR on EKS リリース 6.4 では、インタラクティブエンドポイントで Spark ポッドテンプレート機能を使用できません。

emr-6.4.0-latest

リリースノート : 現在 `emr-6.4.0-latest` が指しているのは `emr-6.4.0-20240321` です。

リージョン: `emr-6.4.0-latest` は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: `emr-6.4.0:latest`

emr-6.4.0-20240321

リリースノート: emr-6.4.0-20240321 は、2024 年 3 月 11 日にリリースされました。以前のリリースと比較すると、このリリースは先ごろ更新された Amazon Linux パッケージと重大な修正で更新されています。

リージョン: emr-6.4.0-20240321 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-6.4.0:20240321

emr-6.4.0-20221219

リリースノート: emr-6.4.0-20221219 は、2023 年 1 月 27 日にリリースされました。以前のバージョンと比較すると、このバージョンは先ごろ追加された Amazon Linux パッケージで更新されています。

リージョン: emr-6.4.0-20221219 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-6.4.0:20221219

emr-6.4.0-20210830

リリースノート: emr-6.4.0-20210830 は、2021 年 12 月 9 日にリリースされました。これは Amazon EMR 6.4.0 の初期リリースです。

リージョン: emr-6.4.0-20210830 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-6.4.0:20210830

Amazon EMR on EKS 6.3.0 リリース

Amazon EMR on EKS では、次の Amazon EMR 6.3.0 リリースが利用可能です。特定の emr-6.3.0-XXXX リリースを選択すると、関連するコンテナイメージタグなどの詳細が表示されます。

- emr-6.3.0-最新
- emr-6.3.0-20240321
- emr-6.3.0-20220802
- emr-6.3.0-20211008
- emr-6.3.0-20210802
- emr-6.3.0-20210429

Amazon EMR 6.3.0 のリリースノート

- 新機能-6.x リリースシリーズの Amazon EMR 6.3.0 以降、Amazon EMR on EKS は Spark のポッドテンプレート機能をサポートしています。Amazon EMR on EKSの Spark イベントログローテーション機能をオンにすることもできます。詳細については、ポッドテンプレートの使用およびSpark イベントログのローテーションを使用するを参照してください。
- サポートされているアプリケーション-Spark 3.1.1-amzn-0、Jupyter エンタープライズゲートウェイ (エンドポイント、パブリックプレビュー) 。
- サポートされるコンポーネント-aws-hm-client (Glue コネクタ) 、aws-sagemaker-spark-sdk、emr-s3-select、emrfs、emr-ddb、hudi-spark。
- サポートされている設定分類 :

分類	説明
core-site	Hadoop の core-site.xml ファイルの値を変更します。
emrfs-site	EMRFS の設定を変更します。
spark-metrics	Spark の metrics.properties ファイルの値を変更します。
spark-defaults	Spark の spark-defaults.conf ファイルの値を変更します。
spark-env	Spark 環境の値を変更します。
spark-hive-site	Spark の hive-site.xml ファイルの値を変更します

分類	説明
spark-log4j	Spark の log4j.properties 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。これらは多くの場合、spark-hive-site.xml などのアプリケーションの構成 XML ファイルに対応します。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-6.3.0-最新

リリースノート : 現在 emr-6.3.0-latest が指しているのは emr-6.3.0-20240321 です。

リージョン: emr-6.3.0-latest は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-6.3.0:latest

emr-6.3.0-20240321

リリースノート: emr-6.3.0-20240321 は、2024 年 3 月 11 日にリリースされました。以前のリリースと比較すると、このリリースは先ごろ更新された Amazon Linux パッケージと重大な修正で更新されています。

リージョン: emr-6.3.0-20240321 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-6.3.0:20240321

emr-6.3.0-20220802

リリースノート: emr-6.3.0-20220802 は、2022 年 9 月 27 日にリリースされました。以前のバージョンと比較すると、このバージョンは先ごろ更新された Amazon Linux パッケージで更新されています。

リージョン: emr-6.3.0-20220802 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-6.3.0:20220802

emr-6.3.0-20211008

リリースノート: emr-6.3.0-20211008 は、2021 年 12 月 9 日にリリースされました。前回のバージョンと比較すると、このバージョンには問題の修正とセキュリティ更新プログラムが含まれています。

