aws

ユーザーガイド

Amazon Managed Workflows for Apache Airflow

Amazon Managed Workflows for Apache Airflow: ユーザーガイド

Table of Contents

Amazon Managed Workflows for Apache Airflow とは何ですか？

Amazon Managed Workflows for Apache Airflow は、「Apache Airflow」用のマネージドオーケストレーションサービスで、クラウド上でデータパイプラインを大規模に設定・運用するために使用できます。Apache Airflow は、「ワークフロー」と呼ばれる一連のプロセスとタスクをプログラムで作成、スケジュール、監視するために使用されるオープンソースのツールです。Amazon MWAA を使用すると、スケーラビリティ、可用性、セキュリティのための基盤を管理する必要なく、Apache Airflow と Python を使用してワークフローを作成できます。Amazon MWAA は、ワークフロー実行容量をニーズに合わせて自動的にスケーリングし、Amazon MWAA は AWS セキュリティサービスと統合して、データへの高速かつ安全なアクセスを提供します。

コンテンツ

機能

- 自動エアフローセットアップ — Amazon MWAA 環境を作成するときに「Apache Airflow バージョン」を選択することで、Apache Airflow をすばやくセットアップできます。Amazon MWAA は、インターネット上でダウンロードできる同じ Apache Airflow ユーザーインターフェイスとオープンソースコードを使用して、Apache Airflow を自動的にセットアップします。
- 自動スケーリング — 環境内で稼働するワーカーの最小数と最大数を設定して、Apache Airflow ワーカーを自動的にスケーリングします。Amazon MWAA は環境内のワーカーを監視し、「自動スケーリングコンポーネント」を使用して需要を満たすワーカーを、定義したワーカーの最大数に達するまで追加します。
- 組み込み認証 – AWS Identity and Access Management (IAM) でアクセスコントロールポリシーを定義することで、Apache Airflow ウェブサーバーのロールベースの認証と認可を有効にします。Apache Airflow ワーカーは、 サービスへの安全なアクセスのためにこれらのポリシーを引き受けます。 AWS

- ビルトインセキュリティ — Apache Airflow ワーカーとスケジューラーは「Amazon MWAA の Amazon VPC」で実行されます。また、データは を使用して自動的に暗号化されるため AWS Key Management Service、環境はデフォルトで安全です。
- パブリックアクセスモードまたはプライベートアクセスモード — プライベートまたはパブリック「アクセスモード」を使用して Apache Airflow ウェブサーバーにアクセスします。[パブリックネットワーク] アクセスモードは、インターネット経由でアクセス可能な Apache Airflow ウェブサーバーの VPC エンドポイントを使用します。プライベートネットワークアクセスモードは、VPC からアクセス可能な Apache Airflow ウェブサーバーの VPC エンドポイントを使用します。どちらの場合も、Apache Airflow ユーザーのアクセスは、 AWS Identity and Access Management (IAM) および AWS SSO で定義したアクセスコントロールポリシーによって制御されます。
- アップグレードとパッチの効率化 — Amazon MWAA は Apache Airflow の新しいバージョンを定期的に提供しています。Amazon MWAA チームはこれらのバージョンに合わせてイメージを更新し、パッチを適用します。
- ワークフローモニタリング – 追加のサードパーティーツールを必要とせずに、Apache Airflow ログと Apache Airflow メトリクスを Amazon で表示 CloudWatch して、Apache Airflow タスクの遅延やワークフローエラーを特定します。Amazon MWAA は、環境メトリクスを自動的に送信し、有効になっている場合は Apache Airflow ログを に送信します CloudWatch。
- AWS 統合 — Amazon MWAA は、Amazon Athena AWS Batch 、、Amazon CloudWatch、Amazon DynamoDB AWS DataSync、Amazon EMR AWS Fargate、Amazon EKS、Amazon Data Firehose AWS Glue、 AWS LambdaAmazon Redshift、Amazon SQS、Amazon SNS、Amazon S3 とのオープンソース統合に加えて SageMaker、組み込みおよびコミュニティで作成された数百の演算子とセンサーをサポートしています。
- ワーカーフリート — Amazon MWAA は、コンテナを使用してワーカーフリートをオンデマンドでスケーリングし、「AWS Fargate のAmazon ECS」を使用してでスケジューラーの停止を減らすことをサポートしています。Amazon ECS コンテナでタスクを呼び出すオペレーターと、Kubernetes クラスターでポッドを作成して実行する Kubernetes オペレーターがサポートされています。

アーキテクチャ

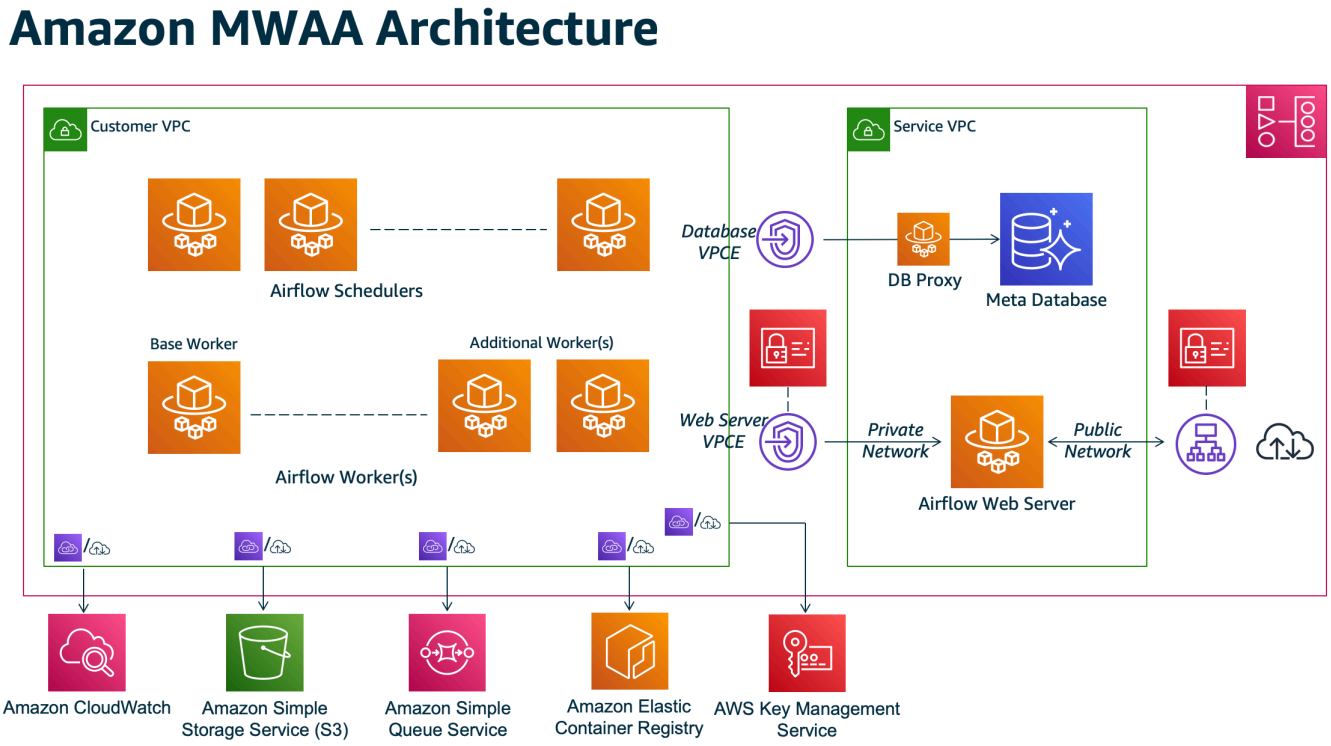
外側のボックス (下の画像) に含まれるすべてのコンポーネントは、アカウント内では単一の Amazon MWAA 環境として表示されます。Apache Airflow スケジューラとワーカーは、環境の Amazon VPC 内のプライベートサブネットに接続する AWS Fargate (Fargate) コンテナです。各環境には、 によって管理 AWS される独自の Apache Airflow メタデータベースがあり、プライベート

に保護された VPC エンドポイントを介してスケジューラおよびワーカー Fargate コンテナにアクセスできます。

Amazon CloudWatch、Amazon S3、Amazon SQS、Amazon ECR、および AWS KMS は Amazon MWAA とは別のものであり、Fargate コンテナ内の Apache Airflow スケジューラ (1 つ) とワーカーからアクセスできる必要があります。

Apache Airflow ウェブサーバーには、[パブリックネットワーク] の Apache Airflow アクセスモードを選択してインターネット経由でアクセスすることも、[プライベートネットワーク] の Apache Airflow アクセスモードを選択して VPC 内からアクセスすることもできます。どちらの場合も、Apache Airflow ユーザーのアクセスは、 AWS Identity and Access Management (IAM) で定義したアクセスコントロールポリシーによって制御されます。

> **Note**
>
> 複数の Apache Airflow スケジューラーは Apache Airflow v2 以上でのみ使用できます。Apache Airflow タスクライフサイクルの詳細については、Apache Airflow リファレンスガイドの 「概念」 を参照してください。



Integration

アクティブで増加している Apache Airflow オープンソースコミュニティは、Apache Airflow が サービスと統合するためのオペレーター (サービスへの接続を簡素化するプラグイン) を提供します AWS 。これには、Amazon S3、Amazon Redshift、Amazon EMR AWS Batch、Amazon などのサービス SageMaker、および他のクラウドプラットフォーム上のサービスが含まれます。

Amazon MWAA で Apache Airflow を使用すると、データ処理タスクを実行するための AWS サービスや、Apache Hadoop、Presto、Hive、Spark などの一般的なサードパーティーツールとの統合が完全にサポートされます。Amazon MWAA は、Amazon MWAA API との互換性を維持することに重点を置いており、Amazon MWAA は AWS サービスへの信頼性の高い統合を提供し、コミュニティで利用できるようにし、コミュニティの機能開発に関与することを目的としています。

サンプルコードについては、「Amazon Managed Workflows for Apache Airflow」を参照してください。

サポートバージョン

Amazon MWAA は複数のバージョンの Apache Airflow をサポートしています。サポートされている Apache Airflow のバージョンと、各バージョンに含まれている Apache Airflow コンポーネントの詳細については、Apache Airflow 用 Amazon マネージドワークフロー上の Apache Airflow のバージョン を参照してください。

次のステップ

- Airflow DAGs およびサポートファイル用の Amazon S3 バケット、パブリックルーティングを備えた Amazon VPC、および の Amazon MWAA 環境を作成する単一の AWS CloudFormation テンプレートの使用を開始しますAmazon Managed Workflows for Apache Airflow のクイックスタートチュートリアル。
- Airflow DAG とサポートファイル用の Amazon S3 バケットを作成し、3 つの Amazon VPC ネットワーキングオプションから 1 つを選択し、Amazon Managed Workflows for Apache Airflowを使い始める で Amazon MWAA 環境を作成することで、段階的に開始できます。

Amazon Managed Workflows for Apache Airflow のクイックスタートチュートリアル

このクイックスタートチュートリアルでは、Amazon VPC インフラストラクチャ、 dagsフォルダを持つ Amazon S3 バケット、および Amazon Managed Workflows for Apache Airflow 環境を同時に作成する AWS CloudFormation テンプレートを使用します。

トピック

このチュートリアルでは、次の作業を行いました。

このチュートリアルでは、Amazon S3 に DAG をアップロードし、Apache Airflow で DAG を実行し、 でログを表示する 3 つの AWS Command Line Interface （ AWS CLI) コマンドについて説明します CloudWatch。最後に、Apache Airflow 開発チーム用の IAM ポリシーを作成する手順を詳しく説明します。

> **Note**
>
> このページの AWS CloudFormation テンプレートは、 で利用可能な最新バージョンの Apache Airflow 用の Amazon Managed Workflows for Apache Airflow 環境を作成します AWS CloudFormation。利用可能な最新バージョンは Apache Airflow v2.8.1 です。

このページの AWS CloudFormation テンプレートは以下を作成します。

- VPC インフラストラクチャ。このテンプレートは インターネット経由のパブリックルーティング を使用しています。それは `WebserverAccessMode: PUBLIC_ONLY` 内の Apache Airflow ウェブサーバー に パブリックネットワークアクセスモード を使用します。

- Amazon S3 バケット。このテンプレートは、dags フォルダー付きの Amazon S3 バケットを作成します。Amazon MWAA 用の Amazon S3 バケットの作成 で定義されているように、[バケットバージョニング] を有効にして、[すべてのパブリックアクセスをブロック] するように設定されています。
- Amazon MWAA 環境。このテンプレートは、Amazon S3 バケットの dagsフォルダに関連付けられている Amazon MWAA 環境、Amazon MWAA が使用する AWS のサービスへのアクセス許可を持つ実行ロール、および で定義されているように、 AWS 所有キー を使用した暗号化のデフォルトを作成しますAmazon MWAA 環境を作成する。 Amazon S3
- をCloudWatch ログに記録します。テンプレートは、 CloudWatch で定義されているように、Airflow スケジューラロググループ 、Airflow ウェブサーバーロググループ 、Airflow ワーカーロググループ 、Airflow DAG 処理ロググループ 、および Airflow タスクロググループ に対して、Apache Airflow が「INFO」レベルでログインできるようにしますAmazon での Airflow ログの表示 CloudWatch。

このチュートリアルでは、以下のタスクを完了します。

- DAG をアップロードして実行します。最新の Amazon MWAA がサポートする Apache Airflow バージョン用の Apache Airflow のチュートリアル DAG を Amazon S3 にアップロードし、DAG の追加と更新 で定義されているように Apache Airflow UI で実行します。
- ログの表示。で定義されているように、Airflow ウェブサーバーロググループを CloudWatch Logs で表示しますAmazon での Airflow ログの表示 CloudWatch。
- アクセスコントロールポリシーを作成します。Amazon MWAA 環境へのアクセス で定義されているように、Apache Airflow 開発チーム用にアクセスコントロールポリシーを IAM で作成します。

前提条件

AWS Command Line Interface （ AWS CLI) は、コマンドラインシェルのコマンドを使用して AWS サービスとやり取りできるオープンソースツールです。このページのステップを完了するには、以下のものが必要です。

- AWS CLI – バージョン 2 をインストールします。
- AWS CLI – を使用したクイック設定aws configure。

ステップ 1: AWS CloudFormation テンプレートをローカルに保存する

- 次のテンプレートの内容をコピーし、mwaa-public-network.yml としてローカルに保存します。「テンプレートをダウンロードする」 こともできます。

```
AWSTemplateFormatVersion: "2010-09-09"

Parameters:

  EnvironmentName:
    Description: An environment name that is prefixed to resource names
    Type: String
    Default: MWAAEnvironment

  VpcCIDR:
    Description: The IP range (CIDR notation) for this VPC
    Type: String
    Default: 10.192.0.0/16

  PublicSubnet1CIDR:
    Description: The IP range (CIDR notation) for the public subnet in the first
 Availability Zone
    Type: String
    Default: 10.192.10.0/24

  PublicSubnet2CIDR:
    Description: The IP range (CIDR notation) for the public subnet in the second
 Availability Zone
    Type: String
    Default: 10.192.11.0/24

  PrivateSubnet1CIDR:
    Description: The IP range (CIDR notation) for the private subnet in the first
 Availability Zone
    Type: String
    Default: 10.192.20.0/24
  PrivateSubnet2CIDR:
    Description: The IP range (CIDR notation) for the private subnet in the second
 Availability Zone
    Type: String
    Default: 10.192.21.0/24
```

```
  MaxWorkerNodes:
    Description: The maximum number of workers that can run in the environment
    Type: Number
    Default: 2
  DagProcessingLogs:
    Description: Log level for DagProcessing
    Type: String
    Default: INFO
  SchedulerLogsLevel:
    Description: Log level for SchedulerLogs
    Type: String
    Default: INFO
  TaskLogsLevel:
    Description: Log level for TaskLogs
    Type: String
    Default: INFO
  WorkerLogsLevel:
    Description: Log level for WorkerLogs
    Type: String
    Default: INFO
  WebserverLogsLevel:
    Description: Log level for WebserverLogs
    Type: String
    Default: INFO

Resources:

 ##################################################################################################
  # CREATE VPC

 ##################################################################################################

  VPC:
    Type: AWS::EC2::VPC
    Properties:
      CidrBlock: !Ref VpcCIDR
      EnableDnsSupport: true
      EnableDnsHostnames: true
      Tags:
        - Key: Name
          Value: MWAAEnvironment

  InternetGateway:
    Type: AWS::EC2::InternetGateway
```

```
  Properties:
    Tags:
      - Key: Name
        Value: MWAAEnvironment

InternetGatewayAttachment:
  Type: AWS::EC2::VPCGatewayAttachment
  Properties:
    InternetGatewayId: !Ref InternetGateway
    VpcId: !Ref VPC

PublicSubnet1:
  Type: AWS::EC2::Subnet
  Properties:
    VpcId: !Ref VPC
    AvailabilityZone: !Select [ 0, !GetAZs '' ]
    CidrBlock: !Ref PublicSubnet1CIDR
    MapPublicIpOnLaunch: true
    Tags:
      - Key: Name
        Value: !Sub ${EnvironmentName} Public Subnet (AZ1)

PublicSubnet2:
  Type: AWS::EC2::Subnet
  Properties:
    VpcId: !Ref VPC
    AvailabilityZone: !Select [ 1, !GetAZs  '' ]
    CidrBlock: !Ref PublicSubnet2CIDR
    MapPublicIpOnLaunch: true
    Tags:
      - Key: Name
        Value: !Sub ${EnvironmentName} Public Subnet (AZ2)

PrivateSubnet1:
  Type: AWS::EC2::Subnet
  Properties:
    VpcId: !Ref VPC
    AvailabilityZone: !Select [ 0, !GetAZs  '' ]
    CidrBlock: !Ref PrivateSubnet1CIDR
    MapPublicIpOnLaunch: false
    Tags:
      - Key: Name
        Value: !Sub ${EnvironmentName} Private Subnet (AZ1)
```

```
  PrivateSubnet2:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      AvailabilityZone: !Select [ 1, !GetAZs  '' ]
      CidrBlock: !Ref PrivateSubnet2CIDR
      MapPublicIpOnLaunch: false
      Tags:
        - Key: Name
          Value: !Sub ${EnvironmentName} Private Subnet (AZ2)

  NatGateway1EIP:
    Type: AWS::EC2::EIP
    DependsOn: InternetGatewayAttachment
    Properties:
      Domain: vpc

  NatGateway2EIP:
    Type: AWS::EC2::EIP
    DependsOn: InternetGatewayAttachment
    Properties:
      Domain: vpc

  NatGateway1:
    Type: AWS::EC2::NatGateway
    Properties:
      AllocationId: !GetAtt NatGateway1EIP.AllocationId
      SubnetId: !Ref PublicSubnet1

  NatGateway2:
    Type: AWS::EC2::NatGateway
    Properties:
      AllocationId: !GetAtt NatGateway2EIP.AllocationId
      SubnetId: !Ref PublicSubnet2

  PublicRouteTable:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: !Sub ${EnvironmentName} Public Routes

  DefaultPublicRoute:
```

```
    Type: AWS::EC2::Route
    DependsOn: InternetGatewayAttachment
    Properties:
      RouteTableId: !Ref PublicRouteTable
      DestinationCidrBlock: 0.0.0.0/0
      GatewayId: !Ref InternetGateway

  PublicSubnet1RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PublicRouteTable
      SubnetId: !Ref PublicSubnet1

  PublicSubnet2RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PublicRouteTable
      SubnetId: !Ref PublicSubnet2


  PrivateRouteTable1:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: !Sub ${EnvironmentName} Private Routes (AZ1)

  DefaultPrivateRoute1:
    Type: AWS::EC2::Route
    Properties:
      RouteTableId: !Ref PrivateRouteTable1
      DestinationCidrBlock: 0.0.0.0/0
      NatGatewayId: !Ref NatGateway1

  PrivateSubnet1RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PrivateRouteTable1
      SubnetId: !Ref PrivateSubnet1

  PrivateRouteTable2:
    Type: AWS::EC2::RouteTable
    Properties:
```

```
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: !Sub ${EnvironmentName} Private Routes (AZ2)

  DefaultPrivateRoute2:
    Type: AWS::EC2::Route
    Properties:
      RouteTableId: !Ref PrivateRouteTable2
      DestinationCidrBlock: 0.0.0.0/0
      NatGatewayId: !Ref NatGateway2

  PrivateSubnet2RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PrivateRouteTable2
      SubnetId: !Ref PrivateSubnet2

  SecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupName: "mwaa-security-group"
      GroupDescription: "Security group with a self-referencing inbound rule."
      VpcId: !Ref VPC

  SecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: !Ref SecurityGroup
      IpProtocol: "-1"
      SourceSecurityGroupId: !Ref SecurityGroup

  EnvironmentBucket:
    Type: AWS::S3::Bucket
    Properties:
      VersioningConfiguration:
        Status: Enabled
      PublicAccessBlockConfiguration:
        BlockPublicAcls: true
        BlockPublicPolicy: true
        IgnorePublicAcls: true
        RestrictPublicBuckets: true
```

```
##################################################################################################
 # CREATE MWAA

##################################################################################################

 MwaaEnvironment:
   Type: AWS::MWAA::Environment
   DependsOn: MwaaExecutionPolicy
   Properties:
     Name: !Sub "${AWS::StackName}-MwaaEnvironment"
     SourceBucketArn: !GetAtt EnvironmentBucket.Arn
     ExecutionRoleArn: !GetAtt MwaaExecutionRole.Arn
     DagS3Path: dags
     NetworkConfiguration:
       SecurityGroupIds:
         - !GetAtt SecurityGroup.GroupId
       SubnetIds:
         - !Ref PrivateSubnet1
         - !Ref PrivateSubnet2
     WebserverAccessMode: PUBLIC_ONLY
     MaxWorkers: !Ref MaxWorkerNodes
     LoggingConfiguration:
       DagProcessingLogs:
         LogLevel: !Ref DagProcessingLogs
         Enabled: true
       SchedulerLogs:
         LogLevel: !Ref SchedulerLogsLevel
         Enabled: true
       TaskLogs:
         LogLevel: !Ref TaskLogsLevel
         Enabled: true
       WorkerLogs:
         LogLevel: !Ref WorkerLogsLevel
         Enabled: true
       WebserverLogs:
         LogLevel: !Ref WebserverLogsLevel
         Enabled: true
 SecurityGroup:
   Type: AWS::EC2::SecurityGroup
   Properties:
     VpcId: !Ref VPC
     GroupDescription: !Sub "Security Group for Amazon MWAA Environment
${AWS::StackName}-MwaaEnvironment"
```

```
      GroupName: !Sub "airflow-security-group-${AWS::StackName}-MwaaEnvironment"

  SecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: !Ref SecurityGroup
      IpProtocol: "-1"
      SourceSecurityGroupId: !Ref SecurityGroup

  SecurityGroupEgress:
    Type: AWS::EC2::SecurityGroupEgress
    Properties:
      GroupId: !Ref SecurityGroup
      IpProtocol: "-1"
      CidrIp: "0.0.0.0/0"

  MwaaExecutionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          - Effect: Allow
            Principal:
              Service:
                - airflow-env.amazonaws.com
                - airflow.amazonaws.com
            Action:
             - "sts:AssumeRole"
      Path: "/service-role/"

  MwaaExecutionPolicy:
    DependsOn: EnvironmentBucket
    Type: AWS::IAM::ManagedPolicy
    Properties:
      Roles:
        - !Ref MwaaExecutionRole
      PolicyDocument:
        Version: 2012-10-17
        Statement:
          - Effect: Allow
            Action: airflow:PublishMetrics
            Resource:
```

```
              - !Sub "arn:aws:airflow:${AWS::Region}:${AWS::AccountId}:environment/
${EnvironmentName}"
          - Effect: Deny
            Action: s3:ListAllMyBuckets
            Resource:
              - !Sub "${EnvironmentBucket.Arn}"
              - !Sub "${EnvironmentBucket.Arn}/*"

          - Effect: Allow
            Action:
              - "s3:GetObject*"
              - "s3:GetBucket*"
              - "s3:List*"
            Resource:
              - !Sub "${EnvironmentBucket.Arn}"
              - !Sub "${EnvironmentBucket.Arn}/*"
          - Effect: Allow
            Action:
              - logs:DescribeLogGroups
            Resource: "*"

          - Effect: Allow
            Action:
              - logs:CreateLogStream
              - logs:CreateLogGroup
              - logs:PutLogEvents
              - logs:GetLogEvents
              - logs:GetLogRecord
              - logs:GetLogGroupFields
              - logs:GetQueryResults
              - logs:DescribeLogGroups
            Resource:
              - !Sub "arn:aws:logs:${AWS::Region}:${AWS::AccountId}:log-
group:airflow-${AWS::StackName}*"
          - Effect: Allow
            Action: cloudwatch:PutMetricData
            Resource: "*"
          - Effect: Allow
            Action:
              - sqs:ChangeMessageVisibility
              - sqs:DeleteMessage
              - sqs:GetQueueAttributes
              - sqs:GetQueueUrl
              - sqs:ReceiveMessage
```

```
              - sqs:SendMessage
            Resource:
              - !Sub "arn:aws:sqs:${AWS::Region}:*:airflow-celery-*"
          - Effect: Allow
            Action:
              - kms:Decrypt
              - kms:DescribeKey
              - "kms:GenerateDataKey*"
              - kms:Encrypt
            NotResource: !Sub "arn:aws:kms:*:${AWS::AccountId}:key/*"
            Condition:
              StringLike:
                "kms:ViaService":
                  - !Sub "sqs.${AWS::Region}.amazonaws.com"
Outputs:
  VPC:
    Description: A reference to the created VPC
    Value: !Ref VPC

  PublicSubnets:
    Description: A list of the public subnets
    Value: !Join [ ",", [ !Ref PublicSubnet1, !Ref PublicSubnet2 ]]

  PrivateSubnets:
    Description: A list of the private subnets
    Value: !Join [ ",", [ !Ref PrivateSubnet1, !Ref PrivateSubnet2 ]]

  PublicSubnet1:
    Description: A reference to the public subnet in the 1st Availability Zone
    Value: !Ref PublicSubnet1

  PublicSubnet2:
    Description: A reference to the public subnet in the 2nd Availability Zone
    Value: !Ref PublicSubnet2

  PrivateSubnet1:
    Description: A reference to the private subnet in the 1st Availability Zone
    Value: !Ref PrivateSubnet1

  PrivateSubnet2:
    Description: A reference to the private subnet in the 2nd Availability Zone
    Value: !Ref PrivateSubnet2

  SecurityGroupIngress:
```

```
    Description: Security group with self-referencing inbound rule
    Value: !Ref SecurityGroupIngress

  MwaaApacheAirflowUI:
    Description: MWAA Environment
    Value: !Sub  "https://${MwaaEnvironment.WebserverUrl}"
```

ステップ 2: を使用してスタックを作成する AWS CLI

1. コマンドプロンプトで、mwaa-public-network.yml が保存されているディレクトリに移動します。例:

```
cd mwaaproject
```

2. AWS CLIを使用してスタックを作成するには、「aws cloudformation create-stack」コマンドを使用します。

```
aws cloudformation create-stack --stack-name mwaa-environment-public-network --
template-body file://mwaa-public-network.yml --capabilities CAPABILITY_IAM
```

> **Note**
>
> Amazon VPC インフラストラクチャ、Amazon S3 バケット、Amazon MWAA 環境の作成には 30 分以上かかります。

ステップ 3: DAG を Amazon S3 にアップロードし、Apache Airflow UI で実行する

1. 「サポートされている最新の Apache Airflow バージョン」の tutorial.py ファイルの内容をコピーし、tutorial.py という名前でローカルに保存します。
2. コマンドプロンプトで、tutorial.py が保存されているディレクトリに移動します。例:

```
cd mwaaproject
```

3. 以下のコマンドを使って、Amazon S3 バケットをすべてリストアップします

```
aws s3 ls
```

4. 以下のコマンドを使用して、ご使用の環境の Amazon S3 バケット内のファイルとフォルダを一覧表示します。

```
aws s3 ls s3://YOUR_S3_BUCKET_NAME
```

5. 以下のスクリプトを使用して、tutorial.py ファイルを dags フォルダにアップロードしてください。*YOUR_S3_BUCKET_NAME* のサンプル値に置き換えてください。

```
aws s3 cp tutorial.py s3://YOUR_S3_BUCKET_NAME/dags/
```

6. Amazon MWAA コンソールで「環境ページ」を開きます。
7. 環境を選択します。
8. [Airflow UI を開く] を選択します。
9. Apache Airflow UI で、使用可能な DAG のリストから [チュートリアル] 用の DAG を選択します。
10. DAG の詳細ページで、DAG 名の横にある [DAG の一時停止/一時停止解除] トグルを選択して DAG の一時停止を解除します。
11. [DAG をトリガー] を選択します。

ステップ 4: Logs で CloudWatch ログを表示する

AWS CloudFormation スタックによって有効になったすべての Apache Airflow ログについて、CloudWatch コンソールで Apache Airflow ログを表示できます。次のセクションでは、「Airflow ウェブサーバーロググループ」のログを表示する方法を示します。

1. Amazon MWAA コンソールで「環境ページ」を開きます。
2. 環境を選択します。
3. [モニタリング] ペインで [Airflow ウェブサーバーのロググループ] を選択します。
4. [ログストリーム] の webserver_console_ip ログを選択します。

次のステップ

- DAG をアップロードし、`requirements.txt` で Python の依存関係を指定し、`plugins.zip` でカスタムプラグインを指定する方法については、Amazon MWAA での DAG の取り扱い で詳細情報を確認します。
- ご使用の環境のパフォーマンスを調整するために推奨するベスト・プラクティスの詳細については、「Amazon MWAA での Apache Airflow のパフォーマンス調整」 を参照してください。
- Amazon MWAA のモニタリングダッシュボードとアラーム で環境のモニタリングダッシュボードをで作成します。
- Amazon Managed Workflows for Apache Airflow にある DAG コードサンプルをいくつか実行します。

Amazon Managed Workflows for Apache Airflowを使い始める

Apache Airflow 用 Amazon マネージドワークフローは、Amazon S3 ストレージバケット内の Amazon VPC、DAG コード、およびサポートファイルを使用して環境を作成します。このガイドでは、Amazon MWAAの利用を開始するための前提条件と AWS リソースについて説明します。

トピック

前提条件

Amazon MWAA環境を作成するには、必要な AWS リソースを作成する権限があることを確認するために、追加の手順が必要な場合があります。

- [AWS アカウント] — AmazonのMWAAおよび AWS サービスと、お客様の環境で使用されているリソースを使用することができる AWS アカウントです。

本ガイドについて

このセクションでは、このガイドで作成する AWS インフラストラクチャとリソースについて説明します。

- [Amazon VPC] — Amazon MWAA環境に必要なAmazon VPCネットワークコンポーネントです。「Amazon MWAA でのネットワーキングについて」に示すように、これらの要件（上級）を満たす既存の VPC を設定することも、「the section called “VPCネットワークを作成する”」 で定義されているように VPC とネットワークコンポーネントを作成することもできます。

- [Amazon S3 バケット] — DAG および関連ファイル (`plugins.zip` や `requirements.txt` など) を保存するための Amazon S3 バケットです。Amazon S3 バケットは、「Amazon MWAA 用の Amazon S3 バケットの作成」で定義されているように、[バケットバージョニング] を有効にして、[すべてのパブリックアクセスをブロック] するように設定する必要があります。
- [Amazon MWAA 環境] — Amazon S3バケットの場所、DAGコードと任意のカスタムプラグインまたは Python 依存関係へのパス、および Amazon VPC とそのセキュリティグループの場所が「Amazon MWAA 環境を作成する」で定義されているように構成されたAmazon MWAA環境。

開始する前に

Amazon MWAA 環境を作成するには、環境を作成する前に、追加の AWS リソースを作成および設定する追加の手順が必要になる場合があります。

環境を作成するには、以下が必要です。

- [AWS KMS キー] — 環境内のデータを暗号化するための AWS KMS キー。Amazon MWAAコンソールでデフォルトオプションを選択して、環境の作成時に「 AWS が所有するキー」 を作成したり、環境設定(詳細)で使用する他の AWS サービスに対する権限を持つ既存の「カスタマーマネージドキー」 を指定することができます。詳細については、「暗号化のためのカスタマーマネージドキーの使用」を参照してください。
- [実行ロール] — Amazon MWAA が環境内にある AWS のリソースにアクセスできるようにする実行ロール。環境を作成するとき、Amazon MWAAコンソールでデフォルトのオプションを選択して実行ロールを作成できます。 詳細については、「Amazon MWAA 実行ロール」を参照してください。
- [VPC セキュリティグループ] — Amazon MWAA が VPC ネットワーク内の他の AWS リソースにアクセスできるようにする VPC セキュリティグループ。Amazon MWAA コンソールのデフォルトオプションを選択して、環境の作成時にセキュリティグループを作成したり、セキュリティグループに適切なインバウンドルールとアウトバウンドルールを設定したりできます (上級者向け)。詳細については、「Amazon MWAA の VPC のセキュリティ」を参照してください。