リージョン: emr-6.3.0-20211008 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-6.3.0:20211008

emr-6.3.0-20210802

リリースノート: emr-6.3.0-20210802 は、2021 年 8 月 2 日にリリースされました。前回のバージョンと比較すると、このバージョンには問題の修正とセキュリティ更新プログラムが含まれています。

リージョン: emr-6.3.0-20210802 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-6.3.0:20210802

emr-6.3.0-20210429

リリースノート : emr-6.3.0-20210429 は2021 年 4 月 29 日にリリースされました。これは Amazon EMR 6.3.0 の初期リリースです。

リージョン: emr-6.3.0-20210429 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-6.3.0:20210429

Amazon EMR on EKS 6.2.0 リリース

Amazon EMR on EKS では、次の Amazon EMR 6.2.0 リリースが利用可能です。特定の emr-6.2.0-XXXX リリースを選択すると、関連するコンテナイメージタグなどの詳細が表示されます。

- emr-6.2.0-最新
- emr-6.2.0-20240321
- emr-6.2.0-20220802
- emr-6.2.0-20211008
- emr-6.2.0-20210802
- emr-6.2.0-20210615
- emr-6.2.0-20210129
- emr-6.2.0-20201218
- emr-6.2.0-20201201

Amazon EMR 6.2.0 のリリースノート

- サポートされているアプリケーション-Spark 3.0.1-amzn-0、Jupyter エンタープライズゲートウェイ (エンドポイント、パブリックプレビュー) 。
- サポートされるコンポーネント-`aws-hm-client` (Glue コネクタ) 、`aws-sagemaker-spark-sdk`、`emr-s3-select`、`emrfs`、`emr-ddb`、`hudi-spark`。
- サポートされている設定分類 :

分類	説明
`core-site`	Hadoop の core-site.xml ファイルの値を変更します。
`emrfs-site`	EMRFS の設定を変更します。
`spark-metrics`	Spark の metrics.properties ファイルの値を変更します。
`spark-defaults`	Spark の spark-defaults.conf ファイルの値を変更します。

分類	説明
spark-env	Spark 環境の値を変更します。
spark-hive-site	Spark の hive-site.xml ファイルの値を変更します
spark-log4j	Spark の log4j.properties 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。これらは多くの場合、spark-hive-site.xml などのアプリケーションの構成 XML ファイルに対応します。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-6.2.0-最新

リリースノート : 現在 emr-6.2.0-latest が指しているのは emr-6.2.0-20240321 です。

リージョン: emr-6.2.0-latest は、Amazon EMR で EKS でサポートされているすべてのリージョンで使用可能です。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-6.2.0:20240321

emr-6.2.0-20240321

リリースノート: emr-6.2.0-20240321 は、2024 年 3 月 11 日にリリースされました。以前のリリースと比較すると、このリリースは先ごろ更新された Amazon Linux パッケージと重大な修正で更新されています。

リージョン: emr-6.2.0-20240321 は、Amazon EMR で EKS でサポートされているすべてのリージョンで使用可能です。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-6.2.0:20240321

emr-6.2.0-20220802

リリースノート: emr-6.2.0-20220802 は、2022 年 9 月 27 日にリリースされました。以前のバージョンと比較すると、このバージョンは先ごろ更新された Amazon Linux パッケージで更新されています。

リージョン: emr-6.2.0-20220802 は、Amazon EMR で EKS でサポートされているすべてのリージョンで使用可能です。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-6.2.0:20220802

emr-6.2.0-20211008

リリースノート: emr-6.2.0-20211008 は、2021 年 12 月 9 日にリリースされました。前回のバージョンと比較すると、このバージョンには問題の修正とセキュリティ更新プログラムが含まれています。

リージョン : emr-6.2.0-20211008 は、米国東部 (バージニア北部)、米国西部 (オレゴン)、アジアパシフィック (東京)、欧州 (アイルランド)、南米 (サンパウロ) の各リージョンでご利用いただけます。

コンテナイメージタグ: emr-6.2.0:20211008

emr-6.2.0-20210802

リリースノート: emr-6.2.0-20210802 は、2021 年 8 月 2 日にリリースされました。前回のバージョンと比較すると、このバージョンには問題の修正とセキュリティ更新プログラムが含まれています。