利用できるリージョン

Amazon MWAA は、次の AWS リージョンで利用できます。

- 欧州 (ストックホルム) eu-north-1
- 欧州 (フランクフルト) eu-central-1

- ヨーロッパ (アイルランド) eu-west-1
- 欧州 (ロンドン) eu-west-2
- 欧州 (パリ) eu-west-3
- アジアパシフィック (ムンバイ) ap-south-1
- アジアパシフィック (シンガポール) ap-southeast-1
- アジアパシフィック (シドニー) - ap-southeast-2
- アジアパシフィック (東京) - ap-northeast-1
- アジアパシフィック (ソウル) ap-northeast-2
- 米国東部 (バージニア北部) - us-east-1
- 米国東部 (オハイオ) - us-east-2
- 米国東部 (オレゴン) - us-west-2
- カナダ (中部) ca-central-1
- 南米 (サンパウロ) sa-east-1

Amazon MWAA 用の Amazon S3 バケットの作成

このガイドでは、Amazon S3 バケットを作成して、Apache Airflow 指向非循環グラフ (DAG)、カスタムプラグイン、`plugins.zip`ファイル内の Python 依存関係を`requirements.txt`ファイルに保存する手順について説明します。

目次

開始する前に

- Amazon S3 バケットを作成した後は、名前を変更することはできません。詳細については、「Amazon Simple Storage Service ユーザーガイド」の「バケットの命名規則」を参照してください。
- Amazon MWAA 環境に使用する Amazon S3 バケットは、バケットバージョニングを有効にして、すべてのパブリックアクセスをブロックするように設定する必要があります。

- Amazon MWAA 環境に使用される Amazon S3 バケットは、Amazon MWAA AWS 環境と同じリージョンにある必要があります。Amazon MWAA AWSのリージョンのリストが表示するには、「AWS 全般のリファレンス」の「Amazon MWAA エンドポイントとクォータ」を参照してください。

バケットを作成する

このセクションでは、環境に合わせて Amazon S3 バケットを作成する手順について説明します。

バケットを作成するには

1. AWS Management Console にサインインし、Amazon S3 コンソール (https://console.aws.amazon.com/s3/) を開きます。
2. [バケットの作成] を選択します。
3. [バケット名] に、バケットの DNS に準拠する名前を入力します。

   バケット名には次の条件があります。

   - すべての Amazon S3 で一意にする。
   - 3～63 文字で指定する。
   - 大文字を含めないでください。
   - 先頭の文字には小文字の英文字または数字を使用する。

   > **Important**
   >
   > バケット名にアカウント番号などの機密情報を含めないでください。バケット名は、バケット内のオブジェクトを参照する URL に表示されます。

4. [リージョン] AWS でリージョンを選択します。Amazon MWAA 環境のAWSと同じリージョンである必要があります。

   - レイテンシーとコストを最小化するため、さらに規制条件に対応するために、最寄りのリージョンを選択します。

5. [Block all public access (すべてのパブリックアクセスをブロック)] を選択します。
6. [バケットバージョニング] で [有効化] を選択します。

7. タグ – オプション。キーと値のタグペアを追加して、Amazon S3 バケットを [タグ] で識別します。例えば、 `Bucket : Staging` です。
8. オプション-サーバー側の暗号化。オプションで、Amazon S3 バケットで次の暗号化オプションのいずれかを有効にすることができます。

    a. バケットに対してサーバー側の暗号化を有効にするには、[サーバー側の暗号化] で [Amazon S3 key (SSE-S3)] を選択します。

    b. Amazon S3 バケットの暗号化にAWS KMSキーを使用するには、AWS Key Management Serviceキー (SSE-KMS) を選択してください。

        i. AWSマネージドキー (aws/s3)-このオプションを選択した場合、Amazon MWAA AWS が管理する所有キーを使用するか、Amazon MWAA 環境の暗号化にカスタマー管理キーを指定できます。

        ii. AWS KMSキーから選択するか、AWS KMSキー ARN を入力-このステップでカスタマー管理キーを指定する場合は、AWS KMSキー ID または ARN を指定する必要があります。 Amazon MWAA ではAWS KMSエイリアスとマルチリージョンキーはサポートされていません。指定する AWS KMS キーは Amazon MWAA 環境での暗号化にも使用する必要があります。

9. オプション - アドバンスト設定 Amazon S3 Object Lock を有効にする場合:

    a. [アドバンスト設定]、[有効にする] を選択します。

    > **Important**
    >
    > Object Lock を有効にすると、このバケット内のオブジェクトは永久にロックされます。Amazon S3 Object Lockの詳細については、Amazon Simple Storage Service ユーザーガイドの「Amazon S3 Object Lockの使用」を参照してください。

    b. 承認を選択します。

10. [バケットの作成] を選択します。

次のステップ

- VPCネットワークを作成する の環境に必要なAmazon VPC ネットワークを作成する方法を学んでください。
- アクセス権限の管理方法については、「ACL バケット権限の設定方法」を参照してください。

- S3 バケットでストレージバケットを削除する方法 。

VPCネットワークを作成する

Apache Airflow の Amazon マネージドワークフローには、環境をサポートするために Amazon VPC と特定のネットワークコンポーネントが必要です。このガイドでは、Apache Airflow 環境の Amazon マネージドワークフロー用に Amazon VPCネットワークを作成するためのさまざまなオプションについて説明します。

> **Note**
>
> Apache Airflow は、ローレイテンシーのネットワーク環境で最もよく機能します。既存の Amazon VPCを使用しており、トラフィックが他のリージョンやオンプレミス環境にルーティングされている場合、Amazon SQS、CloudWatch、Amazon S3、AWS KMS、および Amazon ECRの AWS PrivateLink エンドポイントを追加することをお勧めします。Amazon MWAA用の AWS PrivateLink の設定の詳細については、「インターネットアクセスのない Amazon VPCネットワークの作成」 を参照してください。

目次

前提条件

AWS Command Line Interface (AWS CLI) は、コマンドラインシェルでコマンドを使用して AWS サービスとやり取りするためのオープンソースツールです。このページのステップを完了するには、以下のものが必要です。

- 「AWS CLI - バージョン 2 のインストール」

- 「AWS CLI - aws configure によるクイック設定」

開始する前に

- 環境に指定する「VPC ネットワーク」 の内容は、環境作成後に変更することはできません。
- Amazon VPC と Apache Airflow ウェブサーバーには、プライベートルーティングまたはパブリックルーティングを使用できます。オプションのリストを表示するには、「the section called "Amazon VPC と Apache エアフローアクセスモードのユースケース例"」 を参照してください。

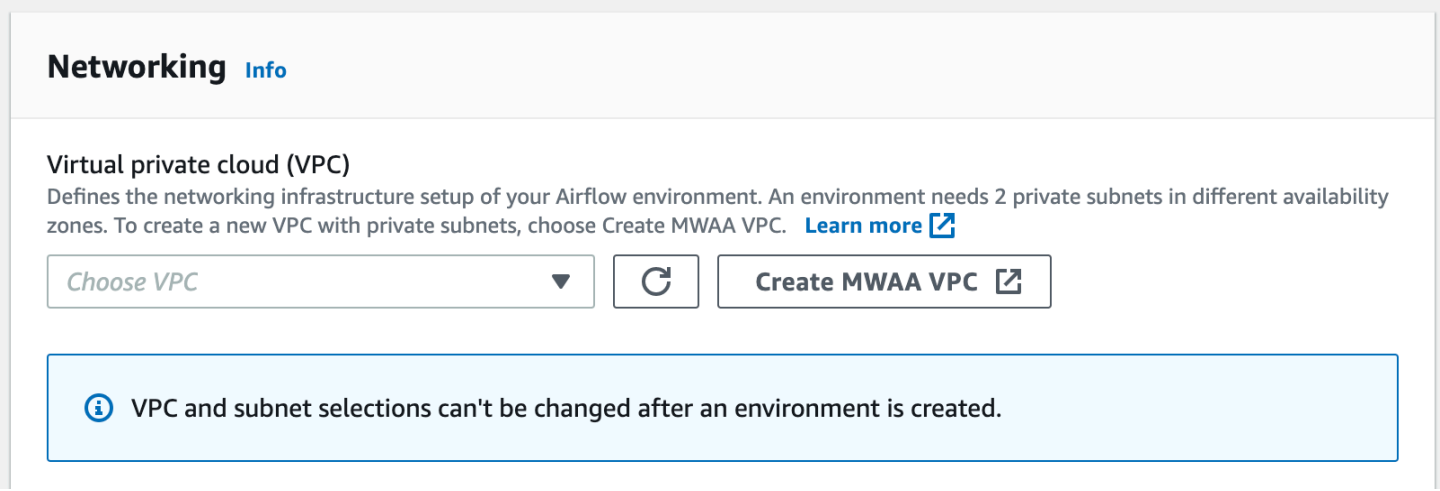
Amazon VPC ネットワークを作成するためのオプション

次のセクションでは、環境用の Amazon VPC ネットワークを作成するために使用するオプションについて説明します。

オプション 1: Amazon MWAA コンソールで VPC ネットワークを作成する

次のセクションでは、Amazon MWAA コンソールで Amazon VPC ネットワークの作成方法を説明します。このオプションは「インターネット経由のパブリックルーティング」 を使用します。これは、[プライベートネットワーク] または [パブリックネットワークアクセスモード] を持つ Apache Airflow Web サーバーに使用できます。

次の図は、Amazon MWAA コンソールの [MWAA VPC の作成] ボタンの場所を示しています。



オプション 2: インターネットにアクセス可能な Amazon VPC ネットワークの作成

以下の AWS CloudFormation テンプレートは、デフォルトの AWS リージョンでインターネットアクセスを備えたAmazon VPCネットワークを作成します。このオプションは「インターネット経由の

パブリックルーティング」を使用します。このテンプレートは、[プライベートネットワーク] または [パブリックネットワークアクセスモード] を持つ Apache Airflow Web サーバーに使用できます。

1. 以下テンプレートの内容をコピーし、cfn-vpc-public-private.yaml としてローカルに保存します。「テンプレートをダウンロードする」こともできます。

```
Description:  This template deploys a VPC, with a pair of public and private
 subnets spread
  across two Availability Zones. It deploys an internet gateway, with a default
  route on the public subnets. It deploys a pair of NAT gateways (one in each AZ),
  and default routes for them in the private subnets.

Parameters:
  EnvironmentName:
    Description: An environment name that is prefixed to resource names
    Type: String
    Default: mwaa-

  VpcCIDR:
    Description: Please enter the IP range (CIDR notation) for this VPC
    Type: String
    Default: 10.192.0.0/16

  PublicSubnet1CIDR:
    Description: Please enter the IP range (CIDR notation) for the public subnet in
 the first Availability Zone
    Type: String
    Default: 10.192.10.0/24

  PublicSubnet2CIDR:
    Description: Please enter the IP range (CIDR notation) for the public subnet in
 the second Availability Zone
    Type: String
    Default: 10.192.11.0/24

  PrivateSubnet1CIDR:
    Description: Please enter the IP range (CIDR notation) for the private subnet
 in the first Availability Zone
    Type: String
    Default: 10.192.20.0/24

  PrivateSubnet2CIDR:
```

```
    Description: Please enter the IP range (CIDR notation) for the private subnet
 in the second Availability Zone
    Type: String
    Default: 10.192.21.0/24

Resources:
  VPC:
    Type: AWS::EC2::VPC
    Properties:
      CidrBlock: !Ref VpcCIDR
      EnableDnsSupport: true
      EnableDnsHostnames: true
      Tags:
        - Key: Name
          Value: !Ref EnvironmentName

  InternetGateway:
    Type: AWS::EC2::InternetGateway
    Properties:
      Tags:
        - Key: Name
          Value: !Ref EnvironmentName

  InternetGatewayAttachment:
    Type: AWS::EC2::VPCGatewayAttachment
    Properties:
      InternetGatewayId: !Ref InternetGateway
      VpcId: !Ref VPC

  PublicSubnet1:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      AvailabilityZone: !Select [ 0, !GetAZs '' ]
      CidrBlock: !Ref PublicSubnet1CIDR
      MapPublicIpOnLaunch: true
      Tags:
        - Key: Name
          Value: !Sub ${EnvironmentName} Public Subnet (AZ1)

  PublicSubnet2:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
```

```
      AvailabilityZone: !Select [ 1, !GetAZs  '' ]
      CidrBlock: !Ref PublicSubnet2CIDR
      MapPublicIpOnLaunch: true
      Tags:
        - Key: Name
          Value: !Sub ${EnvironmentName} Public Subnet (AZ2)

  PrivateSubnet1:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      AvailabilityZone: !Select [ 0, !GetAZs  '' ]
      CidrBlock: !Ref PrivateSubnet1CIDR
      MapPublicIpOnLaunch: false
      Tags:
        - Key: Name
          Value: !Sub ${EnvironmentName} Private Subnet (AZ1)

  PrivateSubnet2:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      AvailabilityZone: !Select [ 1, !GetAZs  '' ]
      CidrBlock: !Ref PrivateSubnet2CIDR
      MapPublicIpOnLaunch: false
      Tags:
        - Key: Name
          Value: !Sub ${EnvironmentName} Private Subnet (AZ2)

  NatGateway1EIP:
    Type: AWS::EC2::EIP
    DependsOn: InternetGatewayAttachment
    Properties:
      Domain: vpc

  NatGateway2EIP:
    Type: AWS::EC2::EIP
    DependsOn: InternetGatewayAttachment
    Properties:
      Domain: vpc

  NatGateway1:
    Type: AWS::EC2::NatGateway
    Properties:
```

```
      AllocationId: !GetAtt NatGateway1EIP.AllocationId
      SubnetId: !Ref PublicSubnet1

  NatGateway2:
    Type: AWS::EC2::NatGateway
    Properties:
      AllocationId: !GetAtt NatGateway2EIP.AllocationId
      SubnetId: !Ref PublicSubnet2

  PublicRouteTable:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: !Sub ${EnvironmentName} Public Routes

  DefaultPublicRoute:
    Type: AWS::EC2::Route
    DependsOn: InternetGatewayAttachment
    Properties:
      RouteTableId: !Ref PublicRouteTable
      DestinationCidrBlock: 0.0.0.0/0
      GatewayId: !Ref InternetGateway

  PublicSubnet1RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PublicRouteTable
      SubnetId: !Ref PublicSubnet1

  PublicSubnet2RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PublicRouteTable
      SubnetId: !Ref PublicSubnet2


  PrivateRouteTable1:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: Name
```

```
          Value: !Sub ${EnvironmentName} Private Routes (AZ1)

  DefaultPrivateRoute1:
    Type: AWS::EC2::Route
    Properties:
      RouteTableId: !Ref PrivateRouteTable1
      DestinationCidrBlock: 0.0.0.0/0
      NatGatewayId: !Ref NatGateway1

  PrivateSubnet1RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PrivateRouteTable1
      SubnetId: !Ref PrivateSubnet1

  PrivateRouteTable2:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: !Sub ${EnvironmentName} Private Routes (AZ2)

  DefaultPrivateRoute2:
    Type: AWS::EC2::Route
    Properties:
      RouteTableId: !Ref PrivateRouteTable2
      DestinationCidrBlock: 0.0.0.0/0
      NatGatewayId: !Ref NatGateway2

  PrivateSubnet2RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PrivateRouteTable2
      SubnetId: !Ref PrivateSubnet2

  SecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupName: "mwaa-security-group"
      GroupDescription: "Security group with a self-referencing inbound rule."
      VpcId: !Ref VPC

  SecurityGroupIngress:
```

```
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: !Ref SecurityGroup
      IpProtocol: "-1"
      SourceSecurityGroupId: !Ref SecurityGroup

Outputs:
  VPC:
    Description: A reference to the created VPC
    Value: !Ref VPC

  PublicSubnets:
    Description: A list of the public subnets
    Value: !Join [ ",", [ !Ref PublicSubnet1, !Ref PublicSubnet2 ]]

  PrivateSubnets:
    Description: A list of the private subnets
    Value: !Join [ ",", [ !Ref PrivateSubnet1, !Ref PrivateSubnet2 ]]

  PublicSubnet1:
    Description: A reference to the public subnet in the 1st Availability Zone
    Value: !Ref PublicSubnet1

  PublicSubnet2:
    Description: A reference to the public subnet in the 2nd Availability Zone
    Value: !Ref PublicSubnet2

  PrivateSubnet1:
    Description: A reference to the private subnet in the 1st Availability Zone
    Value: !Ref PrivateSubnet1

  PrivateSubnet2:
    Description: A reference to the private subnet in the 2nd Availability Zone
    Value: !Ref PrivateSubnet2

  SecurityGroupIngress:
    Description: Security group with self-referencing inbound rule
    Value: !Ref SecurityGroupIngress
```

2. コマンドプロンプトで、cfn-vpc-public-private.yaml が保存されているディレクトリに移動します。例:

```
cd mwaaproject
```

3. AWS CLI を使用してスタックを作成するには、aws cloudformation create-stack コマンドを使用します。

```
aws cloudformation create-stack --stack-name mwaa-environment --template-body
 file://cfn-vpc-public-private.yaml
```

> **Note**
>
> Amazon VPC インフラストラクチャの作成には 30 分ほどかかります。

オプション 3: インターネットにアクセスせずに Amazon VPC ネットワークを作成する

以下の AWS CloudFormation テンプレートは、デフォルトのゾーンにインターネットアクセスなしでAmazon VPCネットワーク AWS を作成します。

> **Important**
>
> インターネットにアクセスしないで Amazon VPC を使用する場合、ゲートウェイエンドポイントを使用して Amazon S3 にアクセスする権限を Amazon ECR に付与する必要があります。ゲートウェイエンドポイントを作成するには、以下を実行します。
>
> 1. 次の JSON IAM ポリシーをコピーし、s3-gw-endpoint-policy.json としてローカルに保存します。 k のポリシーは、Amazon ECR が Amazon S3 リソースのアクセスに必要な最低限の権限を付与します。
>
> ```
> {
>   "Statement": [
>     {
>       "Sid": "Access-to-specific-bucket-only",
>       "Principal": "*",
>       "Action": [
>         "s3:GetObject"
>       ],
>       "Effect": "Allow",
>       "Resource": ["arn:aws:s3:::prod-region-starport-layer-bucket/*"]
>     }
>   ]
> ```

```
}
```

2. 以下の AWS CLI コマンドを使ってエンドポイントを作成します。--vpc-id および --route-table-ids の値を Amazon VPC の情報に置き換えてください。--service-name 地域に応じた名前に置き換えてください。

```
$ aws ec2 create-vpc-endpoint --vpc-id vpc-1a2b3c4d \
--service-name com.amazonaws.us-west-2.s3 \
--route-table-ids rtb-11aa22bb \
--vpc-endpoint-type Gateway \
--policy-document file://s3-gw-endpoint-policy.json
```

Amazon ECR 用の Amazon S3 ゲートウェイエンドポイントの作成についての詳細は、Amazon Elastic Container Registry ユーザーガイドの「Amazon S3 ゲートウェイのエンドポイント作成」を参照してください。

このオプションは「インターネットにアクセスできないプライベートルーティング」 を使用します。このテンプレートは、[プライベートネットワーク] アクセスモードを持つApache Airflow Web サーバーでのみ使用できます。それは 「環境で使用される AWS のサービスに必要なVPCエンドポイント」 を作成します。

1. 以下テンプレートの内容をコピーし、cfn-vpc-private.yaml としてローカルに保存します。「テンプレートをダウンロードする」こともできます。

```
AWSTemplateFormatVersion: "2010-09-09"

Parameters:
   VpcCIDR:
     Description: The IP range (CIDR notation) for this VPC
     Type: String
     Default: 10.192.0.0/16

   PrivateSubnet1CIDR:
     Description: The IP range (CIDR notation) for the private subnet in the first
 Availability Zone
     Type: String
     Default: 10.192.10.0/24

   PrivateSubnet2CIDR:
```

```
    Description: The IP range (CIDR notation) for the private subnet in the second
 Availability Zone
    Type: String
    Default: 10.192.11.0/24

Resources:
  VPC:
    Type: AWS::EC2::VPC
    Properties:
      CidrBlock: !Ref VpcCIDR
      EnableDnsSupport: true
      EnableDnsHostnames: true
      Tags:
       - Key: Name
         Value: !Ref AWS::StackName

  RouteTable:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref VPC
      Tags:
       - Key: Name
         Value: !Sub "${AWS::StackName}-route-table"

  PrivateSubnet1:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      AvailabilityZone: !Select [ 0, !GetAZs  '' ]
      CidrBlock: !Ref PrivateSubnet1CIDR
      MapPublicIpOnLaunch: false
      Tags:
       - Key: Name
         Value: !Sub "${AWS::StackName} Private Subnet (AZ1)"

  PrivateSubnet2:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      AvailabilityZone: !Select [ 1, !GetAZs  '' ]
      CidrBlock: !Ref PrivateSubnet2CIDR
      MapPublicIpOnLaunch: false
      Tags:
       - Key: Name
```

```
          Value: !Sub "${AWS::StackName} Private Subnet (AZ2)"

  PrivateSubnet1RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref RouteTable
      SubnetId: !Ref PrivateSubnet1

  PrivateSubnet2RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref RouteTable
      SubnetId: !Ref PrivateSubnet2

  S3VpcEndoint:
    Type: AWS::EC2::VPCEndpoint
    Properties:
      ServiceName: !Sub "com.amazonaws.${AWS::Region}.s3"
      VpcEndpointType: Gateway
      VpcId: !Ref VPC
      RouteTableIds:
       - !Ref RouteTable

  SecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      VpcId: !Ref VPC
      GroupDescription: Security Group for Amazon MWAA Environments to access VPC
endpoints
      GroupName: !Sub "${AWS::StackName}-mwaa-vpc-endpoints"

  SecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: !Ref SecurityGroup
      IpProtocol: "-1"
      SourceSecurityGroupId: !Ref SecurityGroup

  SqsVpcEndoint:
    Type: AWS::EC2::VPCEndpoint
    Properties:
      ServiceName: !Sub "com.amazonaws.${AWS::Region}.sqs"
      VpcEndpointType: Interface
      VpcId: !Ref VPC
```

```
      PrivateDnsEnabled: true
      SubnetIds:
       - !Ref PrivateSubnet1
       - !Ref PrivateSubnet2
      SecurityGroupIds:
       - !Ref SecurityGroup

  CloudWatchLogsVpcEndoint:
    Type: AWS::EC2::VPCEndpoint
    Properties:
      ServiceName: !Sub "com.amazonaws.${AWS::Region}.logs"
      VpcEndpointType: Interface
      VpcId: !Ref VPC
      PrivateDnsEnabled: true
      SubnetIds:
       - !Ref PrivateSubnet1
       - !Ref PrivateSubnet2
      SecurityGroupIds:
       - !Ref SecurityGroup

  CloudWatchMonitoringVpcEndoint:
    Type: AWS::EC2::VPCEndpoint
    Properties:
      ServiceName: !Sub "com.amazonaws.${AWS::Region}.monitoring"
      VpcEndpointType: Interface
      VpcId: !Ref VPC
      PrivateDnsEnabled: true
      SubnetIds:
       - !Ref PrivateSubnet1
       - !Ref PrivateSubnet2
      SecurityGroupIds:
       - !Ref SecurityGroup

  KmsVpcEndoint:
    Type: AWS::EC2::VPCEndpoint
    Properties:
      ServiceName: !Sub "com.amazonaws.${AWS::Region}.kms"
      VpcEndpointType: Interface
      VpcId: !Ref VPC
      PrivateDnsEnabled: true
      SubnetIds:
       - !Ref PrivateSubnet1
       - !Ref PrivateSubnet2
      SecurityGroupIds:
```

```
      - !Ref SecurityGroup

  EcrApiVpcEndoint:
    Type: AWS::EC2::VPCEndpoint
    Properties:
      ServiceName: !Sub "com.amazonaws.${AWS::Region}.ecr.api"
      VpcEndpointType: Interface
      VpcId: !Ref VPC
      PrivateDnsEnabled: true
      SubnetIds:
       - !Ref PrivateSubnet1
       - !Ref PrivateSubnet2
      SecurityGroupIds:
       - !Ref SecurityGroup

  EcrDkrVpcEndoint:
    Type: AWS::EC2::VPCEndpoint
    Properties:
      ServiceName: !Sub "com.amazonaws.${AWS::Region}.ecr.dkr"
      VpcEndpointType: Interface
      VpcId: !Ref VPC
      PrivateDnsEnabled: true
      SubnetIds:
       - !Ref PrivateSubnet1
       - !Ref PrivateSubnet2
      SecurityGroupIds:
       - !Ref SecurityGroup

  AirflowApiVpcEndoint:
    Type: AWS::EC2::VPCEndpoint
    Properties:
      ServiceName: !Sub "com.amazonaws.${AWS::Region}.airflow.api"
      VpcEndpointType: Interface
      VpcId: !Ref VPC
      PrivateDnsEnabled: true
      SubnetIds:
       - !Ref PrivateSubnet1
       - !Ref PrivateSubnet2
      SecurityGroupIds:
       - !Ref SecurityGroup

  AirflowEnvVpcEndoint:
    Type: AWS::EC2::VPCEndpoint
    Properties:
```

```
        ServiceName: !Sub "com.amazonaws.${AWS::Region}.airflow.env"
        VpcEndpointType: Interface
        VpcId: !Ref VPC
        PrivateDnsEnabled: true
        SubnetIds:
         - !Ref PrivateSubnet1
         - !Ref PrivateSubnet2
        SecurityGroupIds:
         - !Ref SecurityGroup

Outputs:
   VPC:
     Description: A reference to the created VPC
     Value: !Ref VPC

   MwaaSecurityGroupId:
     Description: Associates the Security Group to the environment to allow access
 to the VPC endpoints
     Value: !Ref SecurityGroup

   PrivateSubnets:
     Description: A list of the private subnets
     Value: !Join [ ",", [ !Ref PrivateSubnet1, !Ref PrivateSubnet2 ]]

   PrivateSubnet1:
     Description: A reference to the private subnet in the 1st Availability Zone
     Value: !Ref PrivateSubnet1

   PrivateSubnet2:
     Description: A reference to the private subnet in the 2nd Availability Zone
     Value: !Ref PrivateSubnet2
```

2. コマンドプロンプトで、cfn-vpc-private.yml が保存されているディレクトリに移動します。例:

```
cd mwaaproject
```

3. AWS CLI を使用してスタックを作成するには、aws cloudformation create-stack コマンドを使用します。

```
aws cloudformation create-stack --stack-name mwaa-private-environment --template-
body file://cfn-vpc-private.yml
```

> **Note**
>
> Amazon VPC インフラストラクチャの作成には 30 分ほどかかります。

4. お客様のコンピュータから、これらの VPC エンドポイントにアクセスするためのメカニズムを作成する必要があります。詳細については、「Amazon MWAA のサービス固有の Amazon VPC エンドポイントへのアクセスの管理」を参照してください。

> **Note**
>
> Amazon MWAA セキュリティグループの CIDR でアウトバウンドアクセスをさらに制限することができます。たとえば、自己参照型のアウトバウンドルール、Amazon S3 の「プレフィックスリスト」、Amazon VPC の CIDR を追加することで、それ自体に制限することができます。

次のステップ

- Amazon MWAA 環境を作成する でAmazon MWAA環境を作成する方法を学びます。
- チュートリアル: AWS Client VPN を使用したプライベートネットワークアクセスの設定 で専用ルートを使用して、コンピュータから Amazon VPC への VPN トンネルを作成する方法を学習します。

Amazon MWAA 環境を作成する

Amazon Managed Workflows for Apache Airflow は、Apache から利用可能なのと同じオープンソースの Apache Airflow とユーザーインターフェイスを使用して、選択したバージョンの環境で Apache Airflow をセットアップします。このガイドでは、Amazon MWAA 環境を作成する手順について説明します。

目次

- ステップ2：詳細設定の設定
- ステップ 3: レビューと作成

開始する前に

- 環境用に指定した VPC ネットワークは、環境の作成後に変更することはできません。
- [すべてのパブリックアクセスをブロック] し、[バケットバージョニング] 管理を有効にするには、Amazon S3 バケットを設定する必要があります。
- Amazon MWAA を使用するためのアクセス許可と、IAM ロールを作成するための AWS Identity and Access Management (IAM) のアクセス許可を持つ AWS アカウントが必要です。Amazon VPC 内の Apache Airflow アクセスを制限する Apache Airflow ウェブサーバー のプライベートネットワークアクセスモードを選択した場合、Amazon VPC エンドポイントを作成するには IAM のアクセス許可が必要です。

Apache Airflow のバージョン

以下の Apache Airflow バージョンは、Amazon Managed Workflows for Apache Airflow でサポートされています。

> **Note**
>
> - Apache Airflow v2.2.2 以降、Amazon MWAA は Python 要件、プロバイダーパッケージ、カスタムプラグインを Apache Airflow ウェブサーバーに直接インストールすることをサポートしています。
> - Apache Airflow v2.7.2 から、要件ファイルには `--constraint` ステートメントを含める必要があります。制約を指定しない場合、要件に記載されているパッケージが使用している Apache Airflow のバージョンと互換性があることを確認するため、Amazon MWAA はお客様に代わって制約を指定します。
>
>   要件ファイルに制約を設定する方法の詳細については、「Python 依存関係のインストール」を参照してください。

Apache Airflow のバージョン	Apache Airflow ガイド	Apache Airflow の制約	Python バージョン
v2.8.1	Apache Airflow v2.8.1 リファレンスガイド	Apache Airflow v2.8.1 制約ファイル	「Python 3.11」
「v2.7.2」	「Apache Airflow v2.7.2 リファレンスガイド」	「Apache Airflow v2.7.2 制約ファイル」	「Python 3.11」
「v2.6.3」	「Apache Airflow v2.6.3 リファレンスガイド」	「Apache Airflow v2.6.3 制約ファイル」	「Python 3.10」
「v2.5.1」	「Apache Airflow v2.5.1 リファレンスガイド」	「Apache Airflow v2.5.1 制約ファイル」	「Python 3.10」
「v2.4.3」	「Apache Airflow v2.4.3 リファレンスガイド」	「Apache Airflow v2.4.3 制約ファイル」	「Python 3.10」
「v2.2.2」	「Apache Airflow v2.2.2 リファレンスガイド」	「Apache Airflow v2.2.2 制約ファイル」	「Python 3.7」
「v2.0.2」	「Apache Airflow v2.0.2 リファレンスガイド」	「Apache Airflow v2.0.2 制約ファイル」	「Python 3.7」

自己管理型の Apache Airflow デプロイの移行、または既存の Amazon MWAA 環境の移行について、メタデータデータベースのバックアップ手順を含む詳細情報は、「Amazon MWAA 移行ガイド」 を参照してください。

環境の作成

以下のセクションでは、Amazon MWAA 環境の作成手順について説明します。

ステップ1：詳細の指定

環境の詳細を指定します

1. 「Amazon MWAA コンソール」を開きます。
2. AWS リージョンセレクタを使用して、リージョンを選択します。
3. [Create environment (環境の作成)] を選択します。
4. [詳細を指定] ページの [環境の詳細] で:
   a. [Name] に、環境の一意の名前を入力します。
   b. [エアフローバージョン] で Apache エアフローバージョンを選択します。

   > **Note**
   > 値が指定されない場合、デフォルトは最新の Airflow バージョンとなります。利用可能な最新バージョンは Apache Airflow v2.8.1 です。

5. [Amazon S3 の DAG コード] で以下を指定します。
   a. 「S3 バケット」。[Browse S3]を選択して Amazon S3 バケットを選択するか、Amazon S3 URIを入力します。
   b. 「DAG フォルダー」。[Browse S3] を選択し、Amazon S3 バケット内の `dags` フォルダを選択するか、Amazon S3 URI を入力します。
   c. [プラグインファイル - オプション]。「Browse S3」を選択し、Amazon S3 バケット上の `plugins.zip` ファイルを選択するか、Amazon S3 の URI を入力します。
   d. [要件ファイル - オプション]。「Browse S3」を選択し、Amazon S3 バケット上の `requirements.txt` ファイルを選択するか、Amazon S3 の URI を入力します。
   e. [スタートアップ スクリプトファイル - オプション]、[Browse S3] を選択し、Amazon S3 バケット上のスクリプトファイルを選択するか、Amazon S3 URI を入力します。
6. [次へ] をクリックします。