リージョン : emr-6.2.0-20210802 は、米国東部 (バージニア北部)、米国西部 (オレゴン)、アジアパシフィック (東京)、欧州 (アイルランド)、南米 (サンパウロ) の各リージョンでご利用いただけます。

コンテナイメージタグ: emr-6.2.0:20210802

emr-6.2.0-20210615

リリースノート: emr-6.2.0-20210615 は、2021 年 6 月 15 日にリリースされました。前回のバージョンと比較すると、このバージョンには問題の修正とセキュリティ更新プログラムが含まれています。

リージョン : emr-6.2.0-20210615 は、米国東部 (バージニア北部)、米国西部 (オレゴン)、アジアパシフィック (東京)、欧州 (アイルランド)、南米 (サンパウロ) の各リージョンでご利用いただけます。

コンテナイメージタグ: emr-6.2.0:20210615

emr-6.2.0-20210129

リリースノート : emr-6.2.0-20210129 2021 年 1 月 29 日にリリースされました。emr-6.2.0-20201218 と比較すると、このバージョンには問題の修正とセキュリティ更新プログラムが含まれています。

リージョン : emr-6.2.0-20210129 は、米国東部 (バージニア北部)、米国西部 (オレゴン)、アジアパシフィック (東京)、欧州 (アイルランド)、南米 (サンパウロ) の各リージョンでご利用いただけます。

コンテナイメージタグ: emr-6.2.0-20210129

emr-6.2.0-20201218

リリースノート: emr-6.2.0-20201218 は 2020 年 12 月 18 日にリリースされました。emr-6.2.0-20201201 と比較すると、このバージョンには問題の修正とセキュリティ更新プログラムが含まれています。

リージョン : emr-6.2.0-20201218 は、米国東部 (バージニア北部)、米国西部 (オレゴン)、アジアパシフィック (東京)、欧州 (アイルランド)、南米 (サンパウロ) の各リージョンでご利用いただけます。

コンテナイメージタグ: emr-6.2.0-20201218

emr-6.2.0-20201201

リリースノート : emr-6.2.0-20201201 2020 年 12 月 1 日にリリースされました。これは Amazon EMR 6.2.0 の初期リリースです。

リージョン : emr-6.2.0-20201201 は、米国東部 (バージニア北部)、米国西部 (オレゴン)、アジアパシフィック (東京)、欧州 (アイルランド)、南米 (サンパウロ) の各リージョンでご利用いただけます。

コンテナイメージタグ: emr-6.2.0-20201201

Amazon EMR on EKS 5.36.0 リリース

Amazon EMR on EKS では、次の Amazon EMR 5.36.0 リリースが利用可能です。特定の emr-5.36.0-XXXX リリースを選択すると、関連するコンテナイメージタグなどの詳細が表示されます。

- emr-5.36.0-latest
- emr-5.36.0-20240321
- emr-5.36.0-20221219
- emr-5.36.0-20220620
- emr-5.36.0-20220525

Amazon EMR 5.36.0 のリリースノート

- log4j2 のセキュリティ問題が修正されました。
- サポートされているアプリケーション - Spark 2.4.8-amzn-2、Jupyter Enterprise Gateway (エンドポイント、パブリックプレビュー、Scala カーネルはサポート対象外)、livy-0.7.1、fluentd-4.0.0。
- サポートされているコンポーネント - aws-hm-client、aws-sagemaker-spark-sdk、emr-ddb、emr-goodies、emr-kinesis、kerberos-server。
- サポートされている設定分類 :

分類	説明
`core-site`	Hadoop の core-site.xml ファイルの値を変更します。
`emrfs-site`	EMRFS の設定を変更します。
`spark-metrics`	Spark の metrics.properties ファイルの値を変更します。
`spark-defaults`	Spark の spark-defaults.conf ファイルの値を変更します。
`spark-env`	Spark 環境の値を変更します。

分類	説明
spark-hive-site	Spark の hive-site.xml ファイルの値を変更します
spark-log4j	Spark の log4j.properties 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。これらは多くの場合、spark-hive-site.xml などのアプリケーションの構成 XML ファイルに対応します。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-5.36.0-latest