ステップ2：詳細設定の設定

詳細設定を設定するには

1. [詳細設定の設定] ページの [ネットワーク] で:

- 「Amazon VPC」 を選択してください。

  このステップでは、Amazon VPC の 2 つのサブネットを設定します。

2. [Web サーバーアクセス]で、お好みの 「Apache Airflow アクセスモード」 を選択します。

   a. [プライベートネットワーク]。これは、Apache Airflow UI へのアクセスを、「あなたの環境の IAM ポリシー」にアクセスを許可されたあなたの Amazon VPC 内のユーザーに制限します このステップには Amazon VPC エンドポイントを作成する権限が必要です。

   > **Note**
   >
   > Apache Airflow UI にアクセスできるのは企業ネットワークのみで、ウェブサーバ要件をインストールするためにパブリックリポジトリにアクセスする必要がない場合は、[プライベートネットワーク] オプションを選択します。このアクセスモードオプションを選択する場合は、Amazon VPC 内の Apache Airflow ウェブサーバーにアクセスするメカニズムを作成する必要があります。詳細については、「Apache Airflow ウェブサーバーの VPC エンドポイントへのアクセス (プライベートネットワークアクセス)」を参照してください。

   b. [パブリックネットワーク]。これにより、「環境の IAM ポリシー」 へのアクセスを許可されたユーザーは、インターネット経由で Apache Airflow UI にアクセスできます。

3. [セキュリティグループ] で、「Amazon VPC」 を保護するために使用するセキュリティグループを選択します。

   a. デフォルトでは、Amazon MWAA は Amazon VPC にセキュリティグループを作成し、[新しいセキュリティグループの作成] で特定のインバウンドルールとアウトバウンドルールを設定します。

   b. オプション。[新しいセキュリティグループの作成] のチェックボックスをオフにして、最大 5 つのセキュリティグループを選択します。

   > **Note**
   >
   > 既存の Amazon VPC セキュリティグループでは、ネットワークトラフィックを許可する特定のインバウンドルールとアウトバウンドルールを設定する必要があります。詳細については、「Amazon MWAA の VPC のセキュリティ」を参照してください。

4. [環境クラス] で、「環境クラス」 を選択します。

   ワークロードをサポートするために必要な最小サイズを選択することをお勧めします。環境クラスは、随時変更できます。

5. [最大ワーカー数] は、環境内で実行する Apache Airflow ワーカーの最大数を指定します。

   詳細については、「高パフォーマンスのユースケースの例」を参照してください。

6. 最大ウェブサーバー数と最小ウェブサーバー数を指定して、Amazon MWAA が環境内の Apache Airflow ウェブサーバーをスケーリングする方法を設定します。

   ウェブサーバーのオートスケーリングの詳細については、「」を参照してくださいthe section called “ウェブサーバーのオートスケーリングの設定”。

7. [暗号化] で、データ暗号化オプションを選択します。

   a. デフォルトでは、Amazon MWAA は AWS 所有キーを使用してデータを暗号化します。

   b. オプション。暗号化設定のカスタマイズ (アドバンスト) を選択して、別の AWS KMS キーを選択します。このステップでカスタマーマネージドキーを指定する場合は、 AWS KMS キー ID または ARN を指定する必要があります。 AWS KMS エイリアスとマルチリージョンキーは Amazon MWAA ではサポートされていません。Amazon S3 バケットでサーバー側の暗号化用に Amazon S3 キーを指定した場合、Amazon MWAA 環境にも同じキーを指定する必要があります。

   > **Note**
   >
   > Amazon MWAA コンソールでキーを選択するには、そのキーに対するアクセス権限が必要です。また、「キーポリシーの添付」 で説明されているポリシーを追加して Amazon MWAA にキーを使用する権限を付与する必要があります。

8. 推奨 モニタリング で、Airflow ログ記録設定のログカテゴリを 1 つ以上選択して、Apache Airflow ログを CloudWatch Logs に送信します。

   a. [Airflow タスクログ]。ログレベルの CloudWatch ログに送信する Apache Airflow タスクログのタイプを選択します。

   b. [Airflow ウェブサーバーのログ]。ログレベルの CloudWatch ログに送信する Apache Airflow ウェブサーバーログのタイプを選択します。

   c. [Airflow スケジューラーログ]。ログレベルの CloudWatch ログに送信する Apache Airflow スケジューラログのタイプを選択します。

d. [Airflow ワーカーログ]。ログレベルの CloudWatch ログに送信する Apache Airflow ワーカーログのタイプを選択します。

e. [Airflow DAG 処理ログ]。ログレベルの CloudWatch ログに送信する Apache Airflow DAG 処理ログのタイプを選択します。

9. オプション。[エアフロー設定オプション]で、[カスタム設定オプションの追加]を選択します。

   Apache Airflow バージョンの「Apache Airflow 構成オプション」の推奨ドロップダウンリストから選択するか、カスタム設定オプションを指定できます。例えば、`core.default_task_retries : 3` です。

10. オプション。[タグ] で [新しいタグ] の追加を選択し、タグを環境に関連付けます。例えば、`Environment : Staging` です。

11. [許可] で、実行ロールを選択する：

    a. デフォルトでは、Amazon MWAA は [新しいロールの作成] で「実行ロール」を作成します。このオプションを使用するには、IAM ロールを作成する権限が必要です。

    b. オプション。[ロール ARN を入力] を選択して、既存の実行ロールの Amazon リソースネーム (ARN) を入力します。

12. [次へ] をクリックします。

ステップ 3: レビューと作成

環境の概要の表示

- 環境の概要を確認し、[環境作成] を選択します。

> **Note**
>
> 環境の作成には約 20 ~ 30 分かかります。

次のステップ

- Amazon MWAA 用の Amazon S3 バケットの作成でAmazon S3バケットを作成する方法を学びます。

Amazon MWAA 環境へのアクセスの管理

Amazon Managed Workflows for Apache Airflow は、環境で使用される他の AWS のサービスやリソースの使用を許可する必要があります。また、 AWS Identity and Access Management (IAM) で Amazon MWAA 環境と Apache Airflow UI にアクセスするためのアクセス許可を付与する必要があります。このセクションでは、環境の AWS リソースへのアクセスを許可するために使用される実行ロールと、アクセス許可を追加する方法、および Amazon MWAA 環境と Apache Airflow UI へのアクセスに必要な AWS アカウントアクセス許可について説明します。

トピック

Amazon MWAA 環境へのアクセス

Apache Airflow 用 Amazon マネージドワークフローを使用するには、アカウントと、必要な権限を持つ IAM エンティティを使用する必要があります。このページでは、Apache Airflow 用 Amazon マネージドワークフロー環境において、Apache Airflow 開発チームと Apache Airflow ユーザーにアタッチできるアクセスポリシーについて説明します。

Amazon MWAA リソースにアクセスするには、一時的な認証情報を使用し、グループとロールを使用してフェデレーティッドアイデンティティを構成することを推奨します。ベストプラクティスとして、ポリシーを IAM ユーザーに直接アタッチすることは避け、代わりにグループまたはロールを定義して AWS リソースへの一時的なアクセスを提供します。

IAM ロールは、特定の許可があり、アカウントで作成できるもう 1 つの IAM アイデンティティです。IAM ロールは、 AWS アイデンティティが で実行できることとできないことを決定するアクセス許可ポリシーを持つアイデンティティであるという点で、IAM ユーザーと似ています AWS。ただし、ユーザーは 1 人の特定の人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。また、ロールには標準の長期認証情報 (パスワードやアクセスキーなど) も関連付けられません。代わりに、ロールを引き受けると、ロールセッション用の一時的なセキュリティ認証情報が提供されます。

フェデレーテッドアイデンティティ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アイデンティティが認証されると、そのアイデンティティはロールに関連付けられ、ロールで定義されている権限が付与されます。フェデレーションの詳細については、『IAM ユーザーガイド』の「サードパーティーアイデンティティプロバイダー向けロールの作成」を参照してください。IAM アイデンティティセンターを使用する場合、権限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。権限セットの詳細については、『AWS IAM Identity Center ユーザーガイド』の「権限セット」を参照してください。

アカウントの IAM ロールを使用して、アカウントのリソースにアクセスするための別のアクセス AWS アカウント 許可を付与できます。例については、「IAM ユーザーガイド」の「チュートリアル: IAM ロール AWS アカウント を使用した 間のアクセスの委任」を参照してください。

セクション

仕組み

Amazon MWAA 環境で使用されるリソースとサービスには、すべての AWS Identity and Access Management (IAM) エンティティがアクセスできるわけではありません。Apache Airflow ユーザーにこれらのリソースへのアクセス許可を付与するポリシーを作成する必要があります。例えば、Apache Airflow 開発チームにアクセス権を付与する必要があります。

Amazon MWAA は、ユーザーが AWS コンソールまたは環境で使用される APIs でアクションを実行するために必要なアクセス許可を持っているかどうかを検証するために、これらのポリシーを使用します。

このトピックの JSON ポリシーを使用して IAM の Apache Airflow ユーザー用のポリシーを作成し、そのポリシーを IAM のユーザー、グループ、またはロールにアタッチできます。

- AmazonMWAAFullConsole アクセス — このポリシーを使用して、Amazon MWAA コンソールで環境を設定するアクセス許可を付与します。
- AmazonMWAAFullApi アクセス — このポリシーを使用して、環境の管理に使用されるすべての Amazon MWAA APIs へのアクセスを許可します。
- AmazonMWAAReadOnly アクセス — このポリシーを使用して、Amazon MWAA コンソールで環境が使用するリソースを表示するためのアクセス権限を に付与します。
- AmazonMWAAWebServer アクセス — このポリシーを使用して、Apache Airflow ウェブサーバーへのアクセスを許可します。
- AmazonMWAAAirflowCli アクセス — このポリシーを使用して、Apache Airflow CLI コマンドを実行するためのアクセスを許可します。

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

- のユーザーとグループ AWS IAM Identity Center :

  アクセス許可セットを作成します。「AWS IAM Identity Center ユーザーガイド」の「権限設定を作成する」の手順に従ってください。
- IAM 内で、ID プロバイダーによって管理されているユーザー:

  ID フェデレーションの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) 用のロールの作成」を参照してください。
- IAM ユーザー:
  - ユーザーが担当できるロールを作成します。手順については、「IAM ユーザーガイド」の「IAM ユーザー用ロールの作成」を参照してください。
  - (お奨めできない方法) ポリシーをユーザーに直接アタッチするか、ユーザーをユーザーグループに追加する。詳細については、「IAM ユーザーガイド」の「ユーザー (コンソール) へのアクセス権限の追加」を参照してください。

フルコンソールアクセスポリシー: AmazonMWAAFullConsole Access

Amazon MWAA コンソールで環境を構成する必要がある場合、ユーザーは `AmazonMWAAFullConsoleAccess` アクセス権限ポリシーにアクセスする必要がある場合があります。

> **Note**
>
> コンソールを通じたアクセスポリシーには、iam:PassRole を実行するためのアクセス許可が含まれている必要があります。これにより、ユーザーは「サービスにリンクされたロール」と「実行ロール」を Amazon MWAA に渡すことができます。Amazon MWAA は、ユーザーに代わって他の AWS サービスを呼び出すために、各ロールを引き受けます。次の例では、iam:PassedToService 条件キーを使用して、ロールを渡すことができるサービスとして Amazon MWAA サービスプリンシパル (airflow.amazonaws.com) を指定します。の詳細についてはiam:PassRole、「IAM ユーザーガイド」の「 AWS サービスにロールを渡すアクセス許可をユーザーに付与する」を参照してください。

「保管時の暗号化」に「AWS 所有のキー」を使用して Amazon MWAA 環境を作成、管理する場合は、次のポリシーを使用してください。

の使用 AWS 所有のキー

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":"airflow:*",
         "Resource":"*"
      },
      {
         "Effect":"Allow",
         "Action":[
            "iam:PassRole"
         ],
         "Resource":"*",
         "Condition":{
            "StringLike":{
               "iam:PassedToService":"airflow.amazonaws.com"
            }
         }
      },
      {
         "Effect":"Allow",
         "Action":[
            "iam:ListRoles"
```

```
         ],
         "Resource":"*"
      },
      {
         "Effect":"Allow",
         "Action":[
            "iam:CreatePolicy"
         ],
         "Resource":"arn:aws:iam::YOUR_ACCOUNT_ID:policy/service-role/MWAA-Execution-
Policy*"
      },
            {
         "Effect":"Allow",
         "Action":[
            "iam:AttachRolePolicy",
            "iam:CreateRole"
         ],
         "Resource":"arn:aws:iam::YOUR_ACCOUNT_ID:role/service-role/AmazonMWAA*"
      },
      {
         "Effect":"Allow",
         "Action":[
            "iam:CreateServiceLinkedRole"
         ],
         "Resource":"arn:aws:iam::*:role/aws-service-role/airflow.amazonaws.com/
AWSServiceRoleForAmazonMWAA"
      },
      {
         "Effect":"Allow",
         "Action":[
            "s3:GetBucketLocation",
            "s3:ListAllMyBuckets",
            "s3:ListBucket",
            "s3:ListBucketVersions"
         ],
         "Resource":"*"
      },
      {
         "Effect":"Allow",
         "Action":[
            "s3:CreateBucket",
            "s3:PutObject",
            "s3:GetEncryptionConfiguration"
         ],
```

```
            "Resource":"arn:aws:s3:::*"
         },
         {
            "Effect":"Allow",
            "Action":[
               "ec2:DescribeSecurityGroups",
               "ec2:DescribeSubnets",
               "ec2:DescribeVpcs",
               "ec2:DescribeRouteTables"
            ],
            "Resource":"*"
         },
         {
            "Effect":"Allow",
            "Action":[
               "ec2:AuthorizeSecurityGroupIngress",
               "ec2:CreateSecurityGroup"
            ],
            "Resource":"arn:aws:ec2:*:*:security-group/airflow-security-group-*"
         },
         {
            "Effect":"Allow",
            "Action":[
               "kms:ListAliases"
            ],
            "Resource":"*"
         },
         {
            "Effect":"Allow",
            "Action":"ec2:CreateVpcEndpoint",
            "Resource":[
               "arn:aws:ec2:*:*:vpc-endpoint/*",
               "arn:aws:ec2:*:*:vpc/*",
               "arn:aws:ec2:*:*:subnet/*",
               "arn:aws:ec2:*:*:security-group/*"
            ]
         },
         {
            "Effect":"Allow",
            "Action":[
               "ec2:CreateNetworkInterface"
            ],
            "Resource":[
               "arn:aws:ec2:*:*:subnet/*",
```

```
            "arn:aws:ec2:*:*:network-interface/*"
         ]
      }
   ]
}
```

保管時の暗号化にカスタマーマネージドキーを使用して Amazon MWAA 環境を作成、管理する場合は、次のポリシーを使用してください。カスタマーマネージドキーを使用するには、IAM プリンシパルに、アカウントに保存されているキーを使用して AWS KMS リソースにアクセスするアクセス許可が必要です。

カスタマーマネージドキーを使用する

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":"airflow:*",
         "Resource":"*"
      },
      {
         "Effect":"Allow",
         "Action":[
            "iam:PassRole"
         ],
         "Resource":"*",
         "Condition":{
            "StringLike":{
               "iam:PassedToService":"airflow.amazonaws.com"
            }
         }
      },
      {
         "Effect":"Allow",
         "Action":[
            "iam:ListRoles"
         ],
         "Resource":"*"
      },
      {
         "Effect":"Allow",
         "Action":[
```

```
            "iam:CreatePolicy"
         ],
         "Resource":"arn:aws:iam::YOUR_ACCOUNT_ID:policy/service-role/MWAA-Execution-
Policy*"
      },
            {
         "Effect":"Allow",
         "Action":[
            "iam:AttachRolePolicy",
            "iam:CreateRole"
         ],
         "Resource":"arn:aws:iam::YOUR_ACCOUNT_ID:role/service-role/AmazonMWAA*"
      },
      {
         "Effect":"Allow",
         "Action":[
            "iam:CreateServiceLinkedRole"
         ],
         "Resource":"arn:aws:iam::*:role/aws-service-role/airflow.amazonaws.com/
AWSServiceRoleForAmazonMWAA"
      },
      {
         "Effect":"Allow",
         "Action":[
            "s3:GetBucketLocation",
            "s3:ListAllMyBuckets",
            "s3:ListBucket",
            "s3:ListBucketVersions"
         ],
         "Resource":"*"
      },
      {
         "Effect":"Allow",
         "Action":[
            "s3:CreateBucket",
            "s3:PutObject",
            "s3:GetEncryptionConfiguration"
         ],
         "Resource":"arn:aws:s3:::*"
      },
      {
         "Effect":"Allow",
         "Action":[
            "ec2:DescribeSecurityGroups",
```

```
            "ec2:DescribeSubnets",
            "ec2:DescribeVpcs",
            "ec2:DescribeRouteTables"
         ],
         "Resource":"*"
      },
      {
         "Effect":"Allow",
         "Action":[
            "ec2:AuthorizeSecurityGroupIngress",
            "ec2:CreateSecurityGroup"
         ],
         "Resource":"arn:aws:ec2:*:*:security-group/airflow-security-group-*"
      },
      {
         "Effect":"Allow",
         "Action":[
            "kms:ListAliases"
         ],
         "Resource":"*"
      },
      {
         "Effect":"Allow",
         "Action":[
            "kms:DescribeKey",
            "kms:ListGrants",
            "kms:CreateGrant",
            "kms:RevokeGrant",
            "kms:Decrypt",
            "kms:Encrypt",
            "kms:GenerateDataKey*",
            "kms:ReEncrypt*"
         ],
         "Resource":"arn:aws:kms:*:YOUR_ACCOUNT_ID:key/YOUR_KMS_ID"
      },
      {
         "Effect":"Allow",
         "Action":"ec2:CreateVpcEndpoint",
         "Resource":[
            "arn:aws:ec2:*:*:vpc-endpoint/*",
            "arn:aws:ec2:*:*:vpc/*",
            "arn:aws:ec2:*:*:subnet/*",
            "arn:aws:ec2:*:*:security-group/*"
         ]
```

```
      },
      {
         "Effect":"Allow",
         "Action":[
            "ec2:CreateNetworkInterface"
         ],
         "Resource":[
            "arn:aws:ec2:*:*:subnet/*",
            "arn:aws:ec2:*:*:network-interface/*"
         ]
      }
   ]
}
```

API とコンソールのフルアクセスポリシー: AmazonMWAAFullApi Access

環境の管理に使用されるすべての Amazon MWAA API にアクセスする必要がある場合、AmazonMWAAFullApiAccess ユーザーはアクセス権限ポリシーにアクセスする必要がある場合があります。Apache Airflow UI にアクセスするための許可は付与されません。

> **Note**
>
> API フルアクセスポリシーには、iam:PassRole を実行する権限が含まれている必要があります。これにより、ユーザーは「サービスにリンクされたロール」と「実行ロール」を Amazon MWAA に渡すことができます。Amazon MWAA は、ユーザーに代わって他の AWS サービスを呼び出すために、各ロールを引き受けます。次の例では、iam:PassedToService 条件キーを使用して、ロールを渡すことができるサービスとして Amazon MWAA サービスプリンシパル (airflow.amazonaws.com) を指定します。の詳細についてはiam:PassRole、「IAM ユーザーガイド」の「 AWS サービスにロールを渡すアクセス許可をユーザーに付与する」を参照してください。

保管時の AWS 所有のキー 暗号化に を使用して Amazon MWAA 環境を作成および管理する場合は、次のポリシーを使用します。

の使用 AWS 所有のキー

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

```
         "Effect":"Allow",
         "Action":"airflow:*",
         "Resource":"*"
      },
      {
         "Effect":"Allow",
         "Action":[
            "iam:PassRole"
         ],
         "Resource":"*",
         "Condition":{
            "StringLike":{
               "iam:PassedToService":"airflow.amazonaws.com"
            }
         }
      },
      {
         "Effect":"Allow",
         "Action":[
            "iam:CreateServiceLinkedRole"
         ],
         "Resource":"arn:aws:iam::*:role/aws-service-role/airflow.amazonaws.com/
AWSServiceRoleForAmazonMWAA"
      },
      {
         "Effect":"Allow",
         "Action":[
            "ec2:DescribeSecurityGroups",
            "ec2:DescribeSubnets",
            "ec2:DescribeVpcs",
            "ec2:DescribeRouteTables"
         ],
         "Resource":"*"
      },
      {
         "Effect":"Allow",
         "Action":[
            "s3:GetEncryptionConfiguration"
         ],
         "Resource":"arn:aws:s3:::*"
      },
      {
         "Effect":"Allow",
         "Action":"ec2:CreateVpcEndpoint",
```

```
         "Resource":[
            "arn:aws:ec2:*:*:vpc-endpoint/*",
            "arn:aws:ec2:*:*:vpc/*",
            "arn:aws:ec2:*:*:subnet/*",
            "arn:aws:ec2:*:*:security-group/*"
         ]
      },
      {
         "Effect":"Allow",
         "Action":[
            "ec2:CreateNetworkInterface"
         ],
         "Resource":[
            "arn:aws:ec2:*:*:subnet/*",
            "arn:aws:ec2:*:*:network-interface/*"
         ]
      }
   ]
}
```

保存時の暗号化のためにカスタマーマネージドキーを使用して Amazon MWAA 環境を作成、管理する場合は、次のポリシーを使用してください。カスタマーマネージドキーを使用するには、IAM プリンシパルに、アカウントに保存されているキーを使用して AWS KMS リソースにアクセスするアクセス許可が必要です。

カスタマーマネージドキーを使用する

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":"airflow:*",
         "Resource":"*"
      },
      {
         "Effect":"Allow",
         "Action":[
            "iam:PassRole"
         ],
         "Resource":"*",
         "Condition":{
            "StringLike":{
```

```
                "iam:PassedToService":"airflow.amazonaws.com"
            }
         }
      },
      {
         "Effect":"Allow",
         "Action":[
            "iam:CreateServiceLinkedRole"
         ],
         "Resource":"arn:aws:iam::*:role/aws-service-role/airflow.amazonaws.com/
AWSServiceRoleForAmazonMWAA"
      },
      {
         "Effect":"Allow",
         "Action":[
            "ec2:DescribeSecurityGroups",
            "ec2:DescribeSubnets",
            "ec2:DescribeVpcs",
            "ec2:DescribeRouteTables"
         ],
         "Resource":"*"
      },
      {
         "Effect":"Allow",
         "Action":[
            "kms:DescribeKey",
            "kms:ListGrants",
            "kms:CreateGrant",
            "kms:RevokeGrant",
            "kms:Decrypt",
            "kms:Encrypt",
            "kms:GenerateDataKey*",
            "kms:ReEncrypt*"
         ],
         "Resource":"arn:aws:kms:*:YOUR_ACCOUNT_ID:key/YOUR_KMS_ID"
      },
      {
         "Effect":"Allow",
         "Action":[
            "s3:GetEncryptionConfiguration"
         ],
         "Resource":"arn:aws:s3:::*"
      },
      {
```

```
         "Effect":"Allow",
         "Action":"ec2:CreateVpcEndpoint",
         "Resource":[
            "arn:aws:ec2:*:*:vpc-endpoint/*",
            "arn:aws:ec2:*:*:vpc/*",
            "arn:aws:ec2:*:*:subnet/*",
            "arn:aws:ec2:*:*:security-group/*"
         ]
      },
      {
         "Effect":"Allow",
         "Action":[
            "ec2:CreateNetworkInterface"
         ],
         "Resource":[
            "arn:aws:ec2:*:*:subnet/*",
            "arn:aws:ec2:*:*:network-interface/*"
         ]
      }
   ]
}
```

読み取り専用コンソールアクセスポリシー: AmazonMWAAReadOnlyAccess

Amazon MWAA コンソールの環境詳細ページで環境が使用しているリソースを確認する必要がある場合、AmazonMWAAReadOnlyAccess ユーザーはアクセス許可ポリシーにアクセスする必要がある場合があります。ユーザーが新しい環境を作成したり、既存の環境を編集したり、Apache Airflow UI を表示したりすることはできません。

```
{
        "Version": "2012-10-17",
        "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "airflow:ListEnvironments",
                "airflow:GetEnvironment",
                "airflow:ListTagsForResource"
            ],
            "Resource": "*"
        }
```

```
    ]
}
```

Apache Airflow UI アクセスポリシー: AmazonMWAAWebServer アクセス

Apache Airflow UI にアクセスする必要がある場合、AmazonMWAAWebServerAccess ユーザーはアクセス権限ポリシーにアクセスする必要がある場合があります。ユーザーが Amazon MWAA コンソールで環境を表示したり、Amazon MWAA API を使用してアクションを実行したりすることはできません。{airflow-role} で Admin 、Op 、User 、Viewer または Public ロールを指定して、ウェブトークンのユーザーのアクセスレベルをカスタマイズします。詳細については、Apache Airflow リファレンスガイドの「デフォルトロール」を参照してください。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "airflow:CreateWebLoginToken",
            "Resource": [
                "arn:aws:airflow:{your-region}:YOUR_ACCOUNT_ID:role/{your-environment-name}/{airflow-role}"
            ]
        }
    ]
}
```

> **Note**
>
> Amazon MWAA は、5 つの「デフォルトの Apache Airflow ロールベースのアクセスコントロール (RBAC) ロール」と IAM 統合を提供します。カスタム Apache Airflow ロールの使用詳細については、「the section called “チュートリアル:ユーザーを DAG のサブセットに制限する”」を参照してください。

Apache Airflow CLI ポリシー: AmazonMWAAAirflowCli アクセス

Apache Airflow CLI コマンド (trigger_dag など) を実行する必要がある場合、ユーザーは AmazonMWAAAirflowCliAccess アクセス権限ポリシーにアクセスする必要がある場合がありま

す。ユーザーが Amazon MWAA コンソールで環境を表示したり、Amazon MWAA API を使用してアクションを実行したりすることはできません。

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

JSON ポリシーの作成

JSON ポリシーを作成し、IAM コンソールでそのポリシーをユーザー、ロール、またはグループにアタッチできます。IAM での JSON ポリシーの作成方法については、以下のステップで示します。

JSON ポリシーを作成する方法

1. IAM コンソールで「ポリシーページ」を開きます。
2. [Create policy] (ポリシーを作成) を選択します。
3. [JSON] タブを選択します。
4. JSON ポリシーを追加します。
5. [ポリシーの確認] を選択します。
6. [名前] と [説明] (オプション) のテキストフィールドに値を入力します。

   たとえば、ポリシーには `AmazonMWAAReadOnlyAccess` という名前を付けることができます。
7. [ポリシーの作成] を選択します。

開発者グループにポリシーをアタッチするユースケースの例

Apache Airflow 開発チームのすべての開発者に権限を適用するため、`AirflowDevelopmentGroup` というIAM のグループを使用しているとしましょう。これらのユーザーは `AmazonMWAAFullConsoleAccess`、`AmazonMWAAAirflowCliAccess` および

AmazonMWAAWebServerAccess の権限ポリシーにアクセスする必要があります。このセクションでは、IAM でグループを作成し、ポリシーを作成してアタッチし、そのグループを IAM ユーザーに関連付ける方法について説明します。この手順では、「AWS 」が所有するキーを使用する場合を想定しています。

AmazonMWAAFullConsoleAccess ポリシーを作成するには

1. AmazonMWAAFullConsoleAccess アクセスポリシー をダウンロードします。
2. IAM コンソールで「ポリシーページ」を開きます。
3. [Create policy] (ポリシーを作成) を選択します。
4. [JSON] タブを選択します。
5. AmazonMWAAFullConsoleAccess の JSON ポリシーを貼り付けます。
6. 以下の値を置き換えます。
    a. *{your-account-id}* – AWS アカウント ID (など0123456789)
    b. *{your-kms-id}* — カスタマーマネージドキーの固有識別子。保存時の暗号化にカスタマーマネージドキーを使用する場合にのみ適用されます。
7. [ポリシーの確認] を選択します。
8. [名前] に AmazonMWAAFullConsoleAccess と入力します。
9. [ポリシーの作成] を選択します。

AmazonMWAAWebServerAccess ポリシーを作成するには

1. AmazonMWAAWebServerAccess アクセスポリシー をダウンロードします。
2. IAM コンソールで「ポリシーページ」を開きます。
3. [Create policy] (ポリシーを作成) を選択します。
4. [JSON] タブを選択します。
5. AmazonMWAAWebServerAccess の JSON ポリシーを貼り付けます。
6. 以下の値を置き換えます。
    a. *{your-region}* — Amazon MWAA 環境のリージョン (us-east-1 など)
    b. *{your-account-id}* – AWS アカウント ID (など0123456789)
    c. *{your-environment-name}* — Amazon MWAA 環境名 (MyAirflowEnvironment など)
    d. *{airflow-role}* — Admin Apache Airflowの「デフォルトロール」

7. [ポリシーの確認] を選択します。
8. [名前] に AmazonMWAAWebServerAccess と入力します。
9. [ポリシーの作成] を選択します。

AmazonMWAAAirflowCliAccess ポリシーを作成するには

1. AmazonMWAAAirflowCliAccess アクセスポリシー をダウンロードします。
2. IAM コンソールで「ポリシーページ」を開きます。
3. [Create policy] (ポリシーを作成) を選択します。
4. [JSON] タブを選択します。
5. AmazonMWAAAirflowCliAccess の JSON ポリシーを貼り付けます。
6. [ポリシーの確認] を選択します。
7. [名前] に AmazonMWAAAirflowCliAccess と入力します。
8. [ポリシーの作成] を選択します。

グループを作成するには

1. IAM コンソールで「グループページ」を開きます。
2. AirflowDevelopmentGroup の名前を入力します。
3. [Next Step (次のステップ)] をクリックします。
4. [フィルター] に AmazonMWAA を入力して結果を絞り込みます。
5. 作成した 3 つのポリシーを選択します。
6. [Next Step (次のステップ)] をクリックします。
7. グループを作成 を選択します。

ユーザーに関連付ける手順

1. IAM コンソールで「ユーザーページ」を開きます。
2. ユーザーを選択します。
3. [グループ] を選択します。
4. [グループにユーザーを追加] を選択します。

5. AirflowDevelopmentグループ を選択します。
6. 続いて、[Add to Groups] (グループに追加) を選択します。

次のステップ

- 「Apache Airflow へのアクセス」で Apache Airflow UI にアクセスするトークンを生成する方法について説明します。
- IAM ポリシーの作成の詳細については、「IAM ポリシーの作成」を参照してください。

Amazon MWAA のサービスにリンクされたロール

Amazon Managed Workflows for Apache Airflow 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 MWAA 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 MWAA 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。

必要な許可を手動で追加する必要がないため、サービスリンクロールは Amazon MWAA のセットアップを容易にします。サービスリンクロールの許可は Amazon MWAA が定義し、特に定義されない限り、Amazon MWAA のみがそのロールを引き受けることができます。定義したアクセス許可には、信頼ポリシーと許可ポリシーが含まれます。こ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。

サービスリンクロールは、まずその関連リソースを削除しなければ削除できません。これは、リソースにアクセスするための許可を不用意に削除できないため、Amazon MWAA リソースを保護できます。

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 と連動するAWS のサービス」を参照し、Service-linked roles (サービスにリンクされたロール) の列内で Yes (はい) と表記されたサービスを確認してください。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 MWAA のサービスリンクロール許可

Amazon MWAA は、 という名前のサービスにリンクされたロールを使用しますAWSServiceRoleForAmazonMWAA。 アカウントで作成されたサービスにリンクされたロールは、Amazon MWAA に次の AWS サービスへのアクセスを許可します。

- Amazon CloudWatch Logs (CloudWatch ログ) – Apache Airflow ログのロググループを作成します。
- Amazon CloudWatch （ CloudWatch) – 環境とその基盤となるコンポーネントに関連するメトリクスをアカウントに公開します。
- Amazon Elastic Compute Cloud (Amazon EC2) — 以下のリソースを作成するには:
  - Apache Airflow スケジューラとワーカー が使用する AWSマネージド Amazon Aurora PostgreSQL データベースクラスターの VPC 内の Amazon VPC エンドポイント。
  - Apache Airflow Web serverの「プライベートネットワーク」オプションを選択した場合に、Web serverへのネットワークアクセスを有効にする追加の Amazon VPC エンドポイント。
  - Amazon VPC でホストされている AWS リソースへのネットワークアクセスを有効にするための Amazon VPC ENIs)。

次の信頼ポリシーは、サービスプリンシパルがサービスにリンクされたロールを引き受けることを許可します。Amazon MWAA のサービスプリンシパルは、ポリシーに示されているとおり `airflow.amazonaws.com` です。

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

`AmazonMWAAServiceRolePolicy` という名前のロールのアクセス許可ポリシーは、指定したリソースに対して以下のアクションを実行することを Amazon MWAA に許可します。

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

```
                "logs:CreateLogStream",
                "logs:CreateLogGroup",
                "logs:DescribeLogGroups"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:airflow-*:*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:AttachNetworkInterface",
                "ec2:CreateNetworkInterface",
                "ec2:CreateNetworkInterfacePermission",
                "ec2:DeleteNetworkInterface",
                "ec2:DeleteNetworkInterfacePermission",
                "ec2:DescribeDhcpOptions",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcEndpoints",
                "ec2:DescribeVpcs",
                "ec2:DetachNetworkInterface"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "ec2:CreateVpcEndpoint",
            "Resource": "arn:aws:ec2:*:*:vpc-endpoint/*",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "aws:TagKeys": "AmazonMWAAManaged"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:ModifyVpcEndpoint",
                "ec2:DeleteVpcEndpoints"
            ],
            "Resource": "arn:aws:ec2:*:*:vpc-endpoint/*",
            "Condition": {
                "Null": {
                    "aws:ResourceTag/AmazonMWAAManaged": false
```

```
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateVpcEndpoint",
                "ec2:ModifyVpcEndpoint"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:vpc/*",
                "arn:aws:ec2:*:*:security-group/*",
                "arn:aws:ec2:*:*:subnet/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "ec2:CreateTags",
            "Resource": "arn:aws:ec2:*:*:vpc-endpoint/*",
            "Condition": {
                "StringEquals": {
                    "ec2:CreateAction": "CreateVpcEndpoint"
                },
                "ForAnyValue:StringEquals": {
                    "aws:TagKeys": "AmazonMWAAManaged"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "cloudwatch:PutMetricData",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "cloudwatch:namespace": [
                        "AWS/MWAA"
                    ]
                }
            }
        }
    ]
}
```

サービスリンクロールの作成、編集、削除を IAM エンティティ (ユーザー、グループ、ロールなど) 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「サービスにリンクされたロールのアクセス許可」を参照してください。

Amazon MWAA のサービスリンクロールの作成

サービスリンクロールを手動で作成する必要はありません。 AWS Management Console、、 AWS CLIまたは AWS API を使用して新しい Amazon MWAA 環境を作成すると、Amazon MWAA によってサービスにリンクされたロールが作成されます。

このサービスリンクロールを削除した後で再度作成する必要が生じた場合は、同じ方法でアカウントにロールを再作成できます。別の環境を作成すると、Amazon MWAA によって、サービスにリンクされたロールが再度作成されます。

Amazon MWAA のサービスリンクロールの編集

Amazon MWAA では、 AWSServiceRoleForAmazonMWAA 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。サービスリンクロールを作成した後は、多くのエンティティによってロールが参照される可能性があるため、ロール名を変更することはできません。ただし、IAM を使用したロール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 MWAA のサービスリンクロールの削除

サービスリンクロールが必要な機能またはサービスが不要になった場合には、そのロールを削除することをお勧めします。そうすることで、モニタリングや保守が積極的に行われていない未使用のエンティティを排除できます。

Amazon MWAA 環境を削除すると、Amazon MWAA はサービスの一部として使用する関連リソースをすべて削除します。ただし、Amazon MWAA が環境の削除を完了するまで待ってから、サービスにリンクされたロールを削除する必要があります。Amazon MWAA が環境を削除する前にサービスにリンクされたロールを削除すると、Amazon MWAA は環境に関連するリソースをすべて削除できなくなる可能性があります。

サービスにリンクされたロールを IAM で手動削除するには

IAM コンソール、、または AWS API を使用して AWS CLI、サービスにリンクされたロールを削除します AWSServiceRoleForAmazonMWAA。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 MWAA サービスリンクロールがサポートされるリージョン

Amazon MWAA は、サービスが利用可能なすべてのリージョンでサービスにリンクされたロールの使用をサポートします。詳細は、Amazon Managed Workflows for Apache Airflow エンドポイントとクオータを参照してください。

ポリシーの更新

変更	説明	日付
Amazon MWAA は、サービスにリンクされたロールの権限ポリシーを更新	「AmazonMWAAServiceRolePolicy」— Amazon MWAA は、サービスにリンクされたロールのアクセス権限ポリシーを更新し、サービスの基盤となるリソースに関連する追加のメトリクスをカスタマーアカウントに公開する権限を Amazon MWAA に付与します。これらの新しいメトリクスは、AWS/MWAA 以下で公開されます。	2022 年 11 月 18 日
Amazon MWAA が変更の追跡を開始しました。	Amazon MWAA は、 AWS マネージドサービスにリンクされたロールのアクセス許可ポリシーの変更の追跡を開始しました。	2022 年 11 月 18 日

Amazon MWAA 実行ロール

実行ロールは、ユーザーに代わって他の AWS サービスのリソースを呼び出すアクセス許可を Amazon Managed Workflows for Apache Airflow に付与するアクセス許可ポリシーを持つ AWS Identity and Access Management (IAM) ロールです。これには、Amazon S3 バケット、 AWS 所有キー、 CloudWatch ログなどのリソースが含まれます。Amazon MWAA 環境には、環境ごとに 1 つ

の実行ロールが必要です。このページでは、Amazon MWAA が環境で使用される他の AWS リソースにアクセスできるように、環境の実行ロールを使用および設定する方法について説明します。

目次

実行ロールの概要

Amazon MWAA が環境で使用される他の AWS のサービスを使用するためのアクセス許可は、実行ロールから取得されます。Amazon MWAA 実行ロールには、環境で使用される以下の AWS サービスに対するアクセス許可が必要です。

- Amazon CloudWatch （ CloudWatch) – Apache Airflow メトリクスとログを送信します。
- Amazon Simple Storage Service (Amazon S3) — 環境の DAG コードとサポートファイル (`requirements.txt` など) を解析します。
- Amazon Simple Queue Service (Amazon SQS) — Amazon MWAA が所有する Amazon SQS キューに、環境の Apache Airflow タスクをキューイングします。
- AWS Key Management Service （ AWS KMS) – 環境のデータ暗号化 (AWS 所有キーまたはカスタマーマネージドキー を使用) 用。

> **Note**
>
> Amazon MWAA が AWS マネージド KMS キーを使用してデータを暗号化することを選択した場合は、Amazon SQS を介してアカウント外部に保存されている任意の KMS キーへのアクセスを許可する Amazon MWAA 実行ロールにアタッチされたポリシーでアクセス許可を定義する必要があります。環境の実行ロールが任意の KMS キーにアクセスするには、次の 2 つの条件が必要です。
>
> - サードパーティアカウントの KMS キーは、リソースポリシーを通じてクロスアカウントアクセスを許可する必要があります。
> - DAG コードは、サードパーティアカウントの `airflow-celery-` で始まる Amazon SQS キューにアクセスする必要があり、暗号化に同じ KMS キーを使用します。
>
> リソースへのクロスアカウントアクセスに関連するリスクを軽減するために、DAG に配置されたコードを見直して、ワークフローがアカウント外の任意の Amazon SQS キューにアクセスしていないことを確認することを推奨します。さらに、自分のアカウントに保存されているカスタマーマネージド KMS キーを使用して Amazon MWAA の暗号化を管理できます。これにより、環境の実行ロールがお使いのアカウントの KMS キーのみにアクセスするように制限されます。
>
> 暗号化オプションを選択した後は、既存の環境では選択内容を変更できないことに注意してください。

実行ロールには、次の IAM アクションに対する権限も必要です。

- `airflow:PublishMetrics` — Amazon MWAA が環境の状態を監視できるようにするため。

権限はデフォルトで付与されます。

Amazon MWAA コンソールのデフォルトオプションを使用して実行ロールと「AWS 所有キー」を作成し、このページの手順を使用して実行ロールにアクセス権限ポリシーを追加できます。

- コンソールで [新規ロールの作成] オプションを選択すると、Amazon MWAA は環境に必要な最小限のアクセス権限を実行ロールにアタッチします。
- Amazon MWAA が最大限のアクセス権限を割り当てる場合もあります。たとえば、環境を作成するときに Amazon MWAA コンソールのオプションを選択して実行ロールを作成することを推奨します。Amazon MWAA は、実行ロールの正規表現パターンを として使用して、すべての CloudWatch Logs グループのアクセス許可ポリシーを自動的に追加しま

す"arn:aws:logs:your-region:your-account-id:log-group:airflow-your-environment-name-*"。

他の AWS のサービスを使用するためのアクセス許可を追加する方法

Amazon MWAA は、環境の作成後に既存の実行ロールにアクセス権限ポリシーを追加または編集することはできません。環境に必要な追加のアクセス権限ポリシーを使用して実行ロールを更新する必要があります。例えば、DAG が へのアクセスを必要とする場合 AWS Glue、Amazon MWAA は環境に必要なこれらのアクセス許可を自動的に検出したり、実行ロールにアクセス許可を追加したりすることはできません。

実行ロールにアクセス許可は、次の 2 つのページから追加できます。

- 実行ロールの JSON ポリシーをインラインで変更します。このページにあるサンプル「JSON ポリシードキュメント」を使用して、IAM コンソールの実行ロールの JSON ポリシーを追加したり、置き換えたりできます。
- AWS サービスの JSON ポリシーを作成し、実行ロールにアタッチします。このページのステップを使用して、 AWS サービスの新しい JSON ポリシードキュメントを IAM コンソールの実行ロールに関連付けることができます。

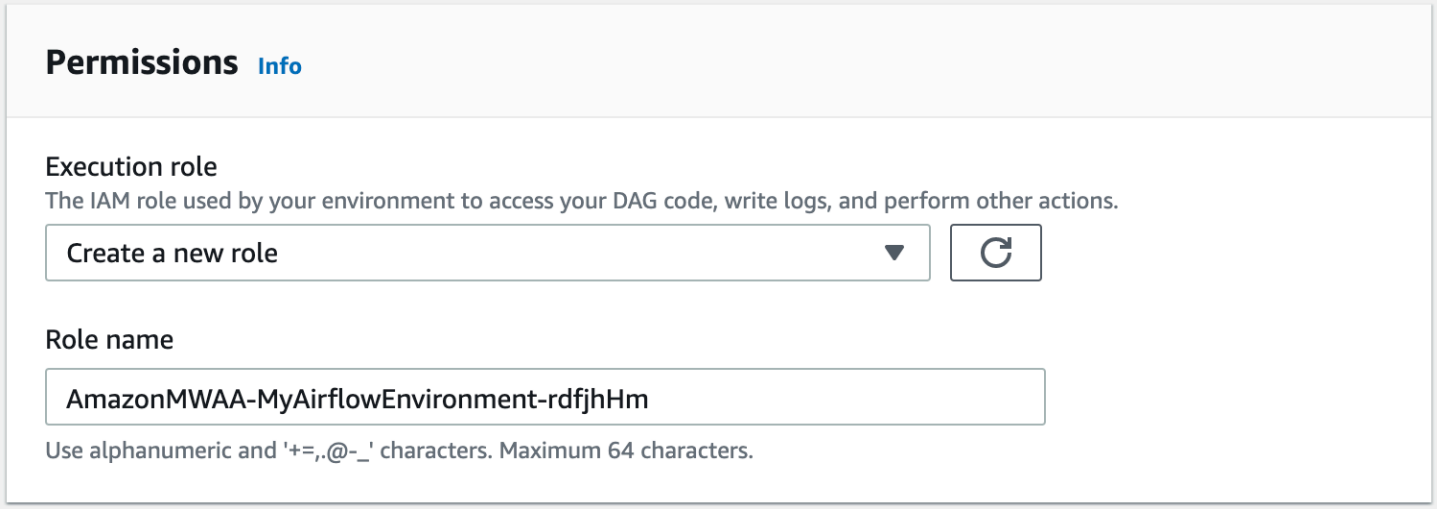
実行ロールが既に環境に関連付けられていると仮定すると、Amazon MWAA は追加されたアクセス権限ポリシーをすぐに使い始めることができます。つまり、実行ロールから必要なアクセス権限を削除すると、お使いのDAG が機能しなくなる可能性があるということです。

新しい実行ロールを関連付ける方法

お使いの環境の実行ロールはいつでも変更できます。新しい実行ロールがまだお使いの環境に関連付けられていない場合は、このページの手順を使用して新しい実行ロールポリシーを作成し、そのロールを環境に関連付けます。

新規ロールの作成

デフォルトでは、Amazon MWAA はデータ暗号化用の「AWS 所有キー」とユーザーに代わって実行ロールを作成します。環境を作成するときに Amazon MWAA コンソールでデフォルトオプションを選択できます。次のイメージは、環境の実行ロールを作成するデフォルトオプションを示しています。



実行ロールポリシーの表示および更新

Amazon MWAA コンソールで環境の実行ロールを確認し、IAM コンソールでロールの JSON ポリシーを更新できます。

実行ロールポリシーを更新するには

1. Amazon MWAA コンソールで「環境ページ」を開きます。
2. 環境を選択します。
3. [権限] ペインで実行ロールを選択し、IAM の権限ページを開きます。
4. 実行ロールの名前を選択し、権限ポリシーを開きます。
5. [Edit policy] (ポリシーの編集) を選択します。
6. [JSON] タブを選択します。
7. JSON ポリシーを更新します。
8. [ポリシーの確認] を選択します。
9. [変更の保存] をクリックします。

JSON ポリシーをアタッチして他の AWS のサービスを使用する

AWS サービスの JSON ポリシーを作成し、実行ロールにアタッチできます。たとえば、次の JSON ポリシーをアタッチして、 AWS Secrets Managerのすべてのリソースへの読み取り専用アクセスレベルを付与することができます。

```
{
```

```
    "Version":"2012-10-17",
    "Statement":[
       {
          "Effect":"Allow",
          "Action":[
             "secretsmanager:GetResourcePolicy",
             "secretsmanager:GetSecretValue",
             "secretsmanager:DescribeSecret",
             "secretsmanager:ListSecretVersionIds"
          ],
          "Resource":[
             "*"
          ]
       }
    ]
}
```

実行ロールにポリシーを添付するには

1. Amazon MWAA コンソールで「環境ページ」を開きます。
2. 環境を選択します。
3. [権限] ペインで実行ロールを選択します。
4. [ポリシーのアタッチ] を選択します。
5. [Create policy] を選択します。
6. [JSON] を選択します。
7. JSON ポリシーを貼り付けます。
8. [次の手順: タグ]、[次の手順: 確認] の順に選択します。
9. わかりやすい名前 (`SecretsManagerReadPolicy` など) とポリシーの説明を入力します。
10. [ポリシーの作成] を選択します。

アカウントレベルのパブリックアクセスブロックで Amazon S3 バケットへのアクセスを許可する

「`PutPublicAccessBlock`」Amazon S3 オペレーションを使用して、アカウントのすべてのバケットへのアクセスをブロックしたい場合があります。アカウントのすべてのバケットへのアクセスをブロックする場合、環境実行ロールはその `s3:GetAccountPublicAccessBlock` アクションをアクセス権限ポリシーに含める必要があります。

次の例は、アカウントのすべての Amazon S3 バケットへのアクセスをブロックするときに実行ロールにアタッチする必要があるポリシーを示しています。

```
{
  "Version": "2012-10-17",
  "Statement": [
     {
       "Effect": "Allow",
       "Action": "s3:GetAccountPublicAccessBlock",
       "Resource": "*"
     }
  ]
}
```

Amazon S3 バケットへのパブリックアクセスのブロックの詳細については、Amazon Simple Storage Service ユーザーガイドの「Amazon S3 ストレージへのパブリックアクセスのブロック」を参照してください。

Apache Airflow 接続を使用する

Apache Airflow 接続を作成し、Apache Airflow 接続オブジェクトで実行ロールとその ARN を指定することもできます。詳細については、「Apache エアフローへの接続の管理」を参照してください。

実行ロールのサンプル JSON ポリシー

このセクションのサンプルアクセス権限ポリシーは、既存の実行ロールに使用されているアクセス権限ポリシーを置き換える場合と、新しい実行ロールを作成して環境で使用する場合に使用できる 2 つのポリシーを示しています。これらのポリシーには、Apache Airflow ロググループ、「Amazon S3 バケット」、および「Amazon MWAA 環境」の「リソース ARN」プレースホルダーが含まれています。

ポリシー例をコピーし、サンプル ARN またはプレースホルダーを置き換えてから、JSON ポリシーを使用して実行ロールを作成または更新することを推奨します。たとえば、`{your-region}` を `us-east-1` に置き換えます。

カスタマーマネージドキーのポリシー例

次の例は、「カスタマーマネージドキー」に使用できる実行ロールポリシーを示しています。

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Deny",
            "Action": "s3:ListAllMyBuckets",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject*",
                "s3:GetBucket*",
                "s3:List*"
            ],
            "Resource": [
                "arn:aws:s3:::{your-s3-bucket-name}",
                "arn:aws:s3:::{your-s3-bucket-name}/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:CreateLogGroup",
                "logs:PutLogEvents",
                "logs:GetLogEvents",
                "logs:GetLogRecord",
                "logs:GetLogGroupFields",
                "logs:GetQueryResults"
            ],
            "Resource": [
                "arn:aws:logs:{your-region}:{your-account-id}:log-group:airflow-{your-
environment-name}-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "*"
            ]
        },
        {
```

```
            "Effect": "Allow",
            "Action": [
                "s3:GetAccountPublicAccessBlock"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "cloudwatch:PutMetricData",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "sqs:ChangeMessageVisibility",
                "sqs:DeleteMessage",
                "sqs:GetQueueAttributes",
                "sqs:GetQueueUrl",
                "sqs:ReceiveMessage",
                "sqs:SendMessage"
            ],
            "Resource": "arn:aws:sqs:{your-region}:*:airflow-celery-*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey",
                "kms:GenerateDataKey*",
                "kms:Encrypt"
            ],
            "Resource": "arn:aws:kms:{your-region}:{your-account-id}:key/{your-kms-cmk-
id}",
            "Condition": {
                "StringLike": {
                    "kms:ViaService": [
                        "sqs.{your-region}.amazonaws.com",
                        "s3.{your-region}.amazonaws.com"
                    ]
                }
            }
        }
```

```
    ]
}
```

次に、ユーザーに代わってアクションを実行するために、Amazon MWAA がこのロールを引き受けることを許可する必要があります。そのためには、「IAM コンソールを使用」してこの実行ロールの信頼できるエンティティのリストに "airflow.amazonaws.com" と "airflow-env.amazonaws.com" サービスプリンシパルを追加するか、または AWS CLIを使用して IAM「create-role」コマンドを介して、これらのサービスプリンシパルをこの実行ロールのロール割り当てポリシードキュメントに配置します。ロール割り当てポリシーのサンプルは以下にあります。

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

次に、次の JSON ポリシーを「カスタマーマネージドキー」に添付します。このポリシーは、kms:EncryptionContext条件キープレフィックスを使用して、 CloudWatch Logs の Apache Airflow ロググループへのアクセスを許可します。

```
{
    "Sid": "Allow logs access",
    "Effect": "Allow",
    "Principal": {
        "Service": "logs.{your-region}.amazonaws.com"
    },
    "Action": [
        "kms:Encrypt*",
        "kms:Decrypt*",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:Describe*"
    ],
    "Resource": "*",
```

```
    "Condition": {
        "ArnLike": {
            "kms:EncryptionContext:aws:logs:arn": "arn:aws:logs:{your-region}:{your-
account-id}:*"
        }
    }
}
```

AWS 所有キーのサンプルポリシー

次の例は、「AWS カスタマーマネージドキー」に使用できる実行ロールポリシーを示しています。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "airflow:PublishMetrics",
            "Resource": "arn:aws:airflow:{your-region}:{your-account-id}:environment/
{your-environment-name}"
        },
        {
            "Effect": "Deny",
            "Action": "s3:ListAllMyBuckets",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject*",
                "s3:GetBucket*",
                "s3:List*"
            ],
            "Resource": [
                "arn:aws:s3:::{your-s3-bucket-name}",
                "arn:aws:s3:::{your-s3-bucket-name}/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:CreateLogGroup",
```

```
                "logs:PutLogEvents",
                "logs:GetLogEvents",
                "logs:GetLogRecord",
                "logs:GetLogGroupFields",
                "logs:GetQueryResults"
            ],
            "Resource": [
                "arn:aws:logs:{your-region}:{your-account-id}:log-group:airflow-{your-
environment-name}-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetAccountPublicAccessBlock"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "cloudwatch:PutMetricData",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "sqs:ChangeMessageVisibility",
                "sqs:DeleteMessage",
                "sqs:GetQueueAttributes",
                "sqs:GetQueueUrl",
                "sqs:ReceiveMessage",
                "sqs:SendMessage"
            ],
```

```
            "Resource": "arn:aws:sqs:{your-region}:*:airflow-celery-*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey",
                "kms:GenerateDataKey*",
                "kms:Encrypt"
            ],
            "NotResource": "arn:aws:kms:*:{your-account-id}:key/*",
            "Condition": {
                "StringLike": {
                    "kms:ViaService": [
                        "sqs.{your-region}.amazonaws.com"
                    ]
                }
            }
        }
    ]
}
```

次のステップ

- あなたと Apache Airflow ユーザーが「Amazon MWAA 環境へのアクセス」で環境にアクセスするために必要な権限について説明します。
- 暗号化のためのカスタマーマネージドキーの使用　についてはこちら。
- その他の「カスタマー管理ポリシー」の例をご覧ください。

サービス間での不分別な代理処理の防止

混乱した代理問題は、アクションを実行するためのアクセス許可を持たないエンティティが、より特権のあるエンティティにアクションの実行を強制できてしまう場合に生じる、セキュリティ上の問題です。では AWS、サービス間のなりすましにより、混乱した代理問題が発生する可能性があります。サービス間でのなりすましは、1 つのサービス (呼び出し元サービス) が、別のサービス (呼び出し対象サービス) を呼び出すときに発生する可能性があります。呼び出し元サービスは、本来ならアクセスすることが許可されるべきではない方法でその許可を使用して、別のお客様のリソースに対する処理を実行するように操作される場合があります。これを防ぐために、 AWS には、アカウント内

のリソースへのアクセス権が付与されたサービスプリンシパルですべてのサービスのデータを保護するために役立つツールが用意されています。

環境実行ロール内では aws:SourceArn および aws:SourceAccount のグローバル条件コンテキストキーを使用して、Amazon MWAA が別のサービスに付与する、リソースへのアクセス許可を制限することをお勧めします。クロスサービスアクセスにリソースを 1 つだけ関連付けたい場合は、aws:SourceArn を使用します。そのアカウント内のリソースをクロスサービスの使用に関連付けることを許可する場合は、aws:SourceAccount を使用します。

混乱した代理問題から保護するための最も効果的な方法は、リソースの完全な ARN を指定して、aws:SourceArn グローバル条件コンテキストキーを使用することです。リソースの完全な ARN が不明な場合や、複数のリソースを指定する場合には、グローバルコンテキスト条件キー aws:SourceArn で、ARN の未知部分を示すためにワイルドカード文字 (*) を使用します。例えば、arn:aws:airflow:*:*123456789012*:environment/* です。

aws:SourceArn の値は、実行ロールを作成している Amazon MWAA 環境 ARN である必要があります。

次の例では、お使いの環境内の実行ロール信頼ポリシーで aws:SourceArn および aws:SourceAccount グローバル条件コンテキストキーを使用して、「混乱した代理」問題を回避する方法を示します。新しい実行ロールを作成する際、次の信頼ポリシーを使用できます。

```
{
    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Principal": {
            "Service": ["airflow.amazonaws.com","airflow-env.amazonaws.com"]
        },
        "Action": "sts:AssumeRole",
        "Condition":{
            "ArnLike":{
               "aws:SourceArn":"arn:aws:airflow:your-
region:123456789012:environment/your-environment-name"
            },
            "StringEquals":{
               "aws:SourceAccount":"123456789012"
            }
         }
      }
```

```
    ]
}
```

Apache Airflow のアクセスモード

Apache Airflow 用 Amazon マネージドワークフローコンソールには、お使いの環境上の Apache Airflow ウェブサーバーへのプライベートルーティングまたはパブリックルーティングを構成するための組み込みオプションが含まれています。このガイドでは、Apache Airflow 用 Amazon マネージドワークフロー環境の Apache Airflow ウェブサーバーで使用できるアクセスモードと、プライベートネットワークオプションを選択した場合に Amazon VPC で構成する必要のある追加リソースについて説明します。

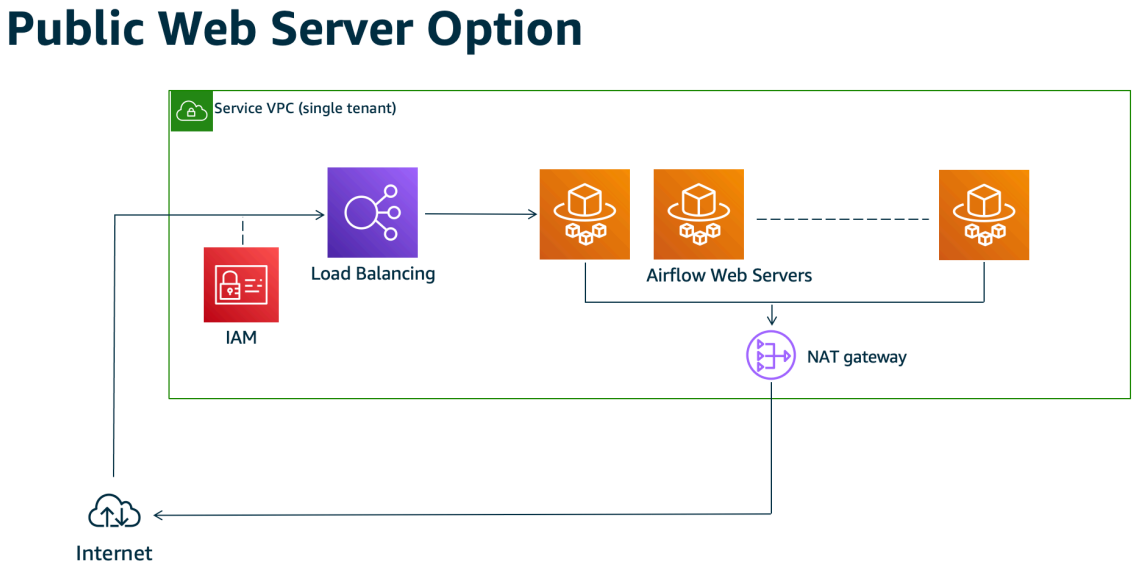
目次

Apache Airflow のアクセスモード

Apache Airflow ウェブサーバーのルーティングは、プライベートルーティングまたはパブリックルーティングを選択できます。プライベートルーティングを有効にするには、「プライベートネットワーク」を選択します。これにより、Apache Airflow ウェブサーバーへのユーザーアクセスは Amazon VPC 内のみに制限されます。パブリックルーティングを有効にするには、「パブリックネットワーク」を選択します。これにより、ユーザーはインターネット経由で Apache Airflow ウェブサーバーにアクセスできます。

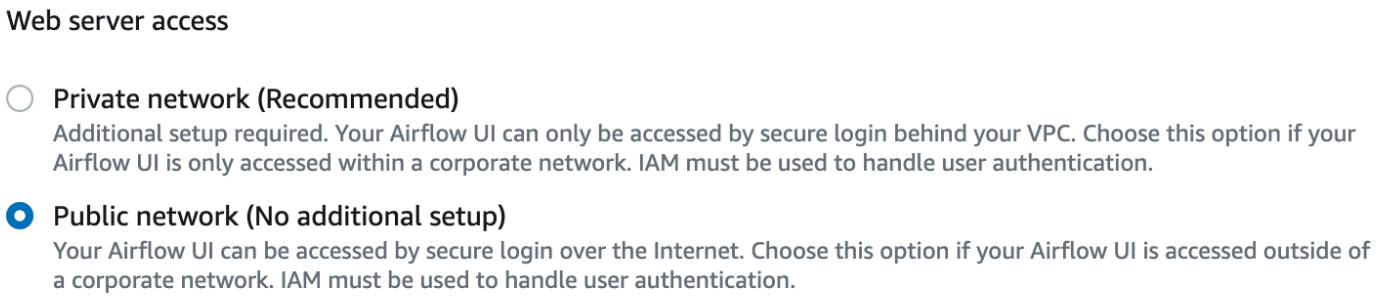
パブリックネットワーク

次のアーキテクチャ図は、パブリックウェブサーバーを使用する Amazon MWAA 環境を示しています。



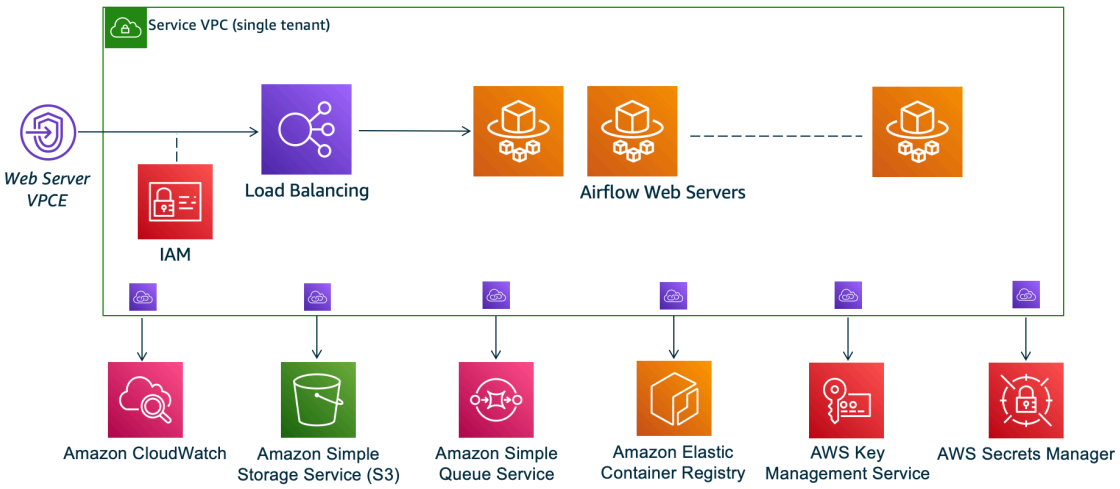
パブリックネットワークアクセスモードでは、「環境の IAM ポリシー」へのアクセスを許可されたユーザーは、インターネット経由で Apache Airflow UI にアクセスできます。

次の画像は、Amazon MWAA コンソールの [パブリックネットワーク] オプションの場所を示しています。



プライベートネットワーク

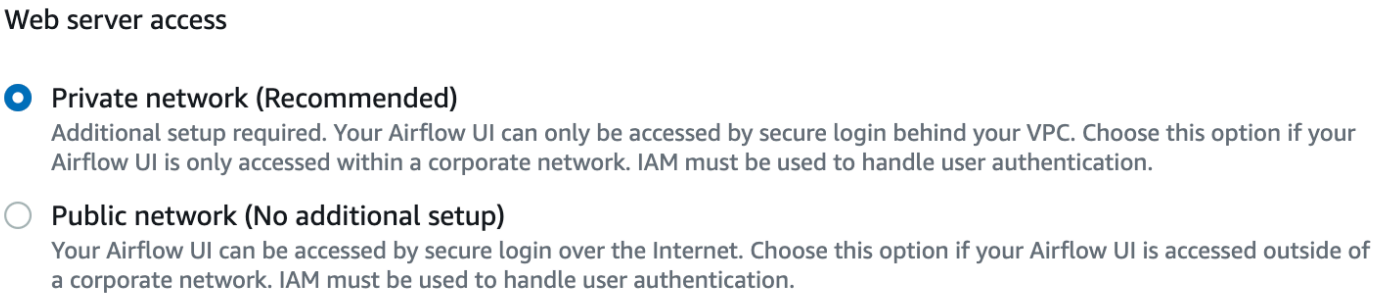
以下のアーキテクチャ図は、プライベートウェブサーバーを備えた Amazon MWAA 環境を示しています。



プライベートネットワークアクセスモードは、Apache Airflow UI へのアクセスを、「お使いの環境のIAM ポリシー」へのアクセスが許可されている Amazon VPC 内のユーザーに制限します。