リリースノート : 現在 emr-5.36.0-latest が指しているのは emr-5.36.0-20240321 です。

リージョン: emr-5.36.0-latest は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-5.36.0:latest

emr-5.36.0-20240321

リリースノート: emr-5.36.0-20240321 は、2024 年 3 月 11 日にリリースされました。以前のリリースと比較すると、このリリースは先ごろ更新された Amazon Linux パッケージと重大な修正で更新されています。

リージョン: emr-5.36.0-20240321 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-5.36.0:20240321

emr-5.36.0-20221219

リリースノート: emr-5.36.0-20221219 は、2023 年 1 月 27 日にリリースされました。以前のバージョンと比較すると、このバージョンは先ごろ更新された Amazon Linux パッケージで更新されています。

リージョン: emr-5.36.0-20221219 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-5.36.0:20221219

emr-5.36.0-20220620

リリースノート: emr-5.36.0-20220620 は、2022 年 7 月 27 日にリリースされました。以前のバージョンと比較すると、このバージョンは先ごろ更新された Amazon Linux パッケージで更新されています。

リージョン: emr-5.36.0-20220620 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-5.36.0:20220620

emr-5.36.0-20220525

リリースノート: emr-5.36.0-20220525 は、2022 年 6 月 16 日にリリースされました。これは Amazon EMR 5.36.0 の初期リリースです。

リージョン: emr-5.36.0-20220525 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-5.36.0:20220525

Amazon EMR on EKS 5.35.0 リリース

Amazon EMR on EKS では、次の Amazon EMR 5.35.0 リリースが利用可能です。特定の emr-5.35.0-XXXX リリースを選択すると、関連するコンテナイメージタグなどの詳細が表示されます。

- emr-5.35.0-latest
- emr-5.35.0-20240321
- emr-5.35.0-20221219

- emr-5.35.0-20220802
- emr-5.35.0-20220307

Amazon EMR 5.35.0 のリリースノート

- log4j2 のセキュリティ問題が修正されました。
- サポートされているアプリケーション - Spark 2.4.8-amzn-1、Hudi 0.9.0-amzn-2、Jupyter Enterprise Gateway (エンドポイント、パブリックプレビュー、Scala カーネルはサポート対象外)。
- サポートされているコンポーネント - aws-hm-client (Glue コネクタ)、aws-sagemaker-spark-sdk、emr-s3-select、emrfs、emr-ddb、hudi-spark。
- サポートされている設定分類 :

分類	説明
core-site	Hadoop の core-site.xml ファイルの値を変更します。
emrfs-site	EMRFS の設定を変更します。
spark-metrics	Spark の metrics.properties ファイルの値を変更します。
spark-defaults	Spark の spark-defaults.conf ファイルの値を変更します。
spark-env	Spark 環境の値を変更します。
spark-hive-site	Spark の hive-site.xml ファイルの値を変更します
spark-log4j	Spark の log4j.properties 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。これらは多くの場合、spark-hive-site.xml などのアプリケーションの構成 XML ファイルに対応します。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-5.35.0-latest

リリースノート : 現在 emr-5.35.0-latest が指しているのは emr-5.35.0-20240321 です。

リージョン: emr-5.35.0-latest は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-5.35.0:latest

emr-5.35.0-20240321

リリースノート: emr-5.35.0-20240321 は、2024 年 3 月 11 日にリリースされました。以前のリリースと比較すると、このリリースは先ごろ更新された Amazon Linux パッケージと重大な修正で更新されています。

リージョン: emr-5.35.0-20240321 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-5.35.0:20240321

emr-5.35.0-20221219

リリースノート: emr-5.35.0-20221219 は、2023 年 1 月 27 日にリリースされました。以前のバージョンと比較すると、このバージョンは先ごろ更新された Amazon Linux パッケージで更新されています。

リージョン: emr-5.35.0-20221219 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-5.35.0:20221219

emr-5.35.0-20220802

リリースノート: emr-5.35.0-20220802 は、2022 年 9 月 27 日にリリースされました。以前のバージョンと比較すると、このバージョンは先ごろ更新された Amazon Linux パッケージで更新されています。