プライベートなウェブサーバーアクセスを持つ環境を作成する場合、すべての依存関係を Python Wheel アーカイブ (`.whl`) にパッケージ化し、その後、`requirements.txt` で `.whl` を参照する必要があります。wheel を使用して依存関係をパッケージ化およびインストールする手順については、「Python wheel を使用した依存関係の管理」を参照してください。。

以下の画像は、Amazon MWAA コンソールの [プライベートネットワーク] オプションの場所を示しています。



アクセスモードの概要

このセクションでは、[パブリックネットワーク] または [プライベートネットワーク] のアクセスモードを選択したときに Amazon VPC に作成される VPC エンドポイント (AWS PrivateLink) について説明します。

パブリックネットワークアクセスモード

Apache Airflow ウェブサーバーでパブリックネットワークアクセスモードを選択した場合、ネットワークトラフィックはインターネット経由でパブリックにルーティングされます。

- Amazon MWAA は Amazon Aurora PostgreSQL メタデータデータベース用の VPC インターフェイスエンドポイントを作成します。エンドポイントは、プライベートサブネットにマッピングされたアベイラビリティーゾーンに作成され、他の AWS アカウントから独立しています。
- その後、Amazon MWAA はプライベートサブネットの IP アドレスをインターフェイスエンドポイントにバインドします。これは、Amazon VPC の各アベイラビリティーゾーンから単一の IP をバインドするというベストプラクティスをサポートするように設計されています。

プライベートネットワークアクセスモード

Apache Airflow ウェブサーバーでプライベートネットワークアクセスモードを選択した場合、ネットワークトラフィックは Amazon VPC 内でプライベートにルーティングされます。

- Amazon MWAA は、Apache Airflow ウェブサーバー用の VPC インターフェイスエンドポイントと、Amazon Aurora PostgreSQL メタデータデータベース用のインターフェイスエンドポイントを作成します。エンドポイントは、プライベートサブネットにマッピングされたアベイラビリティーゾーンに作成され、他の AWS アカウントから独立しています。
- その後、Amazon MWAA はプライベートサブネットの IP アドレスをインターフェイスエンドポイントにバインドします。これは、Amazon VPC の各アベイラビリティーゾーンから単一の IP をバインドするというベストプラクティスをサポートするように設計されています。

詳細については、「the section called “Amazon VPC と Apache エアフローアクセスモードのユースケース例”」を参照してください。

プライベートアクセスモードとパブリックアクセスモードのセットアップ

次のセクションでは、環境に合わせて選択した Apache Airflow アクセスモードに基づいて必要となる追加のセットアップと構成について説明します。

パブリックネットワークのセットアップ

Apache Airflow ウェブサーバーのパブリックネットワークオプションを選択した場合、環境を作成した後に Apache Airflow UI を使い始めることができます。

ユーザーのアクセスと、環境が他の AWS のサービスを使用するためのアクセス許可を設定するには、次のステップを実行する必要があります。

1. アクセス許可を追加します。Amazon MWAA には、他の AWS のサービスを使用するためのアクセス許可が必要です。環境を作成すると、Amazon MWAA は、Amazon Elastic Container Registry (Amazon ECR)、 CloudWatch Logs、および Amazon EC2 に対して特定の IAM アクションを使用できるようにするサービスにリンクされたロールを作成します。

   これらのサービスに追加のアクションを使用するアクセス許可を追加したり、実行ロールにアクセス許可を追加することで、他の AWS サービスを使用するアクセス許可を追加したりできます。詳細については、「Amazon MWAA 実行ロール」を参照してください。
2. ユーザーポリシーを作成します。環境と Apache Airflow UI へのアクセスを構成するために、ユーザー用に複数の IAM ポリシーを作成する必要がある場合があります。詳細については、「Amazon MWAA 環境へのアクセス」を参照してください。

プライベートネットワークのセットアップ

Apache Airflow ウェブサーバー のプライベートネットワークオプションを選択した場合は、ユーザーのアクセス、環境が他の AWS のサービスを使用するためのアクセス許可を設定し、コンピュータから Amazon VPC 内のリソースにアクセスするメカニズムを作成する必要があります。

1. アクセス許可を追加します。Amazon MWAA には、他の AWS のサービスを使用するためのアクセス許可が必要です。環境を作成すると、Amazon MWAA は、Amazon Elastic Container Registry (Amazon ECR)、 CloudWatch Logs、および Amazon EC2 に対して特定の IAM アクションを使用できるようにするサービスにリンクされたロールを作成します。

   これらのサービスに追加のアクションを使用するアクセス許可を追加したり、実行ロールにアクセス許可を追加することで、他の AWS サービスを使用するアクセス許可を追加したりできます。詳細については、「Amazon MWAA 実行ロール」を参照してください。
2. ユーザーポリシーを作成します。環境と Apache Airflow UI へのアクセスを構成するために、ユーザー用に複数の IAM ポリシーを作成する必要がある場合があります。詳細については、「Amazon MWAA 環境へのアクセス」を参照してください。
3. ネットワークアクセスを有効にします。Amazon VPC で、Apache Airflow ウェブサーバーの VPC エンドポイント (AWS PrivateLink) に接続するメカニズムを作成する必要があります。たとえば、 AWS Client VPNを使用して、コンピューターから VPN トンネルを作成します。

Apache Airflow ウェブサーバーの VPC エンドポイントへのアクセス (プライベートネットワークアクセス)

[プライベートネットワーク] オプションを選択した場合は、Amazon VPC に Apache Airflow ウェブサーバーの VPC エンドポイント (AWS PrivateLink) にアクセスするメカニズムを作成する必要があります。これらのリソースには、Amazon MWAA 環境と同じ Amazon MWAA、VPC セキュリティグループ、プライベートサブネットを使用することを推奨します。

詳細については、「VPC エンドポイントのアクセス管理」を参照してください。

Apache Airflow へのアクセス

Amazon MWAA では、Apache Airflow ユーザーインターフェイス (UI) コンソール、Apache Airflow CLI、および Apache Airflow REST API の複数の方法を使用して Apache Airflow 環境にアクセスできます。Amazon MWAA コンソールを使って Apache Airflow UI で DAG を表示して呼び出すか、Amazon MWAA APIを使ってトークンを取得して DAG を呼び出すことができます。このセクションでは、Apache Airflow UI にアクセスするために必要なパーミッション、コマンドシェルで直接 Amazon MWAA API コールを行うためのトークンの生成方法、Apache Airflow CLI でサポートされているコマンドについて説明します。

トピック

前提条件

次のセクションでは、このセクションのコマンドとスクリプトを使用するために必要な準備手順について説明します。

アクセス

- AWS の Amazon MWAA アクセス許可ポリシーへの AWS Identity and Access Management (IAM) の アカウントアクセスApache Airflow UI アクセスポリシー: AmazonMWAAWebServer アクセス。
- AWS Amazon MWAA アクセス許可ポリシー への AWS Identity and Access Management (IAM) での アカウントアクセスAPI とコンソールのフルアクセスポリシー: AmazonMWAAFullApi Access。

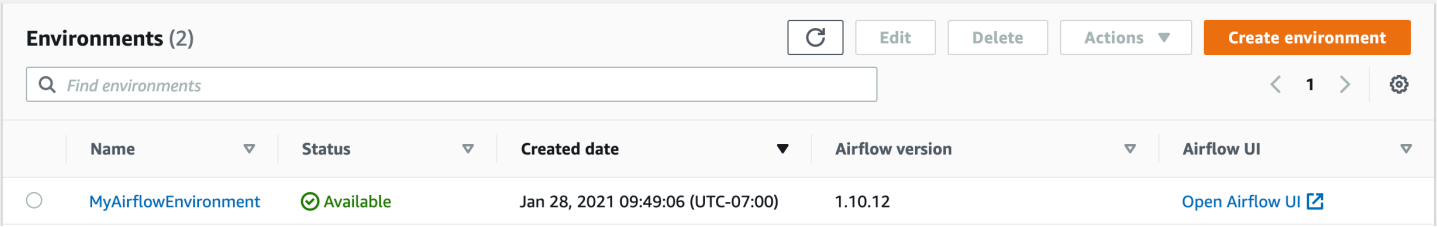
AWS CLI

AWS Command Line Interface （ AWS CLI) は、コマンドラインシェルのコマンドを使用して AWS サービスとやり取りできるようにするオープンソースツールです。このページのステップを完了するには、以下のものが必要です。

- AWS CLI – バージョン 2 をインストールします。
- AWS CLI – を使用したクイック設定aws configure。

Airflow UI を開きます

以下の図は、Amazon MWAA コンソールの Apache Airflow UI へのリンクを示しています。



Apache Airflow へのログイン

Apache Airflow UI を表示するには、 AWS Identity and Access Management (IAM) AWS のアカウントのApache Airflow UI アクセスポリシー: AmazonMWAAWebServer アクセスアクセス許可が必要です。

Apache Airflow UI にアクセスするには

1. Amazon MWAA コンソールで「環境ページ」を開きます。
2. 環境を選択します。
3. [Airflow UI を開く] を選択します。

Apache Airflow ウェブサーバーアクセストークンを作成する

このページのコマンドを使用して、ウェブサーバーアクセストークンを作成できます。アクセストークンを使用すると、Amazon MWAA 環境にアクセスできます。例えば、トークンを取得し、Amazon MWAA API を使用してプログラムで DAG をデプロイできます。次のセクションでは、、bash スクリプト AWS CLI、POST API リクエスト、または Python スクリプトを使用して Apache Airflow

ウェブログイントークンを作成する手順について説明します。レスポンスでリターンされるトークンは 60 秒間有効です。

目次

前提条件

以下のセクションでは、このページのコマンドとスクリプトを使用するために必要な準備手順について説明します。

アクセス

- AWS の Amazon MWAA アクセス許可ポリシーへの AWS Identity and Access Management (IAM) の アカウントアクセスApache Airflow UI アクセスポリシー: AmazonMWAAWebServer アクセス。
- AWS Amazon MWAA アクセス許可ポリシー への AWS Identity and Access Management (IAM) での アカウントアクセスAPI とコンソールのフルアクセスポリシー: AmazonMWAAFullApi Access。

AWS CLI

AWS Command Line Interface （ AWS CLI) は、コマンドラインシェルのコマンドを使用して AWS サービスとやり取りできるようにするオープンソースツールです。このページのステップを完了するには、以下のものが必要です。

- AWS CLI – バージョン 2 をインストールします。
- AWS CLI – を使用したクイック設定`aws configure`。

の使用 AWS CLI

次の例では、 の create-web-login-token コマンドを使用して Apache Airflow ウェブログイントークン AWS CLI を作成します。

```
aws mwaa create-web-login-token --name YOUR_ENVIRONMENT_NAME
```

Bash スクリプトを使用する

次の例では、bash スクリプトを使用して の create-web-login-token コマンドを呼び出し AWS CLI 、Apache Airflow ウェブログイントークンを作成します。

1. 以下のコードサンプルの内容をコピーし、ローカルに get-web-token.sh として保存します。

```
#!/bin/bash
HOST=YOUR_HOST_NAME
YOUR_URL=https://$HOST/aws_mwaa/aws-console-sso?login=true#
WEB_TOKEN=$(aws mwaa create-web-login-token --name YOUR_ENVIRONMENT_NAME --query
 WebToken --output text)
echo $YOUR_URL$WEB_TOKEN
```

2. ##のプレースホルダーを YOUR_HOST_NAME と YOUR_ENVIRONMENT_NAME に置き換えます。たとえば、パブリックネットワークのホスト名は次のようになります (https://を除く)。

```
123456a0-0101-2020-9e11-1b159eec9000.c2.us-east-1.airflow.amazonaws.com
```

3. (オプション) macOS と Linux ユーザーは、次のコマンドを実行して、スクリプトが実行可能であることを確認しなければならないことがあります。

```
chmod +x get-web-token.sh
```

4. 次のスクリプトを実行して、Web へのログイントークンを作成します。

```
./get-web-token.sh
```

5. コマンドプロンプトに次のように表示されるはずです。

```
https://123456a0-0101-2020-9e11-1b159eec9000.c2.us-east-1.airflow.amazonaws.com/
aws_mwaa/aws-console-sso?login=true#{your-web-login-token}
```

POST API リクエストを使用します。

次の例では、POST API リクエストを使用して Apache Airflow ウェブログイントークンを作成します。

1. 次の URL をコピーして、REST API クライアントの URL フィールドに貼り付けます。

```
https://YOUR_HOST_NAME/aws_mwaa/aws-console-sso?login=true#WebToken
```

2. ##のプレースホルダーを代わりに YOUR_HOST_NAME を使用してください。たとえば、パブリックネットワークのホスト名は次のようになります (https://を除く)。

```
123456a0-0101-2020-9e11-1b159eec9000.c2.us-east-1.airflow.amazonaws.com
```

3. 次の JSON をコピーして、REST API クライアントの本文フィールドに貼り付けます。

```
{
  "name": "YOUR_ENVIRONMENT_NAME"
}
```

4. ##のプレースホルダーを YOUR_ENVIRONMENT_NAME の代わりに使用してください。
5. 認証フィールドにキーと値のペアを追加します。たとえば、Postman を使用している場合は、[AWS 署名] を選択し、以下を入力します。
   - AccessKey の AWS_ACCESS_KEY_ID
   - SecretKey の AWS_SECRET_ACCESS_KEY
6. 以下のようなレスポンスが表示されます。

```
{
  "webToken": "<Short-lived token generated for enabling access to the Apache
 Airflow Webserver UI>",
  "webServerHostname": "<Hostname for the WebServer of the environment>"
}
```

Python スクリプトを使用します

以下の例では、Python スクリプトの「boto3 create_web_login_token」メソッドを使用して、Apache Airflow の Web ログイン・トークンを作成しています。このスクリプトは、Amazon

MWAA の外部で実行できます。必要なことは、boto3 ライブラリをインストールすることだけです。ライブラリをインストールするための仮想環境を作成することもできます。アカウントのAWS認証情報が設定されていることを前提としています。

1. 以下のコードサンプルの内容をコピーし、ローカルに create-web-login-token.py として保存します。

```
import boto3
mwaa = boto3.client('mwaa')
response = mwaa.create_web_login_token(
    Name="YOUR_ENVIRONMENT_NAME"
)
webServerHostName = response["WebServerHostname"]
webToken = response["WebToken"]
airflowUIUrl = 'https://{0}/aws_mwaa/aws-console-sso?
login=true#{1}'.format(webServerHostName, webToken)
print("Here is your Airflow UI URL: ")
print(airflowUIUrl)
```

2. ##のプレースホルダーを YOUR_ENVIRONMENT_NAME の代わりに使用してください。
3. 次のスクリプトを実行して、Web へのログイントークンを作成します。

```
python3 create-web-login-token.py
```

次のステップ

- でウェブログイントークンを作成するために使用される Amazon MWAA API オペレーションについて説明しますCreateWebLoginToken。

Apache Airflow CLI トークンの作成

このページのコマンドを使用して CLI トークンを生成し、Apache Airflow 用の Amazon マネージドワークフロー API 呼び出しをコマンドシェルで直接行うことができます。例えば、トークンを取得し、Amazon MWAA API を使用してプログラムで DAG をデプロイできます。次のセクションでは、AWS CLI、curl スクリプト、Python スクリプト、または bash スクリプトを使用して Apache Airflow CLI トークンを作成する手順について説明します。レスポンスでリターンされるトークンは 60 秒間有効です。

> **Note**
>
> この AWS CLI トークンは同期シェルアクションの代わりとなるもので、非同期 API コマンドの代わりとなるものではありません。そのため、同時実行には制限があります。ウェブサーバーがユーザーに応答し続けるためには、前の AWS CLI リクエストが正常に完了するまで新しいリクエストを開かないことが推奨されます。

目次

前提条件

以下のセクションでは、このページのコマンドとスクリプトを使用するために必要な準備手順について説明します。

アクセス

- Amazon MWAA アクセス権限ポリシー Apache Airflow UI アクセスポリシー: AmazonMWAAWebServer アクセス への AWS Identity and Access Management (IAM) の AWS アカウントアクセス。
- Amazon MWAA アクセス権限ポリシー API とコンソールのフルアクセスポリシー: AmazonMWAAFullApi Access への AWS Identity and Access Management (IAM) の AWS アカウントアクセス。

AWS CLI

AWS Command Line Interface (AWS CLI) は、コマンドラインシェルでコマンドを使用して AWS サービスとやり取りするためのオープンソースツールです。このページのステップを完了するには、以下のものが必要です。

- 「AWS CLI - バージョン 2 のインストール」
- 「AWS CLI - aws configure によるクイック設定」

AWS CLI を使用する場合

次の例では、AWS CLI 内のcreate-cli-token コマンドを使用して Apache Airflow CLI トークンを作成しています。

```
aws mwaa create-cli-token --name YOUR_ENVIRONMENT_NAME
```

curl スクリプトを使用する

次の例では、AWS CLI 内の curl スクリプトを使用して create-web-login-token コマンドを呼び出し、Apache Airflow ウェブサーバ上のエンドポイントを介して Apache Airflow CLI を呼び出しています。

Apache Airflow v2

1. テキストファイルから curl ステートメントをコピーし、コマンドシェルに貼り付けます。

   > **Note**
   >
   > クリップボードにコピーしたら、シェルメニューから [編集] > [貼り付け] を使用する必要がある場合があります。

   ```
   CLI_JSON=$(aws mwaa --region YOUR_REGION create-cli-token --
   name YOUR_ENVIRONMENT_NAME) \
     && CLI_TOKEN=$(echo $CLI_JSON | jq -r '.CliToken') \
     && WEB_SERVER_HOSTNAME=$(echo $CLI_JSON | jq -r '.WebServerHostname') \
     && CLI_RESULTS=$(curl --request POST "https://$WEB_SERVER_HOSTNAME/aws_mwaa/
   cli" \
     --header "Authorization: Bearer $CLI_TOKEN" \
   ```

```
  --header "Content-Type: text/plain" \
  --data-raw "dags trigger YOUR_DAG_NAME") \
  && echo "Output:" \
  && echo $CLI_RESULTS | jq -r '.stdout' | base64 --decode \
  && echo "Errors:" \
  && echo $CLI_RESULTS | jq -r '.stderr' | base64 --decode
```

2. プレースホルダー を、環境の AWS リージョンの YOUR_REGION、YOUR_DAG_NAME、および YOUR_ENVIRONMENT_NAME に置き換えます。たとえば、パブリックネットワークのホスト名は次のようになります (https://を除く)。

```
123456a0-0101-2020-9e11-1b159eec9000.c2.us-east-1.airflow.amazonaws.com
```

3. コマンドプロンプトに次のように表示されるはずです。

```
{
  "stderr":"<STDERR of the CLI execution (if any), base64 encoded>",
  "stdout":"<STDOUT of the CLI execution, base64 encoded>"
}
```

Apache Airflow v1

1. テキストファイルから cURL ステートメントをコピーし、コマンドシェルに貼り付けます。

> **Note**
>
> クリップボードにコピーしたら、シェルメニューから [編集] > [貼り付け] を使用する必要がある場合があります。

```
CLI_JSON=$(aws mwaa --region YOUR_REGION create-cli-token --
name YOUR_ENVIRONMENT_NAME) \
  && CLI_TOKEN=$(echo $CLI_JSON | jq -r '.CliToken') \
  && WEB_SERVER_HOSTNAME=$(echo $CLI_JSON | jq -r '.WebServerHostname') \
  && CLI_RESULTS=$(curl --request POST "https://$WEB_SERVER_HOSTNAME/aws_mwaa/
cli" \
  --header "Authorization: Bearer $CLI_TOKEN" \
  --header "Content-Type: text/plain" \
  --data-raw "trigger_dag YOUR_DAG_NAME") \
  && echo "Output:" \
```

```
  && echo $CLI_RESULTS | jq -r '.stdout' | base64 --decode \
  && echo "Errors:" \
  && echo $CLI_RESULTS | jq -r '.stderr' | base64 --decode
```

2. プレースホルダー を、環境の AWS リージョンの YOUR_REGION、YOUR_DAG_NAME、および YOUR_HOST_NAME に置き換えます。たとえば、パブリックネットワークのホスト名は次のようになります (https://を除く)。

```
123456a0-0101-2020-9e11-1b159eec9000.c2.us-east-1.airflow.amazonaws.com
```

3. コマンドプロンプトに次のように表示されるはずです。

```
{
  "stderr":"<STDERR of the CLI execution (if any), base64 encoded>",
  "stdout":"<STDOUT of the CLI execution, base64 encoded>"
}
```

4. プレースホルダーを YOUR_ENVIRONMENT_NAME と YOUR_DAG_NAME に置き換えます。

Bash スクリプトを使用する

次の例では、bash スクリプトを使用して AWS CLI 内のcreate-cli-token コマンドを呼び出し、Apache Airflow CLI トークンを作成しています。

Apache Airflow v2

1. 以下のコードサンプルの内容をコピーし、ローカルに get-cli-token.sh として保存します。

```
# brew install jq
  aws mwaa create-cli-token --name YOUR_ENVIRONMENT_NAME | export CLI_TOKEN=$(jq
 -r .CliToken) && curl --request POST "https://YOUR_HOST_NAME/aws_mwaa/cli" \
      --header "Authorization: Bearer $CLI_TOKEN" \
      --header "Content-Type: text/plain" \
      --data-raw "dags trigger YOUR_DAG_NAME"
```

2. ##のプレースホルダーを、YOUR_ENVIRONMENT_NAME、YOUR_HOST_NAME および YOUR_DAG_NAME に置き換えます。たとえば、パブリックネットワークのホスト名は次のようになります (https://を除く)。

```
123456a0-0101-2020-9e11-1b159eec9000.c2.us-east-1.airflow.amazonaws.com
```

3. (オプション) macOS と Linux ユーザーは、次のコマンドを実行して、スクリプトが実行可能であることを確認しなければならないことがあります。

```
chmod +x get-cli-token.sh
```

4. 次のスクリプトを実行して、Apache Airflow CLI トークンを作成します。

```
./get-cli-token.sh
```

Apache Airflow v1

1. 以下のコードサンプルの内容をコピーし、ローカルに get-cli-token.sh として保存します。

```
# brew install jq
  aws mwaa create-cli-token --name YOUR_ENVIRONMENT_NAME | export CLI_TOKEN=$(jq
 -r .CliToken) && curl --request POST "https://YOUR_HOST_NAME/aws_mwaa/cli" \
      --header "Authorization: Bearer $CLI_TOKEN" \
      --header "Content-Type: text/plain" \
      --data-raw "trigger_dag YOUR_DAG_NAME"
```

2. ##のプレースホルダーを、YOUR_ENVIRONMENT_NAME、YOUR_HOST_NAME および YOUR_DAG_NAME に置き換えます。たとえば、パブリックネットワークのホスト名は次のようになります (https://を除く)。

```
123456a0-0101-2020-9e11-1b159eec9000.c2.us-east-1.airflow.amazonaws.com
```

3. (オプション) macOS と Linux ユーザーは、次のコマンドを実行して、スクリプトが実行可能であることを確認しなければならないことがあります。

```
chmod +x get-cli-token.sh
```

4. 次のスクリプトを実行して、Apache Airflow CLI トークンを作成します。

```
./get-cli-token.sh
```

Python スクリプトを使用します

次の例では、Python スクリプトで boto3 create_cli_token メソッドを使用して Apache Airflow CLI トークンを作成し、DAG をトリガーします。このスクリプトは、Amazon MWAA の外部で実行できます。必要なことは、boto3 ライブラリをインストールすることだけです。ライブラリをインストールするための仮想環境を作成することもできます。アカウントに AWS 認証情報が設定されていることを前提としています。

Apache Airflow v2

1. 以下のコードサンプルの内容をコピーし、ローカルに create-cli-token.py として保存します。

```
"""
Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the "Software"), to deal in
the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of
the Software, and to permit persons to whom the Software is furnished to do so.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS
FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER
IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
"""
import boto3
import json
import requests
import base64

mwaa_env_name = 'YOUR_ENVIRONMENT_NAME'
dag_name = 'YOUR_DAG_NAME'
mwaa_cli_command = 'dags trigger'

client = boto3.client('mwaa')

mwaa_cli_token = client.create_cli_token(
    Name=mwaa_env_name
```

```
)

mwaa_auth_token = 'Bearer ' + mwaa_cli_token['CliToken']
mwaa_webserver_hostname = 'https://{0}/aws_mwaa/
cli'.format(mwaa_cli_token['WebServerHostname'])
raw_data = '{0} {1}'.format(mwaa_cli_command, dag_name)

mwaa_response = requests.post(
        mwaa_webserver_hostname,
        headers={
            'Authorization': mwaa_auth_token,
            'Content-Type': 'text/plain'
            },
        data=raw_data
        )

mwaa_std_err_message = base64.b64decode(mwaa_response.json()
['stderr']).decode('utf8')
mwaa_std_out_message = base64.b64decode(mwaa_response.json()
['stdout']).decode('utf8')

print(mwaa_response.status_code)
print(mwaa_std_err_message)
print(mwaa_std_out_message)
```

2. プレースホルダーを YOUR_ENVIRONMENT_NAME と YOUR_DAG_NAME に置き換えます。
3. 次のスクリプトを実行して、Apache Airflow CLI トークンを作成します。

```
python3 create-cli-token.py
```

Apache Airflow v1

1. 以下のコードサンプルの内容をコピーし、ローカルに create-cli-token.py として保存します。

```
import boto3
import json
import requests
import base64

mwaa_env_name = 'YOUR_ENVIRONMENT_NAME'
```

```
dag_name = 'YOUR_DAG_NAME'
mwaa_cli_command = 'trigger_dag'

client = boto3.client('mwaa')

mwaa_cli_token = client.create_cli_token(
    Name=mwaa_env_name
)

mwaa_auth_token = 'Bearer ' + mwaa_cli_token['CliToken']
mwaa_webserver_hostname = 'https://{0}/aws_mwaa/
cli'.format(mwaa_cli_token['WebServerHostname'])
raw_data = '{0} {1}'.format(mwaa_cli_command, dag_name)

mwaa_response = requests.post(
        mwaa_webserver_hostname,
        headers={
            'Authorization': mwaa_auth_token,
            'Content-Type': 'text/plain'
            },
        data=raw_data
        )

mwaa_std_err_message = base64.b64decode(mwaa_response.json()
['stderr']).decode('utf8')
mwaa_std_out_message = base64.b64decode(mwaa_response.json()
['stdout']).decode('utf8')

print(mwaa_response.status_code)
print(mwaa_std_err_message)
print(mwaa_std_out_message)
```

2. プレースホルダーを YOUR_ENVIRONMENT_NAME と YOUR_DAG_NAME に置き換えます。
3. 次のスクリプトを実行して、Apache Airflow CLI トークンを作成します。

```
python3 create-cli-token.py
```

次のステップ

- CreateClitoken で CLI トークンの作成に使用された Amazon MWAA API オペレーションをご覧ください。

Apache Airflow REST API の使用

Amazon Managed Workflows for Apache Airflow (Amazon MWAA) は、Apache Airflow v2.4.3 以降を実行している環境用の Apache Airflow REST API を使用した Apache Airflow 環境との直接対話をサポートしています。これにより、Amazon MWAA 環境にプログラムでアクセスおよび管理し、データオーケストレーションワークフローの呼び出し、DAGs の管理、メタデータデータベース、トリガー、スケジューラなどのさまざまな Apache Airflow コンポーネントのステータスのモニタリングを標準化できます。

Apache Airflow REST API を直接使用できるように、Amazon MWAA では、REST API リクエスト、コマンドラインインターフェイス (CLI) の使用、または Apache Airflow ユーザーインターフェイス (UI) の同時ユーザーなど、需要の増加に対応するためにウェブサーバー容量を水平方向にスケーリングするオプションが用意されています。Amazon MWAA がウェブサーバーをスケーリングする方法の詳細については、「」を参照してくださいthe section called “ウェブサーバーのオートスケーリングの設定”。

Apache Airflow REST API を使用して、環境に次のユースケースを実装できます。

- プログラムによるアクセス – Apache Airflow UI や CLI を使用せずに、Apache Airflow DAG の実行を開始したり、データセットを管理したり、メタデータデータベース、トリガー、スケジューラなどのさまざまなコンポーネントのステータスを取得したりできるようになりました。
- 外部アプリケーションやマイクロサービスとの統合 – REST API サポートにより、Amazon MWAA 環境を他のシステムと統合するカスタムソリューションを構築できます。例えば、完了したデータベースジョブや新しいユーザーのサインアップなど、外部システムからのイベントに応じてワークフローを開始できます。
- 一元化されたモニタリング – 複数の Amazon MWAA 環境にわたる DAGs のステータスを集約するモニタリングダッシュボードを構築して、一元化されたモニタリングと管理を可能にします。

以下のトピックでは、ウェブサーバーアクセストークンを取得し、そのトークンを使用して Apache Airflow REST API に API コールを行う方法を示します。次の例では、 API を呼び出して新しい DAG 実行を開始します。

Apache Airflow REST API の詳細については、「Apache Airflow REST API リファレンス」を参照してください。

トピック

- ウェブサーバーセッショントークンを作成する

- Apache Airflow REST API を呼び出す

ウェブサーバーセッショントークンを作成する

ウェブサーバーアクセストークンを作成するには、次の Python 関数を使用します。この関数は、まず Amazon MWAA API を呼び出してウェブログイントークンを取得します。60 秒後に期限切れになるウェブログイントークンは、ウェブセッショントークンと交換されます。これにより、ウェブサーバーにアクセスして Apache Airflow REST API を使用できます。

> **Note**
>
> セッショントークンは 12 時間後に期限切れになります。

```
def get_session_info(region, env_name):
    logging.basicConfig(level=logging.INFO)

    try:
        # Initialize MWAA client and request a web login token
        mwaa = boto3.client('mwaa', region_name=region)
        response = mwaa.create_web_login_token(Name=env_name)

        # Extract the web server hostname and login token
        web_server_host_name = response["WebServerHostname"]
        web_token = response["WebToken"]

        # Construct the URL needed for authentication
        login_url = f"https://{web_server_host_name}/aws_mwaa/login"
        login_payload = {"token": web_token}

        # Make a POST request to the MWAA login url using the login payload
        response = requests.post(
            login_url,
            data=login_payload,
            timeout=10
        )

        # Check if login was succesfull
        if response.status_code == 200:

            # Return the hostname and the session cookie
```

```
            return (
                web_server_host_name,
                response.cookies["session"]
            )
        else:
            # Log an error
            logging.error("Failed to log in: HTTP %d", response.status_code)
            return None
    except requests.RequestException as e:
         # Log any exceptions raised during the request to the MWAA login endpoint
        logging.error("Request failed: %s", str(e))
        return None
    except Exception as e:
        # Log any other unexpected exceptions
        logging.error("An unexpected error occurred: %s", str(e))
        return None
```

Apache Airflow REST API を呼び出す

認証が完了すると、API エンドポイントへのリクエストの送信を開始するための認証情報が取得されます。以下の例では、エンドポイント を使用します/dags/*dag_id*/dag。

```
def trigger_dag(region, env_name, dag_name):
    """
    Triggers a DAG in a specified MWAA environment using the Airflow REST API.

    Args:
    region (str): AWS region where the MWAA environment is hosted.
    env_name (str): Name of the MWAA environment.
    dag_name (str): Name of the DAG to trigger.
    """

    logging.info(f"Attempting to trigger DAG {dag_name} in environment {env_name} at
 region {region}")

    # Retrieve the web server hostname and session cookie for authentication
    try:
        web_server_host_name, session_cookie = get_session_info(region, env_name)
        if not session_cookie:
            logging.error("Authentication failed, no session cookie retrieved.")
            return
    except Exception as e:
```

```
        logging.error(f"Error retrieving session info: {str(e)}")
        return

    # Prepare headers and payload for the request
    cookies = {"session": session_cookie}
    json_body = {"conf": {}}

    # Construct the URL for triggering the DAG
    url = f"https://{web_server_host_name}/api/v1/dags/{dag_id}/dagRuns"

    # Send the POST request to trigger the DAG
    try:
        response = requests.post(url, cookies=cookies, json=json_body)
        # Check the response status code to determine if the DAG was triggered
 successfully
        if response.status_code == 200:
            logging.info("DAG triggered successfully.")
        else:
            logging.error(f"Failed to trigger DAG: HTTP {response.status_code} -
 {response.text}")
    except requests.RequestException as e:
        logging.error(f"Request to trigger DAG failed: {str(e)}")

if __name__ == "__main__":
    logging.basicConfig(level=logging.INFO)

    # Check if the correct number of arguments is provided
    if len(sys.argv) != 4:
        logging.error("Incorrect usage. Proper format: python script_name.py {region}
 {env_name} {dag_name}")
        sys.exit(1)

    region = sys.argv[1]
    env_name = sys.argv[2]
    dag_name = sys.argv[3]

    # Trigger the DAG with the provided arguments
    trigger_dag(region, env_name, dag_name)
```