リージョン: emr-5.35.0-20220802 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-5.35.0:20220802

emr-5.35.0-20220307

リリースノート: emr-5.35.0-20220307 は、2022 年 3 月 30 日にリリースされました。以前のバージョンと比較すると、このバージョンは先ごろ更新された Amazon Linux パッケージで更新されています。

リージョン: emr-5.35.0-20220307 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-5.35.0:20220307

Amazon EMR on EKS 5.34.0 リリース

Amazon EMR on EKS では、次の Amazon EMR 5.34.0 リリースが利用可能です。特定の emr-5.34.0-XXXX リリースを選択すると、関連するコンテナイメージタグなどの詳細が表示されます。

- emr-5.34.0-latest
- emr-5.34.0-20240321
- emr-5.34.0-20220802

Amazon EMR 5.34.0 のリリースノート

- サポートされているアプリケーション - Spark 2.4.8-amzn-0、Jupyter Enterprise Gateway (エンドポイント、パブリックプレビュー、Scala カーネルはサポート対象外)。
- サポートされるコンポーネント-aws-hm-client (Glue コネクタ) 、aws-sagemaker-spark-sdk、emr-s3-select、emrfs、emr-ddb、hudi-spark。
- サポートされている設定分類 :

分類	説明
core-site	Hadoop の core-site.xml ファイルの値を変更します。
emrfs-site	EMRFS の設定を変更します。
spark-metrics	Spark の metrics.properties ファイルの値を変更します。
spark-defaults	Spark の spark-defaults.conf ファイルの値を変更します。
spark-env	Spark 環境の値を変更します。
spark-hive-site	Spark の hive-site.xml ファイルの値を変更します
spark-log4j	Spark の log4j.properties 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。これらは多くの場合、spark-hive-site.xml などのアプリケーションの構成 XML ファイルに対応します。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-5.34.0-latest

リリースノート : 現在 emr-5.34.0-latest が指しているのは emr-5.34.0-20220802 です。

リージョン: emr-5.34.0-latest は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-5.34.0:latest

emr-5.34.0-20240321

リリースノート: emr-5.34.0-20240321 は、2024 年 3 月 11 日にリリースされました。以前のリリースと比較すると、このリリースは先ごろ更新された Amazon Linux パッケージと重大な修正で更新されています。

リージョン: emr-5.34.0-20240321 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-5.34.0:20240321

emr-5.34.0-20220802

リリースノート: emr-5.34.0-20220802 は、2022 年 8 月 24 日にリリースされました。以前のバージョンと比較すると、このバージョンは先ごろ更新された Amazon Linux パッケージで更新されています。

リージョン: emr-5.34.0-20220802 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-5.34.0:20220802

emr-5.34.0-20211208

リリースノート: emr-5.34.0-20211208 は、2022 年 1 月 20 日にリリースされました。以前のバージョンと比較すると、このバージョンは先ごろ更新された Amazon Linux パッケージで更新されています。

リージョン: emr-5.34.0-20211208 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-5.34.0:20211208

Amazon EMR on EKS 5.33.0 リリース

Amazon EMR on EKS では、次の Amazon EMR 5.33.0 リリースが利用可能です。特定の emr-5.33.0-XXXX リリースを選択すると、関連するコンテナイメージタグなどの詳細が表示されます。

- emr-5.33.0-最新
- emr-5.33.0-20240321
- emr-5.33.0-20221219
- emr-5.33.0-20220802
- emr-5.33.0-20211008
- emr-5.33.0-20210802
- emr-5.33.0-20210615
- emr-5.33.0-20210323

Amazon EMR 5.33.0 のリリースノート

- 新機能-5.x リリースシリーズの Amazon EMR 5.33.0 以降、Amazon EMR on EKS は Spark のポッドテンプレート機能をサポートしています。詳細については、「ポッドテンプレートの使用」を参照してください。
- サポートされているアプリケーション - Spark 2.4.7-amzn-1、Jupyter Enterprise Gateway (エンドポイント、パブリックプレビュー、Scala カーネルはサポート対象外)。
- サポートされるコンポーネント-`aws-hm-client` (Glue コネクタ) 、`aws-sagemaker-spark-sdk`、`emr-s3-select`、`emrfs`、`emr-ddb`、`hudi-spark`。
- サポートされている設定分類 :

分類	説明
`core-site`	Hadoop の core-site.xml ファイルの値を変更します。
`emrfs-site`	EMRFS の設定を変更します。
`spark-metrics`	Spark の metrics.properties ファイルの値を変更します。
`spark-defaults`	Spark の spark-defaults.conf ファイルの値を変更します。
`spark-env`	Spark 環境の値を変更します。

分類	説明
spark-hive-site	Spark の hive-site.xml ファイルの値を変更します
spark-log4j	Spark の log4j.properties 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。これらは多くの場合、spark-hive-site.xml などのアプリケーションの構成 XML ファイルに対応します。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-5.33.0-最新

リリースノート : 現在 emr-5.33.0-latest が指しているのは emr-5.33.0-20240321 です。

リージョン: emr-5.33.0-latest は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-5.33.0:latest

emr-5.33.0-20240321

リリースノート: emr-5.33.0-20240321 は、2024 年 3 月 11 日にリリースされました。以前のリリースと比較すると、このリリースは先ごろ更新された Amazon Linux パッケージと重大な修正で更新されています。

リージョン: emr-5.33.0-20240321 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-5.33.0:20240321

emr-5.33.0-20221219

リリースノート: emr-5.33.0-20221219 は、2023 年 1 月 19 日にリリースされました。以前のバージョンと比較すると、このバージョンは先ごろ更新された Amazon Linux パッケージと重大な修正で更新されています。

リージョン: emr-5.33.0-20221219 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-5.33.0:20221219

emr-5.33.0-20220802

リリースノート: emr-5.33.0-20220802 は、2022 年 8 月 24 日にリリースされました。以前のバージョンと比較すると、このバージョンは先ごろ更新された Amazon Linux パッケージで更新されています。

リージョン: emr-5.33.0-20220802 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-5.33.0:20220802

emr-5.33.0-20211008

リリースノート: emr-5.33.0-20211008 は、2021 年 12 月 9 日にリリースされました。前回のバージョンと比較すると、このバージョンには問題の修正とセキュリティ更新プログラムが含まれています。

リージョン: emr-5.33.0-20211008 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-5.33.0:20211008

emr-5.33.0-20210802

リリースノート: emr-5.33.0-20210802 は、2021 年 8 月 2 日にリリースされました。前回のバージョンと比較すると、このバージョンには問題の修正とセキュリティ更新プログラムが含まれています。

リージョン: emr-5.33.0-20210802 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-5.33.0:20210802

emr-5.33.0-20210615

リリースノート: emr-5.33.0-20210615 は、2021 年 6 月 15 日にリリースされました。前回のバージョンと比較すると、このバージョンには問題の修正とセキュリティ更新プログラムが含まれています。

リージョン: emr-5.33.0-20210615 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-5.33.0:20210615

emr-5.33.0-20210323

リリースノート : emr-5.33.0-20210323 は、2021 年 3 月 23 日にリリースされました。これは Amazon EMR 5.33.0 の初期リリースです。

リージョン: emr-5.33.0-20210323 は、Amazon EMR で EKS でサポートされているすべてのリージョンでご利用いただけます。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-5.33.0-20210323

Amazon EMR on EKS 5.32.0 リリース

Amazon EMR on EKS では、次の Amazon EMR 5.32.0 リリースが利用可能です。特定の emr-5.32.0-XXXX リリースを選択すると、関連するコンテナイメージタグなどの詳細が表示されます。

- emr-5.32.0-最新
- emr-5.32.0-20240321
- emr-5.32.0-20220802
- emr-5.32.0-20211008
- emr-5.32.0-20210802
- emr-5.32.0-20210615

- emr-5.32.0-20210129
- emr-5.32.0-20201218
- emr-5.32.0-20201201

Amazon EMR 5.32.0 のリリースノート

- サポートされているアプリケーション - Spark 2.4.7-amzn-0、Jupyter Enterprise Gateway (エンドポイント、パブリックプレビュー、Scala カーネルはサポート対象外)。
- サポートされるコンポーネント`-aws-hm-client` (Glue コネクタ) 、`aws-sagemaker-spark-sdk`、`emr-s3-select`、`emrfs`、`emr-ddb`、`hudi-spark`。
- サポートされている設定分類 :