Apache Airflow CLI コマンドリファレンス

このページでは、Apache Airflow 用の Amazon マネージドワークフローでサポートされている Apache Airflow CLI コマンドとサポートされていない Apache Airflow CLI コマンドについて説明します。

目次

前提条件

以下のセクションでは、このページのコマンドとスクリプトを使用するために必要な準備手順について説明します。

アクセス

- AWS の Amazon MWAA アクセス許可ポリシーへの AWS Identity and Access Management (IAM) の アカウントアクセスApache Airflow UI アクセスポリシー: AmazonMWAAWebServer アクセス。
- AWS Amazon MWAA アクセス許可ポリシー への AWS Identity and Access Management (IAM) のアカウントアクセスAPI とコンソールのフルアクセスポリシー: AmazonMWAAFullApi Access。

AWS CLI

AWS Command Line Interface（ AWS CLI) は、コマンドラインシェルのコマンドを使用して AWS サービスとやり取りできるオープンソースツールです。このページのステップを完了するには、以下のものが必要です。

- AWS CLI – バージョン 2 をインストールします。
- AWS CLI – を使用したクイック設定`aws configure`。

v2 での変更点

- 新規:エアフロー CLI コマンド構造。Apache Airflow v2 CLI は、関連するコマンドがサブコマンドとしてグループ化されるように設定されています。つまり、Apache Airflow v2 にアップグレードする場合は、Apache Airflow v1 スクリプトを更新する必要があります。例えば、Apache Airflow v1 `unpause` は、Apache Airflow v2では `dags unpause` になりました。詳細については、Apache Airflow リファレンスガイドの「2 におけるエアフロー CLI の変更点」を参照してください。

サポート済みの CLI コマンド

以下のセクションでは、Amazon MWAA で使用できる Apache Airflow CLI コマンドの一覧を表示しています。

サポートされているコマンド

Apache Airflow v2

マイナーバージョン	Command
v2.0+	「チートシート」
v2.0+	「接続を追加する」
v2.0+	「接続を削除する」
「v2.2+ (注)」	「dags バックフィル」
v2.0+	「dags 削除」

マイナーバージョン	Command
「v2.2+ (注)」	「dags リスト」
v2.0+	「dags リスト-ジョブ」
バージョン 2.6 以降	dags list-import-errors
「v2.2+ (注)」	「dags リスト-ラン」
「v2.2+ (注)」	「dags 次回の実行」
v2.0+	「dags 一時停止」
v2.0+	「dags レポート」
バージョン 2.4 以降	「dags リシリアライズ」
v2.0+	「dags 表示」
v2.0+	「dags 状態」
v2.0+	「dags テスト」
v2.0+	「dags トリガー」
v2.0+	「dags 一時停止解除」
バージョン 2.4 以降	「DB クリーン」
v2.0+	「プロバイダーの行動」
v2.0+	「プロバイダー取得」
v2.0+	「プロバイダーフック」
v2.0+	「プロバイダーリンク」
v2.0+	「プロバイダーリスト」
v2.8 以降	プロバイダーの通知

マイナーバージョン	Command
バージョン 2.6 以降	「プロバイダーの秘密」
v2.7+	「プロバイダートリガー」
v2.0+	「プロバイダーウィジェット」
バージョン 2.6 以降	「ロールアドパーマ」
バージョン 2.6 以降	「ロールの役割」
バージョン 2.6 以降	ロールの作成
v2.0+	「ロールリスト」
v2.0+	「タスククリア」
v2.0+	「タスク失敗-DEPS」
v2.0+	「タスクリスト」
v2.0+	「タスクレンダー」
v2.0+	「タスク状態」
v2.0+	タスク states-for-dag-run
v2.0+	「タスクテスト」
v2.0+	「変数を削除する」
v2.0+	「変数を取得する」
v2.0+	「変数を取得する」
v2.0+	「変数リスト」
v2.0+	バージョン

DAGs を解析するコマンドを使用する

環境が Apache Airflow v1.10.12 または v2.0.2 を実行している場合、DAGs が を介してインストールされたパッケージに依存するプラグインを使用すると、DAG を解析する CLI コマンドは失敗しますrequirements.txt。

Apache Airflow v2.0.2

- dags backfill
- dags list
- dags list-runs
- dags next-execution

DAG が requirements.txt を介してインストールされたパッケージに依存するプラグインを使用していない場合は、次の CLI コマンドを使用できます。

「サンプルコード」

次のセクションには、Apache Airflow CLI を使用する各種方法の例が含まれています。

Apache Airflow v2 変数を設定、取得、または削除します。

次のサンプルコードを使用して、WWH 形式で変数を設定、取得、または削除できます<script> <mwaa env name> get | set | delete <variable> <variable value> </variable> </variable>。

```
[ $# -eq 0 ] && echo "Usage: $0 MWAA environment name " && exit

if [[ $2 == "" ]]; then
    dag="variables list"

elif  [ $2 == "get" ] ||  [ $2 == "delete" ] ||  [ $2 == "set" ]; then
    dag="variables $2 $3 $4 $5"

else
    echo "Not a valid command"
    exit 1
fi

CLI_JSON=$(aws mwaa --region $AWS_REGION create-cli-token --name $1) \
```

```
    && CLI_TOKEN=$(echo $CLI_JSON | jq -r '.CliToken') \
    && WEB_SERVER_HOSTNAME=$(echo $CLI_JSON | jq -r '.WebServerHostname') \
    && CLI_RESULTS=$(curl --request POST "https://$WEB_SERVER_HOSTNAME/aws_mwaa/cli" \
    --header "Authorization: Bearer $CLI_TOKEN" \
    --header "Content-Type: text/plain" \
    --data-raw "$dag" ) \
    && echo "Output:" \
    && echo $CLI_RESULTS | jq -r '.stdout' | base64 --decode \
    && echo "Errors:" \
    && echo $CLI_RESULTS | jq -r '.stderr' | base64 --decode
```

DAG をトリガーするときに設定を追加します。

Apache Airflow v1 および Apache Airflow v2 で次のサンプル コードを使用すると、DAG がトリガーされたときに `airflow trigger_dag 'dag_name' –conf '{"key":"value"}'` などの構成を追加できます。

```
import boto3
import json
import requests
import base64

mwaa_env_name = 'YOUR_ENVIRONMENT_NAME'
dag_name = 'YOUR_DAG_NAME'
key = "YOUR_KEY"
value = "YOUR_VALUE"
conf = "{\"" + key + "\":\"" + value + "\"}"

client = boto3.client('mwaa')

mwaa_cli_token = client.create_cli_token(
  Name=mwaa_env_name
)

mwaa_auth_token = 'Bearer ' + mwaa_cli_token['CliToken']
mwaa_webserver_hostname = 'https://{0}/aws_mwaa/
cli'.format(mwaa_cli_token['WebServerHostname'])
raw_data = "trigger_dag {0} -c '{1}'".format(dag_name, conf)

mwaa_response = requests.post(
      mwaa_webserver_hostname,
      headers={
          'Authorization': mwaa_auth_token,
```

```
          'Content-Type': 'text/plain'
          },
      data=raw_data
      )

mwaa_std_err_message = base64.b64decode(mwaa_response.json()['stderr']).decode('utf8')
mwaa_std_out_message = base64.b64decode(mwaa_response.json()['stdout']).decode('utf8')

print(mwaa_response.status_code)
print(mwaa_std_err_message)
print(mwaa_std_out_message)
```

拠点ホストへの SSH 踏み台ホストで CLI コマンドを実行します

次の例は、Linux 踏み台ホストトへの SSH トンネルプロキシを使用して Airflow CLI コマンドを実行する方法を示しています。

curl を使用

1. ```
   ssh -D 8080 -f -C -q -N YOUR_USER@YOUR_BASTION_HOST
   ```
2. ```
   curl -x socks5h://0:8080 --request POST https://YOUR_HOST_NAME/aws_mwaa/cli --
   header YOUR_HEADERS --data-raw YOUR_CLI_COMMAND
   ```

GitHub および AWS チュートリアルのサンプル

- 「Apache Airflow の Amazon マネージドワークフローでの Apache Airflow v2.0.2 パラメータと変数の使用」
- 「Amazon MWAA の Apache Airflow v1.10.12 をコマンドラインから操作する」
- での Amazon MWAA と Bash Operator での Apache Airflow v1.10.12 を使用したインタラクティブコマンド GitHub

Apache エアフローへの接続の管理

このセクションでは、Amazon Managed Workflows for Apache Airflow 環境の Apache Airflow 接続を構成するさまざまな方法について説明します。

トピック

Apache エアフロー変数と接続の概要

場合によっては、環境に対して追加の接続や変数を指定したいことがあります。例えば、AWSプロファイルを追加したり、Apache Airflowメタストア内の接続オブジェクトに実行ロールを追加したりして、それをDAG内から参照することがあります。

- セルフマネージド Apache Airflow 自己管理型のApache Airflowインストールでは、Apache Airflow 構成オプションは`airflow.cfg`で設定します。

```
[secrets]
backend = airflow.providers.amazon.aws.secrets.secrets_manager.SecretsManagerBackend
backend_kwargs = {"connections_prefix" : "airflow/connections", "variables_prefix" :
 "airflow/variables"}
```

- Amazon MWAA の Apache Airflow。Amazon MWAA では、これらの設定を Amazon MWAA コンソールの Apache Airflow 構成オプション として追加する必要があります。Apache Airflow 構成オプションは環境変数として記述され、同じ設定の他のすべての既存設定よりも優先されます。

Amazon MWAA 環境にインストールされている Apache エアフロープロバイダーパッケージ

Amazon MWAA では、新しい環境を作成すると、Apache Airflow v2 以降の接続タイプ用のプロバイダーエクストラがインストールされます。プロバイダーパッケージをインストールすると、Apache Airflow UI に接続タイプが表示されます。また、これらのパッケージを`requirements.txt`ファイ

ル内の Python 依存関係として指定する必要がないことも意味します。このページには、Amazon MWAA によってすべての Apache Airflow v2 環境にインストールされた Apache Airflow プロバイダーパッケージが一覧表示されます。

> **Note**
>
> Apache Airflow v2 以降では、Amazon MWAA は の実行後に Watchtower バージョン 2.0.1 をインストールしpip3 install -r requirements.txt、 CloudWatch ログ記録との互換性が他の Python ライブラリのインストールによって上書きされないようにします。

目次

Apache Airflow v2.8.1 接続のプロバイダーパッケージ

Apache Airflow v2.8.1 で Amazon MWAA 環境を作成すると、Amazon MWAA は Apache Airflow 接続に使用される以下のプロバイダーパッケージをインストールします。

> **Note**
>
> このプロバイダをアップグレードするには、サポートされている最新バージョンのapache-airflow-providers-amazonを指定できます。新しいバージョンの指定に関する詳細は、「the section called “新しいプロバイダーパッケージの指定”」を参照してください。

接続タイプ	パッケージ
AWS 接続	apache-airflow-providers-amazon[aiobotocore]==8.16.0
Postgres 接続	apache-airflow-providers-postgres==5.10.0
FTP 接続	apache-airflow-providers-ftp==3.7.0
セロリ接続	apache-airflow-providers-celery==3.5.1
HTTP 接続	apache-airflow-providers-http==4.8.0
IMAP 接続	apache-airflow-providers-imap==3.5.0
共通 SQL	apache-airflow-providers-common-sql==1.10.0
SQLite コネクション	apache-airflow-providers-sqlite==3.7.0

Apache Airflow v2.7.2 接続用のプロバイダーパッケージ

Apache Airflow v2.7.2 で Amazon MWAA 環境を作成すると、Amazon MWAA は Apache Airflow 接続に使用される以下のプロバイダーパッケージをインストールします。

> **Note**
>
> このプロバイダをアップグレードするには、サポートされている最新バージョンのapache-airflow-providers-amazonを指定できます。新しいバージョンの指定に関する詳細は、「the section called “新しいプロバイダーパッケージの指定”」を参照してください。

接続タイプ	パッケージ
AWS 接続	apache-airflow-providers-amazon[aiobotocore]==8.7.1
Postgres 接続	apache-airflow-providers-postgres==5.6.1

接続タイプ	パッケージ
FTP 接続	apache-airflow-providers-ftp==3.5.2
セロリ接続	apache-airflow-providers-celery==3.3.4
HTTP 接続	apache-airflow-providers-http==4.5.2
IMAP 接続	apache-airflow-providers-imap==3.3.2
共通 SQL	apache-airflow-providers-common-sql==1.7.2
SQLite コネクション	apache-airflow-providers-sqlite==3.4.3

Apache Airflow v2.6.3 接続用のプロバイダーパッケージ

Apache Airflow v2.6.3 で Amazon MWAA 環境を作成すると、Amazon MWAA は Apache Airflow 接続に使用される以下のプロバイダーパッケージをインストールします。

> **Note**
>
> このプロバイダをアップグレードするには、サポートされている最新バージョンのapache-airflow-providers-amazonを指定できます。新しいバージョンの指定に関する詳細は、「the section called “新しいプロバイダーパッケージの指定”」を参照してください。

接続タイプ	パッケージ
AWS 接続	apache-airflow-providers-amazon[aiobotocore]=8.2.0
Postgres 接続	apache-airflow-providers-postgres==5.5.1
FTP 接続	apache-airflow-providers-ftp==3.4.2
セロリ接続	apache-airflow-providers-celery==3.2.1
HTTP 接続	apache-airflow-providers-http==4.4.2

接続タイプ	パッケージ
IMAP 接続	apache-airflow-providers-imap==3.2.2
共通 SQL	apache-airflow-providers-common-sql==1.5.2
SQLite コネクション	apache-airflow-providers-sqlite==3.4.2

Apache Airflow v2.5.1 接続用のプロバイダーパッケージ

Apache Airflow v2.5.1 で Amazon MWAA 環境を作成すると、Amazon MWAA は Apache Airflow 接続に使用される以下のプロバイダーパッケージをインストールします。

> **Note**
>
> このプロバイダをアップグレードするには、サポートされている最新バージョンのapache-airflow-providers-amazonを指定できます。新しいバージョンの指定に関する詳細は、「the section called “新しいプロバイダーパッケージの指定”」を参照してください。

接続タイプ	パッケージ
AWS 接続	apache-airflow-providers-amazon==7.1.0
Postgres 接続	apache-airflow-providers-postgres==5.4.0
FTP 接続	apache-airflow-providers-ftp==3.3.0
セロリ接続	apache-airflow-providers-celery==3.1.0
HTTP 接続	apache-airflow-providers-http==4.1.1
IMAP 接続	apache-airflow-providers-imap==3.1.1
共通 SQL	apache-airflow-providers-common-sql==1.3.3
SQLite コネクション	apache-airflow-providers-sqlite==3.3.1

Apache Airflow v2.4.3 接続用のプロバイダーパッケージ

Apache Airflow v2.4.3でAmazon MWAA環境を作成すると、Amazon MWAAはApache Airflow接続に使用される以下のプロバイダーパッケージをインストールします。

接続タイプ	パッケージ
AWS 接続	apache-airflow-providers-amazon==6.0.0
Postgres 接続	apache-airflow-providers-postgres==5.2.2
FTP 接続	apache-airflow-providers-ftp==3.1.0
セロリ接続	apache-airflow-providers-celery==3.0.0
HTTP 接続	apache-airflow-providers-http==4.0.0
IMAP 接続	apache-airflow-providers-imap==3.0.0
共通 SQL	apache-airflow-providers-common-sql==1.2.0
SQLite コネクション	apache-airflow-providers-sqlite==3.2.1

Apache Airflow v2.2.2 接続用のプロバイダーパッケージ

Apache Airflow v2.2.2でAmazon MWAA環境を作成すると、Amazon MWAAはApache Airflow接続に使用される以下のプロバイダーパッケージをインストールします。

接続タイプ	パッケージ
AWS 接続	apache-airflow-providers-amazon==2.4.0
Postgres 接続	apache-airflow-providers-postgres==2.3.0
FTP 接続	apache-airflow-providers-ftp==2.0.1
セロリ接続	apache-airflow-providers-celery==2.1.0
HTTP 接続	apache-airflow-providers-http==2.0.1

接続タイプ	パッケージ
IMAP 接続	apache-airflow-providers-imap==2.0.1
SQLite コネクション	apache-airflow-providers-sqlite==2.0.1

Apache Airflow v2.0.2 接続用のプロバイダーパッケージ

Apache Airflow v2.0.2でAmazon MWAA環境を作成すると、Amazon MWAAはApache Airflow接続に使用される以下のプロバイダーパッケージをインストールします。

接続タイプ	パッケージ
Tableau Connection	apache-airflow-providers-tableau==1.0.0
データブリックス接続	apache-airflow-providers-databricks==1.0.1
SSH 接続	apache-airflow-providers-ssh==1.3.0
Postgres 接続	apache-airflow-providers-postgres==1.0.2
Docker Connection	apache-airflow-providers-docker==1.2.0
Oracle Connection	apache-airflow-providers-oracle==1.1.0
Presto Connection	apache-airflow-providers-presto==1.0.2
SFTP 接続	apache-airflow-providers-sftp==1.2.0

新しいプロバイダーパッケージの指定

Apache Airflow v2.7.2 から、要件ファイルには `--constraint` ステートメントを含める必要があります。制約を指定しない場合、要件に記載されているパッケージが使用している Apache Airflow のバージョンと互換性があることを確認するため、Amazon MWAA はお客様に代わって制約を指定します。

Apache Airflow の制約ファイルには、Apache Airflow のリリース時点で利用可能なプロバイダーのバージョンが指定されています。ただし、多くの場合、新しいプロバイダーはそのバージョンの

Apache Airflow と互換性があります。制約を使用する必要があるため、プロバイダ・パッケージの新しいバージョンを指定するには、特定のプロバイダ・バージョンに合わせて制約ファイルを変更できます。

1. バージョン固有の制約ファイルを https://raw.githubusercontent.com/apache/airflow/constraints-2.7.2/constraints-3.11.txt "からダウンロードしてください。
2. 制約ファイル内のapache-airflow-providers-amazonバージョンを、使用したいバージョンに変更します。
3. 変更した制約ファイルを、Amazon MWAA 環境の Amazon S3 DAGs フォルダに保存してください。例えば、constraints-3.11-updated.txtとして保存します。
4. 次に示すように、要件を指定します。

```
--constraint "/usr/local/airflow/dags/constraints-3.11-updated.txt"

apache-airflow-providers-amazon==version-number
```

> **Note**
>
> プライベートウェブサーバーを使用している場合は、Amazon MWAA local-runner を使用して必要なライブラリを WHL ファイルとしてパッケージ化することをお勧めします。

接続タイプの概要

Apache Airflow は接続を接続 URI 文字列として保存します。接続タイプに関係なく、接続 URI 文字列を生成するための接続テンプレートが Apache Airflow UI に提供されます。Apache Airflow UI で接続テンプレートが使用できない場合は、HTTP 接続テンプレートを使用するなど、代替の接続テンプレートを使用してこの接続 URI 文字列を生成できます。主な違いは、my-conn-type:// などのURIのプレフィックスにあります。通常、Apache Airflow プロバイダーは接続時にこれを無視します。このページでは、Apache Airflow UI の接続テンプレートを異なる接続タイプで互換的に使用する方法について説明します。

> **Warning**
>
> Amazon MWAA で `aws_default` の接続を上書きしないでください。Amazon MWAA はこの接続を使用して、タスクログの収集など、さまざまな重要なタスクを実行します。この接続を上書きすると、データが失われ、環境の可用性が中断される可能性があります。

トピック

接続 URI 文字列の例

次の例は、MySQL 接続タイプの接続 URI 文字列を示しています。

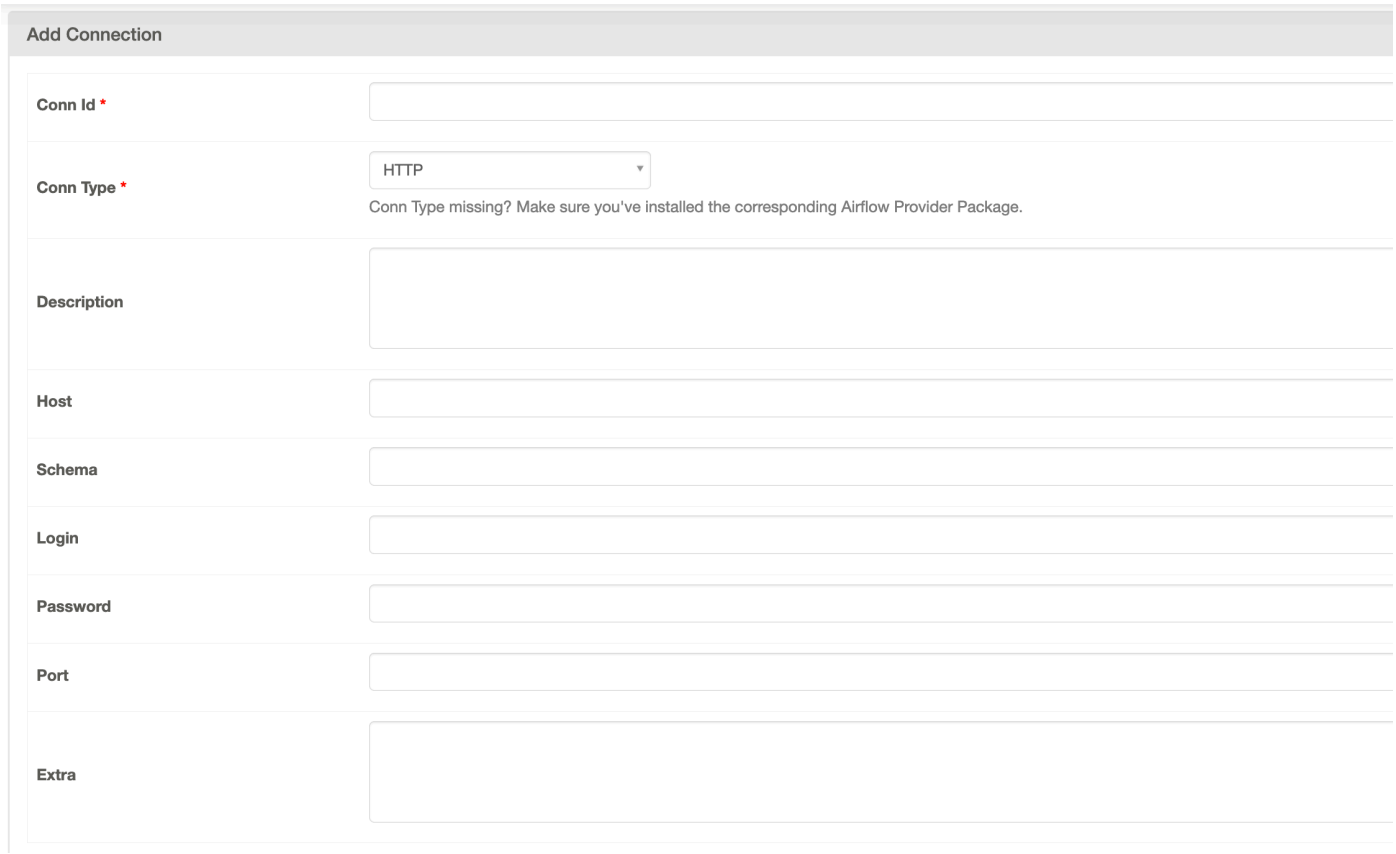
```
'mysql://288888a0-50a0-888-9a88-1a111aaa0000.a1.us-east-1.airflow.amazonaws.com
%2Fhome?role_arn=arn%3Aaws%3Aiam%3A%3A001122332255%3Arole%2Fservice-role%2FAmazonMWAA-
MyAirflowEnvironment-iAaaaA&region_name=us-east-1'
```

サンプル接続テンプレート

次の例は、Apache Airflow UI の HTTP 接続テンプレートを示しています。

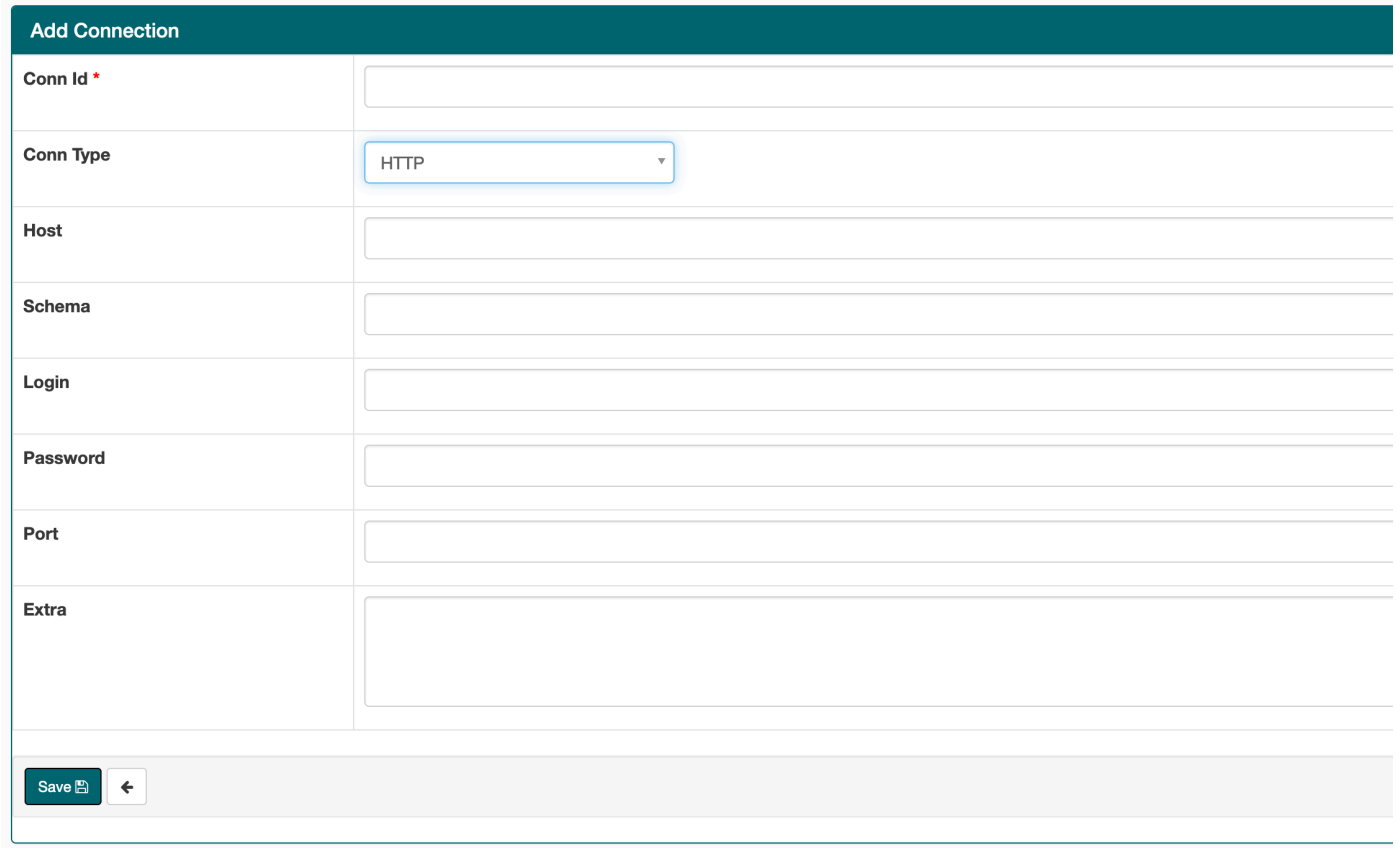
Apache Airflow v2

次の例は、Apache Airflow UI の Apache Airflow v2 の HTTP 接続テンプレートを示しています。



Apache Airflow v1

次の例は、Apache Airflow UI の Apache Airflow v1 の HTTP 接続テンプレートを示しています。



JDBC 接続に HTTP 接続テンプレートを使用する例

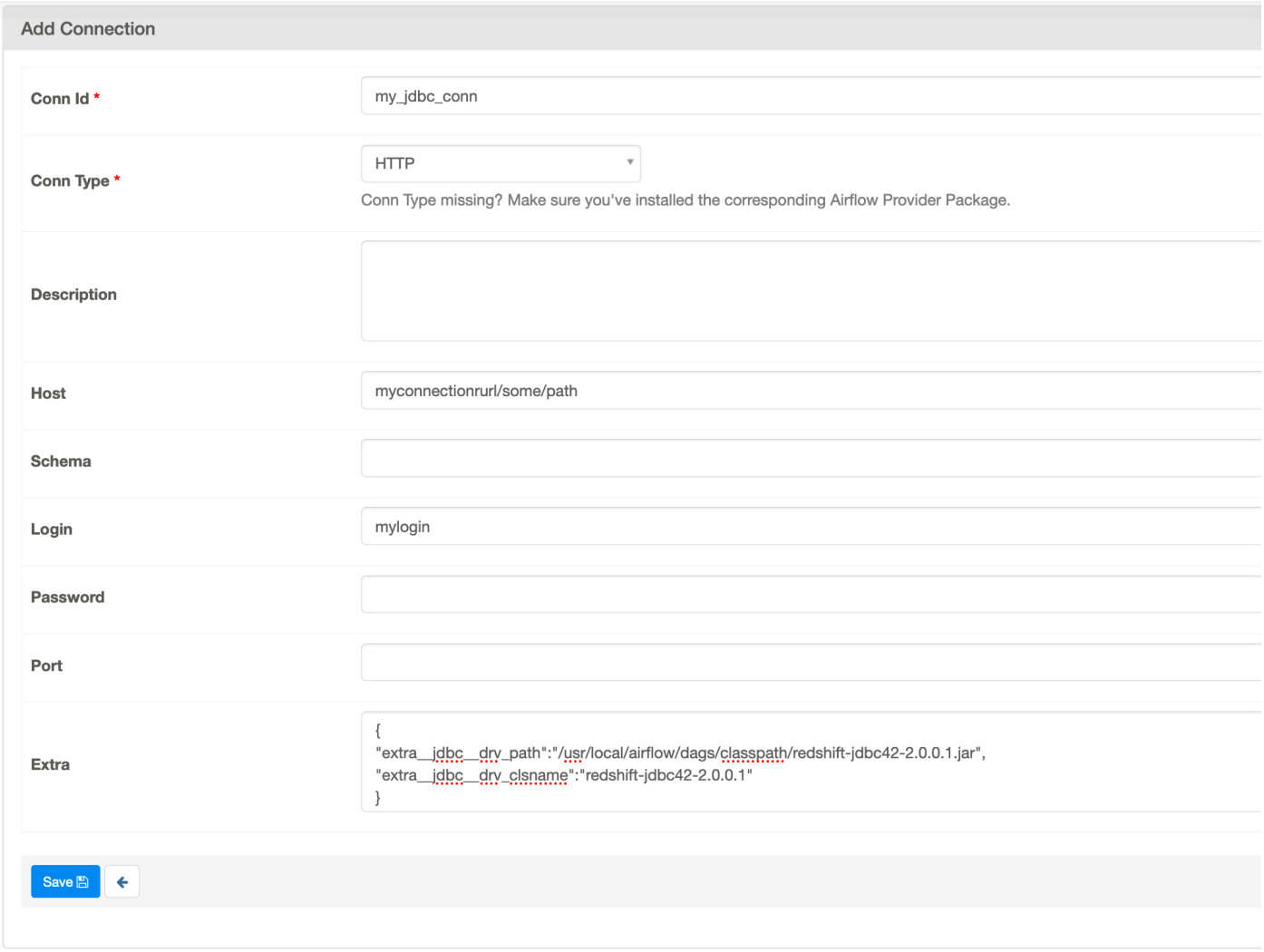
次の例は、Apache Airflow v2.0.2 の Jdbc 接続タイプの HTTP 接続テンプレートと、Apache Airflow UI の Apache Airflow v1.10.12 の Jdbc 接続テンプレートの同じ値を使用する方法を示しています。

Apache Airflow v2

次の例は、このセクションの例で Apache Airflow によって生成された接続 URI 文字列を示しています。

```
http://myconnectionurl/some/path&login=mylogin&extra__jdbc__dry__path=usr/local/
airflow/dags/classpath/redshif-
jdbc42-2.0.0.1.jar&extra__jdbc__dry__clsname=redshift-jdbc42-2.0.0.1
```