分類	説明
`core-site`	Hadoop の core-site.xml ファイルの値を変更します。
`emrfs-site`	EMRFS の設定を変更します。
`spark-metrics`	Spark の metrics.properties ファイルの値を変更します。
`spark-defaults`	Spark の spark-defaults.conf ファイルの値を変更します。
`spark-env`	Spark 環境の値を変更します。
`spark-hive-site`	Spark の hive-site.xml ファイルの値を変更します
`spark-log4j`	Spark の log4j.properties 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。これらは多くの場合、spark-hive-site.xml などのアプリケーションの構成 XML ファイルに対応します。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-5.32.0-最新

リリースノート : 現在 emr-5.32.0-latest が指しているのは emr-5.32.0-20240321 です。

リージョン: emr-5.32.0-latest は、Amazon EMR で EKS でサポートされているすべてのリージョンで使用可能です。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-5.32.0:latest

emr-5.32.0-20240321

リリースノート: emr-5.32.0-20240321 は、2024 年 3 月 11 日にリリースされました。以前のリリースと比較すると、このリリースは先ごろ更新された Amazon Linux パッケージと重大な修正で更新されています。

リージョン: emr-5.32.0-20240321 は、Amazon EMR で EKS でサポートされているすべてのリージョンで使用可能です。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-5.32.0:20240321

emr-5.32.0-20220802

リリースノート: emr-5.32.0-20220802 は、2022 年 8 月 24 日にリリースされました。以前のバージョンと比較すると、このバージョンは先ごろ更新された Amazon Linux パッケージで更新されています。

リージョン: emr-5.32.0-20220802 は、Amazon EMR で EKS でサポートされているすべてのリージョンで使用可能です。詳細については、「Amazon EMR on EKS サービスエンドポイント」を参照してください。

コンテナイメージタグ: emr-5.32.0:20220802

emr-5.32.0-20211008

リリースノート: emr-5.32.0-20211008 は、2021 年 12 月 9 日にリリースされました。前回のバージョンと比較すると、このバージョンには問題の修正とセキュリティ更新プログラムが含まれています。

リージョン : emr-5.32.0-20211008 は、米国東部 (バージニア北部)、米国西部 (オレゴン)、アジアパシフィック (東京)、欧州 (アイルランド)、南米 (サンパウロ) の各リージョンでご利用いただけます。

コンテナイメージタグ: emr-5.32.0:20211008

emr-5.32.0-20210802

リリースノート: emr-5.32.0-20210802 は、2021 年 8 月 2 日にリリースされました。前回のバージョンと比較すると、このバージョンには問題の修正とセキュリティ更新プログラムが含まれています。

リージョン : emr-5.32.0-20210802 は、米国東部 (バージニア北部)、米国西部 (オレゴン)、アジアパシフィック (東京)、欧州 (アイルランド)、南米 (サンパウロ) の各リージョンでご利用いただけます。

コンテナイメージタグ: emr-5.32.0:20210802

emr-5.32.0-20210615

リリースノート: emr-5.32.0-20210615 は、2021 年 6 月 15 日にリリースされました。前回のバージョンと比較すると、このバージョンには問題の修正とセキュリティ更新プログラムが含まれています。

リージョン : emr-5.32.0-20210615 は、米国東部 (バージニア北部)、米国西部 (オレゴン)、アジアパシフィック (東京)、欧州 (アイルランド)、南米 (サンパウロ) の各リージョンでご利用いただけます。

コンテナイメージタグ: emr-5.32.0:20210615

emr-5.32.0-20210129

リリースノート : emr-5.32.0-20210129 2021 年 1 月 29 日にリリースされました。emr-5.32.0-20201218 と比較すると、このバージョンには問題の修正とセキュリティ更新プログラムが含まれています。

リージョン : emr-5.32.0-20210129 は、米国東部 (バージニア北部)、米国西部 (オレゴン)、アジアパシフィック (東京)、欧州 (アイルランド)、南米 (サンパウロ) の各リージョンでご利用いただけます。

コンテナイメージタグ: emr-5.32.0-20210129

emr-5.32.0-20201218

リリースノート : 5.32.0-20201218 2020 年 12 月 18 日にリリースされました。5.32.0-20201201 と比較すると、このバージョンには問題の修正とセキュリティ更新プログラムが含まれています。