次の例は、Apache Airflow UI で Apache Airflow v2 の Jdbc 接続に HTTP 接続テンプレートを使用する方法を示しています。

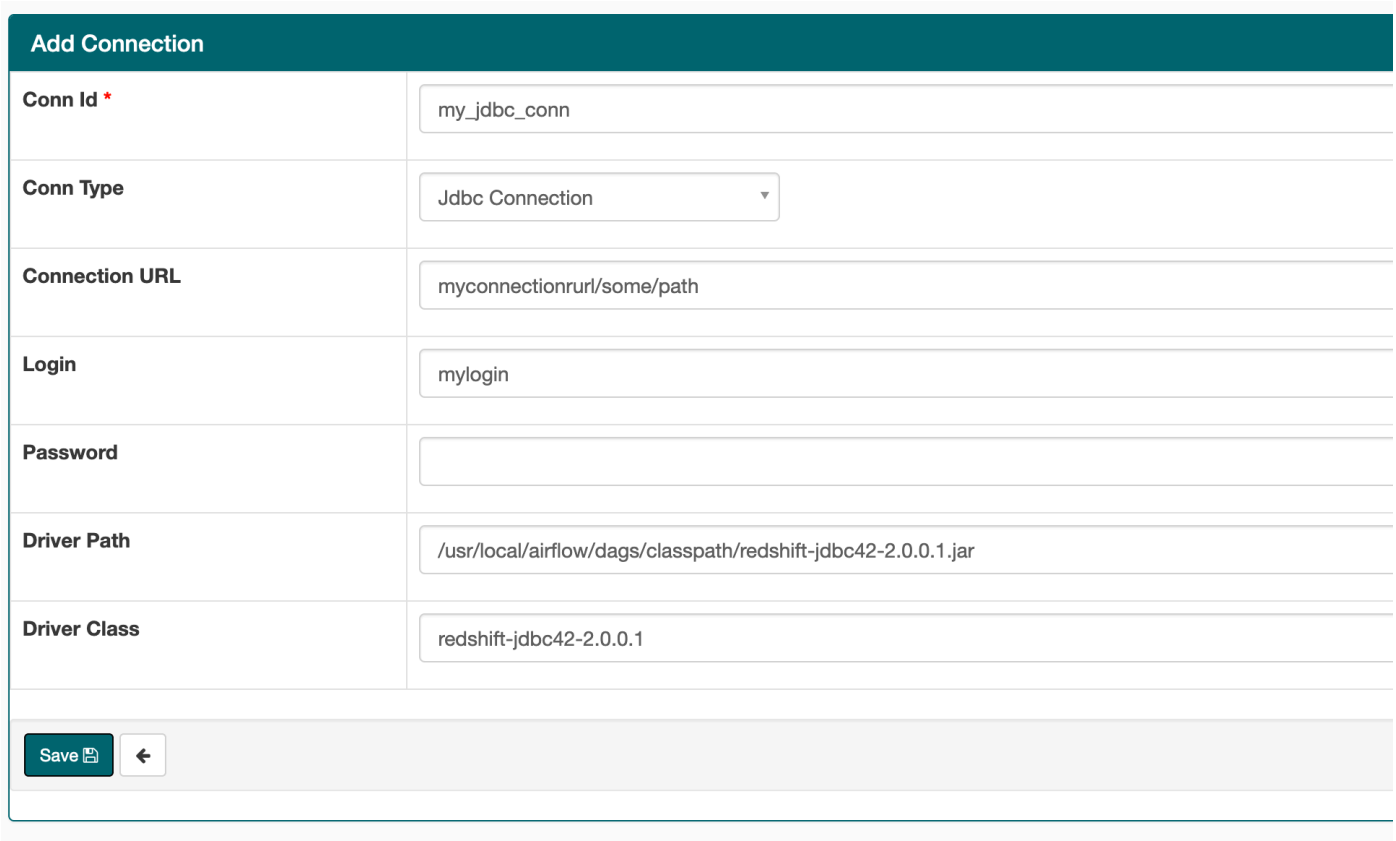


Apache Airflow v1

次の例は、このセクションの例で Apache Airflow によって生成された接続 URI 文字列を示しています。

```
jdbc://myconnectionurl/some/path&login=mylogin&extra__jdbc__dry__path=usr/local/
airflow/dags/classpath/redshif-
jdbc42-2.0.0.1.jar&extra__jdbc__dry__clsname=redshift-jdbc42-2.0.0.1
```

次の例は、Apache Airflow UI の Apache Airflow v1.10.12 用の Jdbc 接続テンプレートを示しています。



AWS Secrets Manager シークレットを使用した Apache Airflow 接続の設定

AWS Secrets Manager は、Amazon Managed Workflows for Apache Airflow 環境でサポートされている代替 Apache Airflow バックエンドです。このガイドでは、 AWS Secrets Manager を使用して Apache Airflow 変数のシークレットと Apache Airflow 接続を Amazon Managed Workflows for Apache Airflow に安全に保存する方法を示します。

> **Note**
>
> - 作成したシークレットには料金が発生します。Secrets Manager の価格については、「AWS 価格」を参照してください。

目次

ステップ 1: Amazon MWAA にSecrets Manager のシークレットキーにアクセスする権限を付与する

Amazon MWAA 環境の実行ロールには、 AWS Secrets Manager内のシークレットキーへの読み取りアクセス権が必要です。次の IAM ポリシーは、 AWS 管理SecretsManagerReadWriteポリシーを使用した読み取り/書き込みアクセスを許可します。

実行ロールにポリシーを添付するには

1. Amazon MWAA コンソールで「環境ページ」を開きます。
2. 環境を選択します。
3. [権限] ペインで実行ロールを選択します。
4. [Attach policies] (ポリシーの添付) を選択します。
5. [Filter policies] (フィルターポリシー) テキストフィールドに `SecretsManagerReadWrite` をタイプ。
6. Attach policy] (ポリシーのアタッチ) を選択します。

AWS 管理アクセス許可ポリシーを使用しない場合は、環境の実行ロールを直接更新して、Secrets Manager リソースへの任意のレベルのアクセスを許可できます。例えば、次のポリシーステートメントは、Secrets Manager の特定の AWS リージョンで作成したすべてのシークレットへの読み取りアクセスを許可します。

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

```
            "Action": [
                "secretsmanager:GetResourcePolicy",
                "secretsmanager:GetSecretValue",
                "secretsmanager:DescribeSecret",
                "secretsmanager:ListSecretVersionIds"
            ],
            "Resource": "arn:aws:secretsmanager:us-west-2:012345678910:secret:*"
        },
        {
            "Effect": "Allow",
            "Action": "secretsmanager:ListSecrets",
            "Resource": "*"
        }
    ]
}
```

ステップ 2: Secrets Manager のバックエンドを Apache Airflow 構成オプションとして作成する

次のセクションでは、 AWS Secrets Manager バックエンドの Amazon MWAA コンソールで Apache Airflow 設定オプションを作成する方法について説明します。`airflow.cfg`で同じ名前の設定を使用している場合は、次のステップで作成した設定が優先され、設定よりも優先されます。

1. Amazon MWAA コンソールで「環境ページ」を開きます。
2. 環境を選択します。
3. [編集] を選択します。
4. [次へ] をクリックします。
5. Airflow 設定オプションペインで [カスタム設定を追加] を選択します。以下のキーと値のペアを追加します。

    a. **`secrets.backend`**: **`airflow.providers.amazon.aws.secrets.secrets_manager.SecretsManagerBacken`**

    b. **`secrets.backend_kwargs`**: **`{"connections_prefix" : "airflow/connections", "variables_prefix" : "airflow/variables"}`** これにより、Apache Airflowは接続文字列と変数を`airflow/connections/*`および`airflow/variables/*`のパスで検索するように構成されます。

    ルックアップパターンを使用すると、Amazon MWAA がユーザーに代わって Secrets Manager に対して行う API 呼び出しの数を減らすことができます。検索パターンを指定し

ない場合、Apache Airflow は設定されたバックエンドのすべての接続と変数を検索します。パターンを指定することで、Apache Airflow が検索する可能性のあるパスを絞り込むことができます。これにより、Amazon MWAA でSecrets Manager を使用する場合のコストを削減できます。

ルックアップパターンを指定するには、connections_lookup_patternおよびvariables_lookup_patternパラメータを指定します。これらのパラメータは RegEx 文字列を入力として受け入れます。例えば、testで始まるシークレットを検索するには、secrets.backend_kwargsに次のように入力します：

```
{
  "connections_prefix": "airflow/connections",
  "connections_lookup_pattern": "^test",
  "variables_prefix" : "airflow/variables",
  "variables_lookup_pattern": "^test"
}
```

> **Note**
>
> connections_lookup_patternおよびvariables_lookup_patternを使用するには、apache-airflow-providers-amazonバージョン7.3.0以上をインストールする必要があります。プロバイダパッケージを新しいバージョンに更新する方法の詳細については、the section called “新しいプロバイダーパッケージの指定”を参照してください。

6. [保存] を選択します。

ステップ 3: Apache Airflow AWS 接続 URI 文字列を生成する

接続文字列を作成するには、キーボードの「タブ」キーを使用して、接続オブジェクトのキーと値のペアをインデントします。また、シェルセッションでextraオブジェクト用の変数を作成することをおすすめします。以下のセクションでは、Apache Airflow または Python スクリプトを使用して Amazon MWAA 環境用の Apache Airflow 接続 URI 文字列を生成する手順を順を追って説明します。

Apache Airflow CLI

次のシェルセッションでは、ローカル Airflow CLI を使用して接続文字列を生成します。CLI がインストールされていない場合は、Python スクリプトの使用をお勧めします。

1. Python シェルセッションを開きます。

```
python3
```

2. 次のコマンドを入力します。

```
>>> import json
```

3. 次のコマンドを入力します。

```
>>> from airflow.models.connection import Connection
```

4. シェルセッションにextraオブジェクト用の変数を作成します。*YOUR_EXECUTION_ROLE_ARN*のサンプル値を、実行ロール ARN と *YOUR_REGION*内のリージョン (us-east-1など) に置き換えてください。

```
>>> extra=json.dumps({'role_arn': 'YOUR_EXECUTION_ROLE_ARN', 'region_name':
 'YOUR_REGION'})
```

5. 接続オブジェクトを作成します。myconnのサンプル値を Apache Airflow 接続の名前に置き換えてください。

```
>>> myconn = Connection(
```

6. キーボードの「Tab」キーを使用して、接続オブジェクト内の以下のキーと値のペアをそれぞれインデントします。サンプル値を#で置き換えてください。

   a. AWS 接続タイプを指定します。

```
... conn_id='aws',
```

   b. Apache Airflow データベースオプションを指定します。

```
... conn_type='mysql',
```

   c. Amazon MWAA の Apache Airflow UI URL を指定してください。

```
... host='288888a0-50a0-888-9a88-1a111aaa0000.a1.us-
east-1.airflow.amazonaws.com/home',
```

d. Amazon MWAA にログインするための AWS アクセスキー ID (ユーザー名) を指定します。

```
... login='YOUR_AWS_ACCESS_KEY_ID',
```

e. Amazon MWAA にログインする AWS シークレットアクセスキー (パスワード) を指定します。

```
... password='YOUR_AWS_SECRET_ACCESS_KEY',
```

f. extraシェルセッション変数を指定します。

```
... extra=extra
```

g. 接続オブジェクトを閉じます。

```
... )
```

7. 接続 URI 文字列を出力します。

```
>>> myconn.get_uri()
```

レスポンスに接続 URI 文字列が表示されるはずです。

```
'mysql://288888a0-50a0-888-9a88-1a111aaa0000.a1.us-east-1.airflow.amazonaws.com
%2Fhome?role_arn=arn%3Aaws%3Aiam%3A%3A001122332255%3Arole%2Fservice-role
%2FAmazonMWAA-MyAirflowEnvironment-iAaaaA&region_name=us-east-1'
```

Python script

次の Python スクリプトは、Apache エアフロー CLI を必要としません。

1. 以下のコードサンプルの内容をコピーし、ローカルに mwaa_connection.py として保存します。

```
import urllib.parse

conn_type = 'YOUR_DB_OPTION'
host = 'YOUR_MWAA_AIRFLOW_UI_URL'
port = 'YOUR_PORT'
```

```
login = 'YOUR_AWS_ACCESS_KEY_ID'
password = 'YOUR_AWS_SECRET_ACCESS_KEY'
role_arn = urllib.parse.quote_plus('YOUR_EXECUTION_ROLE_ARN')
region_name = 'YOUR_REGION'

conn_string = '{0}://{1}:{2}@{3}:{4}?
role_arn={5}&region_name={6}'.format(conn_type, login, password, host, port,
 role_arn, region_name)
print(conn_string)
```

2. プレースホルダーを#で置き換えてください。
3. 次のスクリプトを実行して、接続文字列を生成します。

```
python3 mwaa_connection.py
```

ステップ 4: Secrets Manager に変数を追加する

次のセクションでは、Secrets Manager で変数のシークレットを作成する方法について説明します。

シークレットを作成するには

1. AWS Secrets Manager コンソールを開きます。
2. [新しいシークレットを保存] を選択します。
3. 「他の種類のシークレット」を選択します。
4. 「このシークレットに保存するキーと値のペアを指定してください」ペインで、「プレーンテキスト」を選択します。
5. 変数値を次の形式でプレーンテキストとして追加します。

```
"YOUR_VARIABLE_VALUE"
```

たとえば、整数を指定するには:

```
14
```

例えば、文字列を指定する :

```
"mystring"
```

6. 暗号化キー で、ドロップダウンリストから AWS KMS キーオプションを選択します。
7. 「シークレット名」のテキストフィールドに、次の形式で名前を入力します。

```
airflow/variables/YOUR_VARIABLE_NAME
```

例:

```
airflow/variables/test-variable
```

8. [次へ] をクリックします。
9. 「シークレットの設定」ページの「シークレットの名前と説明」ペインで、次の操作を行います。

   a. [シークレット名] には、シークレットの名前を入力します。

   b. （オプション）「説明」として、シークレットの説明を入力します。

   [次へ] をクリックします。
10. 「ローテーションの設定-オプション」では、デフォルトオプションのままにして、「次へ」を選択します。
11. 追加したい変数があれば、Secrets Manager でこれらの手順を繰り返します。
12. 「レビュー」ページで、自分の秘密を確認し、「ストア」を選択します。

ステップ 5: Secrets Manager に接続を追加する

次のセクションでは、Secrets Manager で接続文字列 URI のシークレットを作成する方法について説明します。

シークレットを作成するには

1. AWS Secrets Manager コンソールを開きます。
2. [新しいシークレットを保存] を選択します。
3. 「他の種類のシークレット」を選択します。
4. 「このシークレットに保存するキーと値のペアを指定してください」ペインで、「プレーンテキスト」を選択します。
5. 接続 URI 文字列を次の形式でプレーンテキストとして追加します。

```
YOUR_CONNECTION_URI_STRING
```

例:

```
mysql://288888a0-50a0-888-9a88-1a111aaa0000.a1.us-east-1.airflow.amazonaws.com
%2Fhome?role_arn=arn%3Aaws%3Aiam%3A%3A001122332255%3Arole%2Fservice-role
%2FAmazonMWAA-MyAirflowEnvironment-iAaaaA&region_name=us-east-1
```

> **Warning**
>
> Apache Airflow は接続文字列の各値を解析します。一重引用符や二重引用符は使用しないでください。そうしないと、接続が 1 つの文字列として解析されます。

6. 暗号化キー で、ドロップダウンリストから AWS KMS キーオプションを選択します。
7. 「シークレット名」のテキストフィールドに、次の形式で名前を入力します。

```
airflow/connections/YOUR_CONNECTION_NAME
```

例:

```
airflow/connections/myconn
```

8. [次へ] をクリックします。
9. 「シークレットの設定」ページの「シークレットの名前と説明」ペインで、次の操作を行います。

   a. [シークレット名] には、シークレットの名前を入力します。

   b. （オプション）「説明」として、シークレットの説明を入力します。

   [次へ] をクリックします。
10. 「ローテーションの設定-オプション」では、デフォルトオプションのままにして、「次へ」を選択します。
11. 追加したい変数があれば、Secrets Manager でこれらの手順を繰り返します。
12. 「レビュー」ページで、自分の秘密を確認し、「ストア」を選択します。

「サンプルコード」

- このページの Apache Airflow 接続 (`myconn`) のシークレットキーの使用方法については、AWS Secrets Manager の Apache Airflow 接続でのシークレットキーの使用のサンプルコードを参照してください。
- このページの Apache Airflow 変数 (`test-variable`) のシークレットキーの使用方法については、。Apache Airflow 変数の AWS Secrets Manager におけるシークレットキーの使用にあるサンプルコードを使用して学習してください

リソース

- コンソールと を使用して Secrets Manager シークレットを設定する方法の詳細については AWS CLI、「 AWS Secrets Manager ユーザーガイド」の「シークレットの作成」を参照してください。
- Apache Airflowの大量の変数や接続をSecrets Managerに移行するために、Pythonスクリプトを使用して、Apache Airflowの接続と変数を AWS Secrets Managerに移動してください

次のステップ

- 「Apache Airflow へのアクセス」で Apache Airflow UI にアクセスするトークンを生成する方法について説明します。

Amazon MWAA 環境の管理

Apache Airflow 用 Amazon マネージドワークフローコンソールには、Apache Airflow UI へのプライベートアクセスまたはパブリックアクセスを構成するための組み込みオプションが含まれています。また、環境サイズ、ワーカーをスケーリングするタイミングを構成する組み込みオプション、Apache Airflow 構成オプションも用意されています。これにより、通常は `airflow.cfg` でのみアクセス可能な Apache Airflow 構成をオーバーライドできます。このガイドでは、Amazon MWAA コンソールでこれらの構成を使用する方法について説明します。

トピック

- Amazon MWAA 環境クラスの構成
- Amazon MWAA ワーカーの自動スケーリングの設定
- Amazon MWAA ウェブサーバーの自動スケーリングの設定
- Amazon MWAA での Apache Airflow 構成オプションの使用
- Apache Airflow バージョンのアップグレード
- Amazon MWAA でのスタートアップスクリプトの使用

Amazon MWAA 環境クラスの構成

Amazon MWAA 環境用に選択した環境クラスによって、Celery Executor が実行される AWSマネージド AWS Fargate コンテナのサイズと、Apache Airflow スケジューラがタスクインスタンスを作成する AWSマネージド Amazon Aurora PostgreSQL メタデータデータベースが決まります。このページでは、Amazon MWAA の各環境クラスと、Amazon MWAA コンソールで環境クラスを更新する手順について説明します。

セクション

- 環境機能
- Apache Airflow スケジューラー

環境機能

次のセクションには、各環境クラスのデフォルトの同時 Apache Airflow タスク、ランダムアクセスメモリ (RAM)、および仮想中央処理装置 (vCPUs) が含まれています。記載されている同時実行タス

クは、タスクの同時実行性が環境内の Apache Airflow ワーカーのキャパシティを超えないことを前提としています。

次の表では、DAG 容量は実行ではなく DAG 定義を参照し、DAGs が単一の Python ファイルで動的であり、Apache Airflow のベストプラクティス で記述されていることを前提としています。

タスクの実行は、同時にスケジュールされる DAG の数によって異なり、同時に開始するように設定されている DAG 実行の数と、このトピックで詳しく説明するワーカーのサイズと数がデフォルト「`max_dagruns_per_loop_to_schedule`」を超えないことを前提としています。

mw1.small

- 最大 50 個の DAG 容量
- 同時 5 タスク (デフォルト)
- 1 vCPU
- 2 GB RAM

mw1.medium

- 最大 200 個の DAG 容量
- 同時 10 タスク (デフォルト)
- 2 vCPU
- 4 GB RAM

mw1.large

- 最大 1000 個の DAG 容量
- 同時 20 タスク (デフォルト)
- 4 vCPU
- 8 GB RAM

mw1.xlarge

- 最大 2000 個の DAG 容量
- 40 個の同時タスク (デフォルト)

- 8 vCPUs
- 24 GB RAM

mw1.2xlarge

- 最大 4000 個の DAG 容量
- 80 個の同時タスク (デフォルト)
- 16 vCPU
- 48 GB RAM

`celery.worker_autoscale` を使用して、作業者 1 人あたりのタスク数を増やすことができます。詳細については、「the section called “高パフォーマンスのユースケースの例”」を参照してください。

Apache Airflow スケジューラー

以下のセクションでは、Amazon MWAA で使用できる Apache Airflow スケジューラーオプションと、スケジューラーの数がトリガーの数にどのように影響するかについて説明します。

Apache Airflow では、「トリガー」を使用して指定された特定の条件が満たされるまでタスクを延期するトリガーがタスクを管理します。Amazon MWAA では、トリガーは同じ Fargate タスクでスケジューラーと並行して実行されます。スケジューラーの数を増やすと、それに応じて使用可能なトリガーの数も増え、遅延されたタスクを環境がどのように管理するかが最適化されます。これにより、タスクを効率的に処理できるようになり、条件が満たされた時点でタスクを迅速に実行するようにスケジューリングできます。

Apache Airflow v2

- v2 - 2 から 5 までを受け入れます。デフォルトは 2 です。

Amazon MWAA ワーカーの自動スケーリングの設定

自動スケーリングメカニズムは、Amazon Managed Workflows for Apache Airflow 環境で実行中のタスクとキューに入れられたタスクに応じて Apache Airflow ワーカーの数を自動的に増やし、キューに入れられたタスクや実行中のタスクがなくなったときに余分なワーカーを破棄します。このページでは、Amazon MWAA コンソールを使用して環境で実行される Apache Airflow ワーカーの最大数を指定して、自動スケーリングを設定する方法について説明します。

> **Note**
>
> Amazon MWAA は Apache Airflow メトリクスを使用して Celery Executor ワーカーをいつ追加する必要があるかを判断し、必要に応じて Fargate ワーカーの数を`max-workers`で指定された値まで増やします。追加のワーカーが作業を完了し、作業負荷が減少すると、Amazon MWAA はそれらを削除し、 によって設定された値にダウンスケーリングします`min-workers`。
> ワーカーがダウンスケーリング中に新しいタスクを取得した場合、Amazon MWAA は Fargate リソースを保持し、ワーカーを削除しません。詳細については、「Amazon MWAA Auto Scaling の仕組み」を参照してください。

セクション

ワーカースケーリングの仕組み

Amazon MWAA は、(実行中のタスク + キューに入れられたタスク) / (ワーカーごとのタスク) = (必要なワーカー) という`RunningTasks`と`QueuedTasks`のメトリクスを使用します。必要なワーカー数が現在のワーカー数よりも多い場合、Amazon MWAA は Fargate ワーカーコンテナを、`max-workers`で指定された最大値までその値に追加します。

ワークロードが減少し、 `RunningTasks`および `QueuedTasks`メトリクスの合計が減少すると、Amazon MWAA は Fargate に環境のワーカーのスケールダウンを要求します。まだ作業を完了しているワーカーは、作業が完了するまでダウンスケーリング中も保護されます。ワークロードによっては、ワーカーがダウンスケールしている間にタスクがキューに入れられる場合があります。

Amazon MWAA コンソールの使用

Amazon MWAA コンソールでは、環境上で同時に実行できるワーカーの最大数を選択できます。デフォルトでは、最大値を 25 まで指定できます。

ワーカー数を設定するには

1. Amazon MWAA コンソールで「環境ページ」を開きます。
2. 環境を選択します。
3. [編集] を選択します。
4. [次へ] をクリックします。
5. [環境クラス] ペインで、[最大ワーカー数] に値を入力します。
6. [保存] を選択します。

> **Note**
>
> 環境への変更が適用されるまで数分ほどかかります。

高パフォーマンスのユースケースの例

次のセクションでは、ある環境で高いパフォーマンスと並列処理を実現するために使用できる構成の種類について説明します。

オンプレミスの Apache エアフロー

通常、オンプレミスの Apache Airflow プラットフォームでは、 `airflow.cfg` ファイルでタスクの並列処理、自動スケーリング、および同時実行の設定を行います。

- `core.parallelism`— スケジューラーごとに同時に実行できるタスクインスタンスの最大数。
- `core.dag_concurrency`— DAG (ワーカーではない) の最大同時実行数。
- `celery.worker_autoscale`— 任意のワーカーで同時に実行できるタスクの最大数と最小数。

例えば、`core.parallelism`が`100`に設定され、`core.dag_concurrency`が`7`に設定されている場合、2つのDAGがある場合でも、同時に合計14つのタスクしか実行できません。各DAGが同時に実行されるタスク数が`core.dag_concurrency`で`7`に設定されている場合、全体の並列度が`core.parallelism`で`100`に設定されていても、同時に実行されるタスクは最大で7つに制限されます。

Amazon MWAA 環境で

Amazon MWAA 環境では、、Amazon MWAA での Apache Airflow 構成オプションの使用、および最大ワーカー数の自動スケーリングメカニズムを使用してAmazon MWAA 環境クラスの構成、Amazon MWAA コンソールでこれらの設定を直接設定できます。`core.dag_concurrency` は Amazon MWAA コンソールの Apache Airflow 設定オプションとしてドロップダウンリストでは使用できませんが、カスタム Apache Airflow 設定オプション として追加できます。

環境を作成したときに、次の設定を選択したとします。

1. mw1.small 環境クラスは、各ワーカーがデフォルトで実行できる同時タスクの最大数とコンテナーの vCPU を制御します。
2. 「最大ワーカー数」のデフォルト設定は`10`のワーカーです。
3. `celery.worker_autoscale`のワーカーあたりの`5,5`のタスクのための Apache Airflow 構成オプション。

つまり、環境内で 50 件のタスクを同時に実行できます。50 個を超えるタスクはキューに入れられ、実行中のタスクが完了するまで待機します。

同時実行タスクをさらに実行する。以下の構成を使用して、より多くのタスクを同時に実行するように環境を変更できます。

1. 環境クラス `mw1.medium` (デフォルトでは 10 個の同時タスク) を選択して、各ワーカーがデフォルトで実行できる同時タスクの最大数とコンテナの vCPU を増やします。
2. `celery.worker_autoscale`をApache Airflow 構成オプションとして追加してください。
3. 最大ワーカー数を増やしてください。この例では、最大ワーカーを`10`から`20`に増やすと、環境が同時に実行できるタスクの数が倍になります。

最小ワーカー数を指定します。 AWS Command Line Interface () を使用して、環境で実行される Apache Airflow ワーカーの最小数と最大数を指定することもできますAWS CLI。例:

```
aws mwaa update-environment --max-workers 10 --min-workers 10 --
name YOUR_ENVIRONMENT_NAME
```

詳細については、 AWS CLIの update-environment コマンドを参照してください。

タスクが実行状態で止まってしまう問題のトラブルシューティング

まれに、Apache Airflow がまだ実行中のタスクがあると判断することがあります。この問題を解決するには、Apache Airflow UI で孤立したタスクをクリアする必要があります。詳細については、トラブルシューティングトピック「タスクが行き詰まっているか、完了していません。」 を参照してください。

次のステップ

- ご使用の環境のパフォーマンスを調整するために推奨するベスト・プラクティスの詳細については、「Amazon MWAA での Apache Airflow のパフォーマンス調整」 を参照してください。

Amazon MWAA ウェブサーバーの自動スケーリングの設定

Apache Airflow Apache Airflow v2.2.2 以降を実行している環境では、Amazon MWAA は変動するワークロードを処理するようにウェブサーバーを動的にスケーリングし、ピーク負荷時のパフォーマンスの問題を防ぎます。CPU 使用率とアクティブな接続数に基づいてウェブサーバーの数を自動的にスケーリングすることで、Amazon MWAA は、REST API リクエスト、CLI 使用状況、または複数の同時 Apache Airflow ユーザーインターフェイスユーザーのいずれからでも、Apache Airflow 環境が需要の増加にシームレスに対応できるようにします。

セクション

- ウェブサーバーのスケーリングの仕組み
- Amazon MWAA コンソールの使用

ウェブサーバーのスケーリングの仕組み

Amazon MWAA は、コンテナメトリクス `CPUUtilization`とロードバランサーメトリクス を使用して`ActiveConnectionCount`、トラフィックの量に基づいてウェブサーバーのスケーリングが必要かどうかを判断します。`CPUUtilization` が 70 より大きい場合、または 15 `ActiveConnectionCount`より大きい場合、Amazon MWAA は で指定された最大値まで Fargate ウェブサーバーコンテナを追加します`MaxWebservers`。

トラフィックが減少し、 `CPUUtilization` と のActiveConnectionCount値が減少すると、Amazon MWAA は Fargate に環境のウェブサーバーコンテナを で設定された最小値にスケールダウンするよう要求します`MinimumWebservers`。

Amazon MWAA コンソールの使用

Amazon MWAA コンソールで、環境で同時に実行できるウェブサーバーの数を選択できます。デフォルトでは、ウェブサーバーの最小数は 2 で、ウェブサーバーの最大数は 5 です。

ウェブサーバーの数を設定するには

1. Amazon MWAA コンソールで「環境ページ」を開きます。
2. 環境を選択します。
3. [編集] を選択します。
4. [次へ] をクリックします。
5. 環境クラスペインで、ウェブサーバーの最大数 に値を入力します。
6. 次に、「ウェブサーバーの最小数」に値を入力します。
7. [保存] を選択します。

> **Note**
>
> 環境への変更が適用されるまで数分ほどかかります。

Amazon MWAA での Apache Airflow 構成オプションの使用

Apache Airflow 構成オプションは、Apache Airflow 用 Amazon マネージドワークフロー環境に環境変数としてアタッチできます。推奨ドロップダウンリストから選択するか、Amazon MWAA コンソールでお使いの Apache Airflow バージョンに合わせてカスタム構成オプションを指定できます。このページでは、使用可能な Apache Airflow 構成オプションと、これらのオプションを使用して環境の Apache Airflow 構成設定を上書きする方法について説明します。

目次

前提条件

このページのステップを完了するには、以下のものが必要です。

- アクセス許可 — AWS アカウントには、管理者から環境の AmazonMWAAFullConsole アクセスコントロールポリシーへのアクセスが付与されている必要があります。さらに、Amazon MWAA 環境は、環境で使用される AWS リソースへのアクセスを実行ロールで許可する必要があります。
- アクセス — 依存関係をウェブサーバーに直接インストールするためにパブリックリポジトリにアクセスする必要がある場合は、パブリックネットワークのウェブサーバーアクセスが環境に設定されている必要があります。詳細については、「the section called “Apache Airflow のアクセスモード”」を参照してください。
- Amazon S3 設定 — `plugins.zip` で DAG、カスタムプラグイン、および `requirements.txt` で Python の依存関係を保存するために使用される「Amazon S3 バケット」は、Public Access Blocked と Versioning Enabledで構成する必要があります。

仕組み

環境を作成すると、Amazon MWAA は Airflow 設定オプションの Amazon MWAA コンソールで指定した設定を環境変数として環境の AWS Fargate コンテナにアタッチします。`airflow.cfg`で

同じ名前の設定を使用している場合は、Amazon MWAA コンソールで指定するオプションのほうが、airflow.cfgの値よりも優先されます。

デフォルトでは、Amazon MWAA 環境の Apache Airflow UI airflow.cfgで を公開しませんが、設定を公開webserver.expose_configする設定を含め、Amazon MWAA コンソールで Apache Airflow 設定オプションを直接変更できます。

設定オプションを使用して Apache Airflow v2 にプラグインをロードする

Apache Airflow v2 のデフォルトでは、プラグインは設定を使用して「遅延的に」ロードされるようにcore.lazy_load_plugins : True設定されています。Apache Airflow v2でカスタムプラグインを使用している場合、デフォルトの設定をオーバーライドするために、Airflow プロセスの開始時にプラグインをロードするためのApache Airflow 構成オプションとしてcore.lazy_load_plugins : Falseを追加する必要があります。を追加する必要があります。