リージョン : emr-5.32.0-20201218 は、米国東部 (バージニア北部)、米国西部 (オレゴン)、アジアパシフィック (東京)、欧州 (アイルランド)、南米 (サンパウロ) の各リージョンでご利用いただけます。

コンテナイメージタグ: emr-5.32.0-20201218

emr-5.32.0-20201201

リリースノート : 5.32.0-20201201 2020 年 12 月 1 日にリリースされました。これは Amazon EMR 5.32.0 の初期リリースです。

リージョン : 5.32.0-20201201d は、米国東部 (バージニア北部)、米国西部 (オレゴン)、アジアパシフィック (東京)、欧州 (アイルランド)、南米 (サンパウロ) の各リージョンでご利用いただけます。

コンテナイメージタグ: emr-5.32.0-20201201

ドキュメント履歴

次の表に、EKS での Amazon EMR の前回のリリース以後に行われたドキュメントの重要な変更を示します。このドキュメントの更新情報は、RSS フィードに加入して取得できます。

変更	説明	日付
更新内容	Amazon EMR on EKS の管理ポリシー – に対する追加のアクセス許可`AmazonEMRContainersServiceRolePolicy` 。	2025 年 2 月 3 日
新規リリース	Amazon EMR on EKS 7.6.0 リリース	2025年1月10日
新規リリース	Amazon EMR on EKS 7.5.0 リリース	2024 年 11 月 21 日
新規リリース	Amazon EMR on EKS 7.4.0 リリース	2024 年 11 月 13 日
新規リリース	Amazon EMR on EKS 7.3.0 リリース	2024 年 10 月 16 日
新規リリース	Amazon EMR on EKS 7.2.0 リリース	2024 年 7 月 25 日
新規リリース	Amazon EMR on EKS 7.1.0 リリース	2024 年 4 月 17 日
新規リリース	Amazon EMR on EKS 7.0.0 リリース	2023 年 12 月 22 日
新規リリース	Amazon EMR on EKS 6.15.0 リリース	2023 年 11 月 17 日
新規リリース	Amazon EMR on EKS 6.14.0 リリース	2023 年 10 月 17 日
更新内容	「マネージドエンドポイント」の名前をインタラクティブエンドポイント、インタラクティブエンドポイントの一般提供に変更	2023 年 9 月 29 日
新規リリース	Amazon EMR on EKS 6.13.0 リリース、および EKS での Amazon EMR を使用した Flink ジョブの実行 のパブリックプレビュードキュメント	2023 年 9 月 12 日

変更	説明	日付
新規リリース	Amazon EMR on EKS 6.12.0 リリース	2023 年 7 月 21 日
新しいコンテンツ	「Amazon EMR on EKS で Apache Spark のカスタムスケジューラとして Volcano を使用する方法」を追加	2023 年 6 月 13 日
新しいコンテンツ	「Amazon EMR on EKS で Apache Spark のカスタムスケジューラとして Volcano を使用する方法」を追加	2023 年 6 月 13 日
新しいコンテンツ	「Spark コンテナログのローテーションを使用する」を追加	2023 年 6 月 12 日
更新内容	Amazon ECR Public Gallery でベースイメージ情報を検索するためのカスタムイメージのドキュメントを更新しました。	2023 年 6 月 8 日
新規リリース	Amazon EMR on EKS 6.11.0 リリース	2023 年 6 月 8 日
新しいコンテンツ	Amazon EMR on EKS での Spark ジョブの実行 に Spark 演算子を使用して Spark ジョブを実行する を追加し、「Job Runs」セクション再編成しました。	2023 年 6 月 5 日
新しいコンテンツ	「Amazon EMR Spark ジョブで垂直的自動スケーリングを使用する」セクションと「セルフホスト型 Jupyter Notebook を使用する」セクションを追加しました。	2023 年 5 月 4 日
ドキュメント履歴ページ	EKS での Amazon EMR のドキュメント履歴ページを作成しました。	2023 年 3 月 13 日
マネージドポリシーのページ	EKS での Amazon EMR のマネージドポリシーのページを作成しました。	2023 年 3 月 13 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。