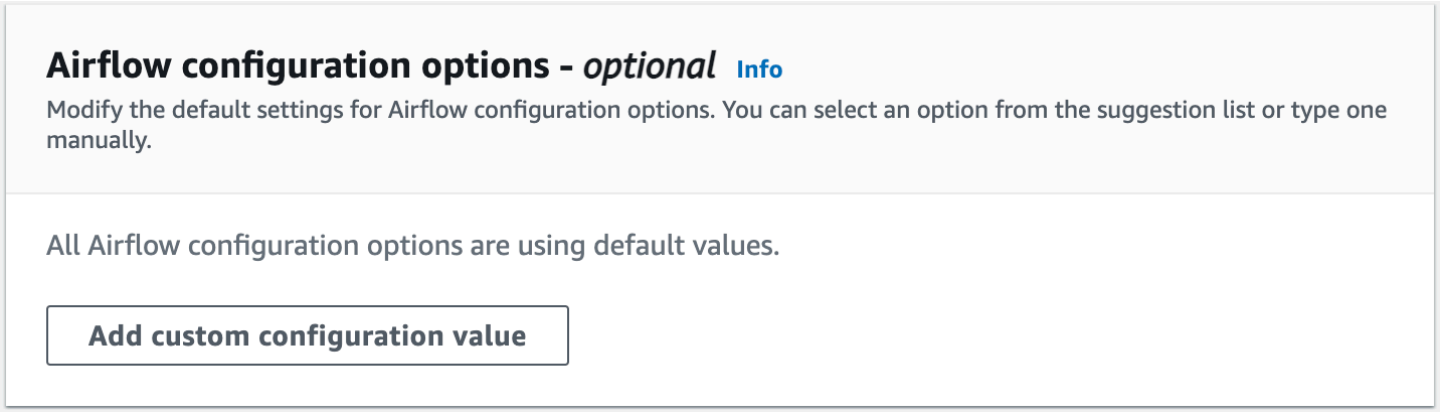
設定オプションの概要

Amazon MWAA コンソールで設定を追加すると、Amazon MWAA はその設定を環境変数として書き込みます。

- リストされたオプション。ドロップダウンリストで、お使いの Apache Airflow バージョンで使用可能な設定のいずれかを選択できます。例えば、dag_concurrency : 16 です。その構成設定は、環境のFargateコンテナにAIRFLOW__CORE__DAG_CONCURRENCY : 16として反映されます。
- 「カスタムオプション」 お使いの Apache Airflow バージョンにはない Airflow 設定オプションをドロップダウンリストに指定することもできます。例えば、foo.user : YOUR_USER_NAME です。その構成設定は、環境のFargateコンテナにAIRFLOW__FOO__USER : YOUR_USER_NAMEとして反映されます。

Apache Airflow 構成オプション

以下の画像は、Amazon MWAA コンソールで Apache Airflow 構成オプションをカスタマイズできる場所を示しています。



Apache エアフローリファレンス

Apache Airflow でサポートされている設定オプションのリストについては、『Apache Airflow リファレンスガイド』の「設定リファレンス」を参照してください。Amazon MWAA で実行している Apache Airflow のバージョンのオプションを表示するには、ドロップダウンリストからバージョンを選択します。

Amazon MWAA コンソールの使用

以下の手順では、Airflow 構成オプションを環境に追加するステップを説明します。

1. Amazon MWAA コンソールで「環境ページ」を開きます。
2. 環境を選択します。
3. [編集] を選択します。
4. [次へ] をクリックします。
5. Airflow 設定オプションペインで [カスタム設定を追加] を選択します。
6. ドロップダウンリストから構成を選択して値を入力するか、カスタム構成を入力して値を入力します。
7. 追加する設定ごとに [カスタム設定を追加] を選択します。
8. [保存] を選択します。

設定リファレンス

以下のセクションでは、Amazon MWAA コンソールのドロップダウンリストに利用可能な Apache Airflow 設定のリストが表示されます。

電子メールの設定

次のリストは、Amazon MWAA で使用できる Airflow E メール通知設定オプションを示しています。

SMTP トラフィックにはポート 587 を使用することをお勧めします。デフォルトでは、 はすべての Amazon EC2 インスタンスのポート 25 でアウトバウンド SMTP トラフィックを AWS ブロックします。ポート 25 でアウトバウンド· トラフィックを送信したい場合は、「この制限を解除するようリクエストする」ことができます。

Apache Airflow v2

Airflow のバージョン	エアフロー設定のオプション	説明	値の例
v2	メール.メール_バックエンド	email_backend のメール通知に使用される Apache エアフローユーティリティ。	airflow.utils.email.send_email_smtp
v2	smtp.smtp_host	smtp_host の電子メールアドレスに使用される送信サーバーの名前。	localhost
v2	smtp.smtp_starttls	Transport Layer Security (TLS) は、smtp_starttlsでインターネットの E メールの暗号化に使用されます。	False
v2	smtp.smtp_ssl	セキュアソケットレイヤー (SSL) は、smtp_ssl でサーバーとメールクライアントを接続するために使用されます。	True

Airflow のバージョン	エアフロー設定のオプション	説明	値の例
v2	smtp.smtp_port	smtp_portでサーバに指定されたTransmission Control Protocol (TCP)。	587
v2	smtp.smtp_mail_from	smtp_mail_fromにあるアウトバウンドメールアドレス。	myemail@domain.com

タスクの設定

次のリストは、Amazon MWAA の Airflow タスクのドロップダウンリストで使用できる設定を示しています。

Apache Airflow v2

Airflow のバージョン	エアフロー設定のオプション	説明	値の例
v2	core .default_task_retries	Apache エアフロータスクを default_task_retriesでリトライする回数。	3
v2	コア・パラレリズム	環境全体で同時に実行できるタスクインスタンスの最大数 (parallel 処理)。	40

スケジューラーの設定

次のリストは、Amazon MWAA のドロップダウンリストで使用できる Apache Airflow スケジューラー設定を示しています。

Apache Airflow v2

Airflow のバージョン	エアフロー設定のオプション	説明	値の例
v2	スケジューラー.キャッチアップ・バイ・デフォルト	catchup_by_default の特定の時間間隔に「catch く」ための DAG 実行を作成するようにスケジューラーに指示します。	False
v2	scheduler.scheduler_zombie_task_threshold	タスクインスタンスを障害としてマークし、scheduler_zombie_task_threshold内のタスクを再スケジュールするかどうかをスケジューラーに指示します。	300

ワーカーの設定

次のリストは、Amazon MWAA のドロップダウンリストで使用できる Airflow ワーカー設定を示しています。

Apache Airflow v2

Airflow のバージョン	エアフロー設定のオプション	説明	値の例
v2	celery.worker_autoscale	worker_autoscale の Celery Executor を使用するすべてのワーカーで同時に実行できるタスクの最大数と最小数。値は	16、12

Airflow のバージョン	エアフロー設定のオプション	説明	値の例
		次の順序でカンマで区切る必要があります:`max_concurrency,min_concurrency`	

ウェブサーバーの設定

次のリストは、Amazon MWAA のドロップダウンリストで使用できる Airflow ウェブサーバー設定を示しています。

Apache Airflow v2

Airflow のバージョン	エアフロー設定のオプション	説明	値の例
v2	ウェブサーバー.default_ui_timezone	default_ui_timezone の Apache Airflow UI のデフォルト日時設定。 **Note** この`default_ui_timezone` オプションを設定しても、DAG の実行がスケジュールされているタイムゾーンは変更されませ	America/New_York

Airflow のバージョン	エアフロー設定のオプション	説明	値の例
		ん。DAG のタイムゾーンを変更するには、カスタムプラグインを使用できます。詳細については、「the section called “DAG のタイムゾーンの変更”」を参照してください。	

トリガー設定

以下のリストは、Amazon MAA で利用できる Apache エアフロートリガー設定を示しています。

Apache Airflow v2

Airflow のバージョン	エアフロー設定のオプション	説明	値の例
v2.7	mwaa トリガー_enabled	Amazon MAA でトリガーをアクティブ化および非アクティブ化するために使用されます。デフォルトでは、この値は `True` に設定されます。`False`に設定すると、Amazon	True

Airflow のバージョン	エアフロー設定のオプション	説明	値の例
		MWAA はスケジューラーのトリガープロセスを開始しません。	
v2.7	トリガー. デフォルト容量	各トリガーでparallelに実行できる、トリガーの数を定義します。Amazon MWAA では、両方のコンポーネントが互いに並行して実行されるため、この容量はトリガーごととスケジューラーごとに設定されます。スケジューラごとのデフォルトは、スモール60インスタンス、ミディアムインスタンス500、ラージインスタンス、xlarge インスタンス、および125 2502xlarge インスタンス1000で、それぞれ 、、、、および に設定されます。	125

例とサンプルコード

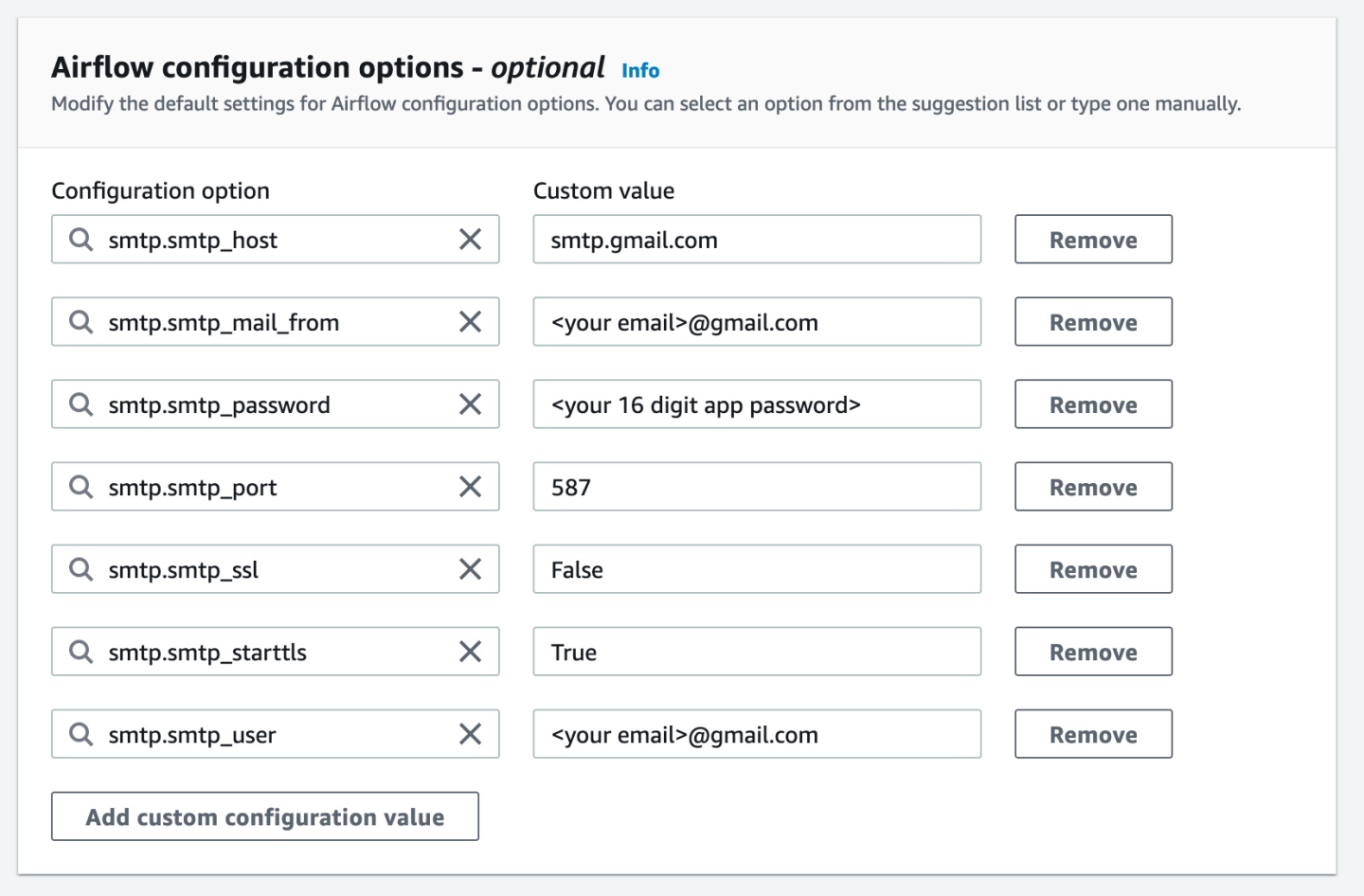
DAG の例

次の DAG を使用して email_backend Apache Airflow 構成オプションを印刷できます。Amazon MWAA イベントに応答して実行するには、Amazon S3 ストレージバケットにある環境の DAG フォルダにコードをコピーします。

```
from airflow.decorators import dag
from datetime import datetime

def print_var(**kwargs):
  email_backend = kwargs['conf'].get(section='email', key='email_backend')
  print("email_backend")
  return email_backend

@dag(
  dag_id="print_env_variable_example",
  schedule_interval=None,
  start_date=datetime(yyyy, m, d),
  catchup=False,
)
def print_variable_dag():
  email_backend_test = PythonOperator(
  task_id="email_backend_test",
  python_callable=print_var,
  provide_context=True
)

print_variable_test = print_variable_dag()
```

電子メール通知の設定例

アプリパスワードを使用する Gmail.com メールアカウントには、次の Apache Airflow 構成オプションを使用できます。詳しくは、Gmail ヘルプリファレンスガイドの「アプリパスワードを使用してログインする」を参照してください。



次のステップ

- DAG の追加と更新で、DAG フォルダを Amazon S3 バケットにアップロードする方法について説明します。

Apache Airflow バージョンのアップグレード

Amazon MWAA はマイナーバージョンアップグレードをサポートしています。これは、環境をバージョン x.**4**.z から x.**5**.z にアップグレードできることを意味します。メジャーバージョンのアップグレード、たとえばバージョン **1**.y.z から **2**.y.z へのアップグレードを行うには、新しい環境を作成し、リソースを移行する必要があります。Apache Airflow の新しいメジャーバージョンへのアップグレードの詳細については、Amazon MWAA 移行ガイドの「新しい Amazon MWAA 環境への移行」を参照してください。

アップグレードプロセス中、Amazon MWAA は環境メタデータのスナップショットをキャプチャし、ワーカー、スケジューラー、ウェブサーバーを新しい Apache Airflow バージョンにアップグレードし、最後にそのスナップショットを使用してメタデータデータベースを復元します。

> **Note**
>
> ご使用の環境に合わせて Apache Airflow バージョンをダウングレードすることはできません。

アップグレードする前に、DAG やその他のワークフローリソースが、アップグレード先の新しい Apache Airflow バージョンと互換性があることを確認してください。`requirements.txt` を使用して依存関係を管理する場合は、要件で指定する依存関係が新しいバージョンと互換性があることも確認する必要があります。

トピック

- ワークフローリソースのアップグレード
- 新しいバージョンを指定してください

ワークフローリソースのアップグレード

Apache Airflow のバージョンを変更するときは必ず、`--constraint` 内の正しい `requirements.txt` URL お客様するようにしてください。

> **Warning**
>
> アップグレード中に対象の Apache Airflow バージョンと互換性のない要件を指定すると、以前の要件バージョンの Apache Airflow へのロールバックプロセスに時間がかかる可能性があります。

ワークフローリソースを移行するには

1. aws-mwaa-local-runner リポジトリのフォークを作成し、Amazon MWAA ローカルランナーのコピーをクローンしてください。
2. アップグレードしようとしているバージョンと一致する aws-mwaa-local-runner リポジトリのブランチにチェックアウトします。

3. Amazon MWAA ローカルランナー CLI ツールを使用して Docker イメージをビルドし、Apache Airflow をローカルで実行します。詳細については、GitHub リポジトリのローカル・ランナー「README」を参照してください。
4. `requirements.txt` を更新するには、Amazon MWAA ユーザーガイドの「Python 依存関係の管理」で推奨されているベストプラクティスに従ってください。
5. (オプション) アップグレードプロセスをスピードアップするには、「環境のメタデータデータベースをクリーンアップ」します。メタデータの量が多い環境では、アップグレードにかなり時間がかかることがあります。
6. ワークフローリソースのテストに成功したら、DAG、`requirements.txt`、およびプラグインを環境の Amazon S3 バケットにコピーします。

これで、環境を編集し、新しい Apache Airflow バージョンを指定して、更新手順を開始する準備ができました。

新しいバージョンを指定してください

ワークフローリソースの更新を完了して新しい Apache Airflow バージョンとの互換性を確認したら、次の手順を実行して環境の詳細を編集し、アップグレードする Apache Airflow のバージョンを指定します。

> **Note**
>
> アップグレードを実行すると、その環境で現在実行されているすべてのタスクが手順中に終了します。アップデート手順には最大 2 時間かかることがあり、その間は環境が使用できなくなります。

コンソールを使用して新しいバージョンを指定するには

1. Amazon MWAA コンソールで「環境ページ」を開きます。
2. [環境] のリストで、アップグレードしたい環境を選択します。
3. 環境ページで [編集] を選択して環境を編集します。
4. [環境の詳細] セクションの [Airflow バージョン] で、環境をアップグレードしたい新しい Apache Airflow バージョン番号をドロップダウンリストから選択します。
5. [レビューと保存] ページが表示されるまで、[次へ] を選択します。

6. [レビューと保存] ページで変更内容を確認し、[保存] を選択します。

変更を適用すると、環境がアップグレード手順を開始します。この期間中、Amazon MWAA がどのようなアクションを実行しているか、および手順が成功したかどうかが環境の「ステータス」でわかります。

アップグレードが成功したシナリオでは、ステータスは UPDATING が表示され、その後 CREATING_SNAPSHOT となり、Amazon MWAA がメタデータのバックアップをキャプチャします。最後に、ステータスはまず UPDATING に戻り、手順が完了すると AVAILABLE に戻ります。

環境のアップグレードに失敗した場合、環境のステータスは ROLLING_BACK を表示します。ロールバックが成功した場合、ステータスは最初に UPDATE_FAILED を表示し、これは更新が失敗したが環境は利用可能であることを示します。ロールバックが失敗した場合、ステータスは UNAVAILABLE を表示し、環境にアクセスできないことを示します。

Amazon MWAA でのスタートアップスクリプトの使用

スタートアップスクリプトは、DAG、要件、プラグインと同様に、環境の Amazon S3 バケットでホストするシェル (.sh) スクリプトです。Amazon MWAA は、要件のインストールと Apache Airflow プロセスの初期化の前に、このスクリプトをすべての個々の Apache Airflow コンポーネント (ワーカー、スケジューラ、ウェブサーバー) で起動中に実行します。起動スクリプトを使用して次のことができます。

- ランタイムのインストール — ワークフローと接続に必要な Linux ランタイムをインストールします。
- 環境変数の設定 — Apache Airflow コンポーネントごとに環境変数を設定します。PATH、PYTHONPATH、LD_LIBRARY_PATH などの一般的な変数を上書きします。
- キーとトークンの管理 – カスタムリポジトリへのアクセストークンを requirements.txt に渡し、セキュリティキーを構成します。

以下のトピックでは、Linux ランタイムをインストールし、環境変数を設定し、CloudWatch Logs を使用して関連する問題をトラブルシューティングするための起動スクリプトを設定する方法について説明します。

トピック

- スタートアップスクリプトを設定します。

- スタートアップスクリプトを使用して Linux ランタイムをインストールします。
- 起動スクリプトを使用して環境変数を設定する

スタートアップスクリプトを設定します。

既存の Amazon MWAA 環境で起動スクリプトを使用するには、環境の Amazon S3 バケットに .sh ファイルをアップロードします。次に、スクリプトを環境に関連付けるには、環境の詳細で以下を指定します。

- スクリプトへの Amazon S3 URL パス — バケットでホストされているスクリプトへの相対パス。例: `s3://mwaa-environment/startup.sh`
- スクリプトの Amazon S3 バージョン ID — Amazon S3 バケット内の起動シェルスクリプトのバージョン。スクリプトを更新するたびに、Amazon S3 がファイルに割り当てる「バージョン ID」を指定する必要があります。バージョン ID は、ユニコード、UTF-8 エンコード、URL レディ、不透明文字列で、長さは 1,024 バイト以下で、例えば `3sL4kqtJlcpXroDTDmJ+rmSpXd3dIbrHY+MTRCxf3vjVBH40Nr8X8gdRQBpUMLUo` です。

このセクションの手順を完了するには、次のサンプルスクリプトを使用します。このスクリプトは、`MWAA_AIRFLOW_COMPONENT` に割り当てられた値を出力します。この環境変数は、スクリプトが実行される各 Apache Airflow コンポーネントを識別します。

コードをコピーし、`startup.sh` としてローカルに保存します。

```
#!/bin/sh

echo "Printing Apache Airflow component"
echo $MWAA_AIRFLOW_COMPONENT
```

次に、スクリプトを Amazon S3 バケットにアップロードします。

AWS Management Console

シェルスクリプトをアップロードするには (コンソール)

1. AWS Management Console にサインインし、Amazon S3 コンソール https://console.aws.amazon.com/s3/ を開きます。
2. [バケット] リストで、環境に関連付けられているバケットの名前を選択します。

3. [オブジェクト] タブで、[アップロード] を選択します。
4. [アップロード] ページで、作成したシェルスクリプトをドラッグアンドドロップします。
5. [アップロード] を選択します。

スクリプトは [オブジェクト] のリストに表示されます。Amazon S3 によって、ファイルの新しいバージョン ID が作成されます。スクリプトを更新し、同じファイル名を使用して再度アップロードすると、新しいバージョン ID がファイルに割り当てられます。

AWS CLI

シェルスクリプト (CLI) を作成してアップロードするには

1. 新しいコマンドプロンプトを開き、Amazon S3 `ls` コマンドを実行して、環境に関連付けられているバケットを一覧表示して識別します。

```
$ aws s3 ls
```

2. シェルスクリプトを保存したフォルダに移動します。新しいプロンプトウィンドウで cp を使用して、スクリプトをバケットにアップロードします。*your-s3-bucket* を情報に置き換えてください。

```
$ aws s3 cp startup.sh s3://your-s3-bucket/startup.sh
```

成功すると、Amazon S3 はオブジェクトへの URL パスを出力します。

```
upload: ./startup.sh to s3://your-s3-bucket/startup.sh
```

3. 次のコマンドを使って、スクリプトの最新バージョン ID を取得します。

```
$ aws s3api list-object-versions --bucket your-s3-bucket --prefix startup --
query 'Versions[?IsLatest].[VersionId]' --output text
```

```
BbdVMmBRjtestta1EsVnbybZp1Wqh1J4
```

このバージョン ID は、スクリプトを環境に関連付けるときに指定します。

次に、スクリプトを環境に関連付けます。

AWS Management Console

スクリプトを環境に関連付けるには (コンソール)

1. Amazon MWAA コンソールで「環境ページ」を開きます。
2. 更新する環境の行を選択し、[編集] を選択します。
3. [詳細の指定] ページの [スタートアップスクリプトファイル — オプション] に、スクリプトの Amazon S3 URL を入力します 例: `s3://your-mwaa-bucket/startup-sh.`。
4. ドロップダウンリストから最新バージョンを選択するか、[ブラウザ S3] をクリックしてスクリプトを検索します。
5. [次] を選択して、[レビュー] ページを開きます。
6. 変更を確認し、[保存] を選択します。

環境の更新には 10 分から 30 分かかることがあります。Amazon MWAA は、環境内の各コンポーネントの再起動時に起動スクリプトを実行します。

AWS CLI

スクリプトを環境 (CLI) に関連付けるには

- コマンドプロンプトを開き、`update-environment` を使用してスクリプトの Amazon S3 URL とバージョン ID を指定します。

```
$ aws mwaa update-environment \
    --name your-mwaa-environment \
    --startup-script-s3-path startup.sh \
    --startup-script-s3-object-version BbdVMmBRjtestta1EsVnbybZp1Wqh1J4
```

成功すると、Amazon MWAA から環境の Amazon リソースネーム (ARN) が返されます。

```
arn:aws::airflow:us-west-2:123456789012:environment/your-mwaa-environment
```

環境の更新には 10 分から 30 分かかることがあります。Amazon MWAA は、環境内の各コンポーネントの再起動時に起動スクリプトを実行します。

最後に、ログイベントを取得して、スクリプトが期待どおりに動作していることを確認します。各 Apache Airflow コンポーネントのロギングを有効にすると、Amazon MWAA は新しいロググループ

とログストリームを作成します。詳細については、「Apache Airflow ログタイプ」を参照してください。

AWS Management Console

Apache Airflow ログストリームを確認するには (コンソール)

1. Amazon MWAA コンソールで「環境ページ」を開きます。
2. 環境を選択します。
3. [モニタリング] ペインで、ログを表示したいロググループ ([Airflow スケジューラロググループ] など) を選択します。
4. CloudWatch コンソールで、[ログストリーム] リストから以下のプレフィックスを持つストリームを選択します: `startup_script_exection_ip`.
5. [ログイベント] ペインには、`MWAA_AIRFLOW_COMPONENT` の値を表示するコマンドの出力が表示されます。たとえば、スケジューラーログの場合、次のようになります。

```
Printing Apache Airflow component
scheduler
Finished running startup script. Execution time: 0.004s.
Running verification
Verification completed
```

前の手順を繰り返して、ワーカーとウェブサーバーのログを表示できます。

スタートアップスクリプトを使用して Linux ランタイムをインストールします。

起動スクリプトを使用して Apache Airflow コンポーネントのオペレーティングシステムを更新し、ワークフローで使用する追加のランタイムライブラリをインストールします。例えば、次のスクリプトは `yum update` を実行してオペレーティングシステムを更新します。

スタートアップスクリプトで `yum update` を実行する際は、例に示すように `--exclude=python*` を使用して Python を除外する必要があります。環境を実行するために、Amazon MWAA は環境と互換性のある特定のバージョンの Python をインストールします。そのため、起動スクリプトを使用して環境の Python バージョンを更新することはできません。

```
#!/bin/sh
```

```
echo "Updating operating system"
sudo yum update -y --exclude=python*
```

特定の Apache Airflow コンポーネントにランタイムをインストールするには、MWAA_AIRFLOW_COMPONENT、if、fi の条件文を使用します。この例では、libaio ライブラリをスケジューラとワーカーにインストールする単一のコマンドを実行しますが、ウェブサーバーにはインストールしません。

> **Important**
>
> - 「プライベートウェブサーバー」を設定した場合は、インストールタイムアウトを回避するために、次の条件を使用するか、すべてのインストールファイルをローカルで提供する必要があります。
> - 管理特権が必要な操作を実行するには、sudo を使用します。

```
#!/bin/sh

if [[ "${MWAA_AIRFLOW_COMPONENT}" != "webserver" ]]
then
     sudo yum -y install libaio
fi
```

起動スクリプトを使用して Python のバージョンを確認できます。

```
#!/bin/sh

export PYTHON_VERSION_CHECK=`python -c 'import sys; version=sys.version_info[:3];
 print("{0}.{1}.{2}".format(*version))'`
echo "Python version is $PYTHON_VERSION_CHECK"
```

Amazon MWAA はデフォルトの Python バージョンのオーバーライドをサポートしていません。これは、インストールされている Apache Airflow ライブラリとの互換性がなくなる可能性があるためです。

起動スクリプトを使用して環境変数を設定する

起動スクリプトを使用して環境変数を設定し、Apache Airflow 設定を変更します。以下は、新しい変数 ENVIRONMENT_STAGE を定義しています。この変数は DAG またはカスタムモジュールで参照できます。

```
#!/bin/sh

export ENVIRONMENT_STAGE="development"
echo "$ENVIRONMENT_STAGE"
```

起動スクリプトを使用して、一般的な Apache Airflow またはシステム変数を上書きします。例えば、LD_LIBRARY_PATH を設定して Python に対してバイナリを指定したパスで探すように指示します。これにより、「プラグイン」を使用してワークフローにカスタムバイナリを提供できます。

```
#!/bin/sh

export LD_LIBRARY_PATH=/usr/local/airflow/plugins/your-custom-binary
```

予約済み環境変数

Amazon MWAA は一連の重要な環境変数を予約します。予約された変数を上書きすると、Amazon MWAA はその変数をデフォルトに戻します。リザーブド変数は以下のとおりです。

- MWAA__AIRFLOW__COMPONENT — 次のいずれかの値で Apache Airflow コンポーネントを識別するために使用されます: scheduler、worker、または webserver。
- AIRFLOW__WEBSERVER__SECRET_KEY — Apache Airflow ウェブサーバのセッションクッキーに安全に署名するために使用されるシークレットキー。
- AIRFLOW__CORE__FERNET_KEY — メタデータデータベースに保存されている機密データ (接続パスワードなど) の暗号化と復号化に使用されるキー。
- AIRFLOW_HOME — 設定ファイルと DAG ファイルがローカルに保存されている Apache Airflow ホームディレクトリへのパス。
- AIRFLOW__CELERY__BROKER_URL — Apache Airflow スケジューラーと Celery ワーカーノード間の通信に使用されるメッセージブローカーの URL。
- AIRFLOW__CELERY__RESULT_BACKEND — Celery タスクの結果を保存するために使用されるデータベースの URL。

- AIRFLOW__CORE__EXECUTOR — Apache Airflow が使用する必要のあるエグゼキュータークラス。Amazon MWAA では、これは CeleryExecutor です
- AIRFLOW__CORE__LOAD_EXAMPLES — サンプル DAG のロードを有効化または無効化するために使用されます。
- AIRFLOW__METRICS__METRICS_BLOCK_LIST — Amazon MWAA がどのような Apache Airflow メトリクスを生成し、CloudWatch でキャプチャするかを管理するために使用されます。
- SQL_ALCHEMY_CONN — Amazon MWAA に Apache Airflow メタデータを保存するために使用される RDS for PostgreSQL データベースの接続文字列。
- AIRFLOW__CORE__SQL_ALCHEMY_CONN — SQL_ALCHEMY_CONN と同じ目的で使用されますが、新しい Apache Airflow の命名規則に従います。
- AIRFLOW__CELERY__DEFAULT_QUEUE — Apache Airflow の Celery タスクのデフォルトキュー。
- AIRFLOW__OPERATORS__DEFAULT_QUEUE — 特定の Apache Airflow オペレータを使用するタスクのデフォルトキュー。
- AIRFLOW_VERSION — Amazon MWAA 環境にインストールされている Apache Airflow バージョン。
- AIRFLOW_CONN_AWS_DEFAULT — 他の AWS サービスと統合するために使用されるデフォルトの AWS 資格情報。
- AWS_DEFAULT_REGION — デフォルトの資格情報と統合するために使用されるデフォルトの AWS リージョンを設定します。他の AWS サービスと統合します。
- AWS_REGION — 定義されている場合、この環境変数は環境変数 AWS_DEFAULT_REGION およびプロファイル設定のリージョンの値を上書きします。
- PYTHONUNBUFFERED —コンテナログに stdout および stderr のストリームを送信するために使用されます。
- AIRFLOW__METRICS__STATSD_ALLOW_LIST — リストの要素で始まるメトリクスを送信するための、コンマで区切られたプレフィックスの許可リストを設定するために使用されます。
- AIRFLOW__METRICS__STATSD_ON — StatsD へのメトリクスの送信を有効にします。
- AIRFLOW__METRICS__STATSD_HOST — StatSD デーモンへの接続に使用されます。
- AIRFLOW__METRICS__STATSD_PORT — StatSD デーモンへの接続に使用されます。
- AIRFLOW__METRICS__STATSD_PREFIX — StatSD デーモンへの接続に使用されます。
- AIRFLOW__CELERY__WORKER_AUTOSCALE — 同時実行数の最大値と最小値を設定します。
- AIRFLOW__CORE__DAG_CONCURRENCY — 1 つの DAG でスケジューラーが同時に実行できるタスクインスタンスの数を設定します。

- AIRFLOW__CORE__MAX_ACTIVE_TASKS_PER_DAG — DAG あたりのアクティブタスクの最大数を設定します。
- AIRFLOW__CORE__PARALLELISM — 同時に実行できるタスクインスタンスの最大数を定義します。
- AIRFLOW__SCHEDULER__PARSING_PROCESSES — DAG をスケジュールするためにスケジューラーが解析するプロセスの最大数を設定します。
- AIRFLOW__CELERY_BROKER_TRANSPORT_OPTIONS__VISIBILITY_TIMEOUT — メッセージが別のワーカーに再配信されるまでに、ワーカーがタスクの承認を待つ秒数を定義します。
- AIRFLOW__CELERY_BROKER_TRANSPORT_OPTIONS__REGION — 基礎となる Celery トランスポートの AWS リージョンを設定します。
- AIRFLOW__CELERY_BROKER_TRANSPORT_OPTIONS__PREDEFINED_QUEUES — 基盤となる Celery トランスポートのキューを設定します。
- AIRFLOW_SCHEDULER_ALLOWED_RUN_ID_PATTERN— DAG をトリガーする際の run_id パラメータの入力の妥当性を確認するために使用されます。
- AIRFLOW__WEBSERVER__BASE_URL — Apache Airflow UI をホストするために使用されるウェブサーバーの URL。

予約されていない環境変数

起動スクリプトを使用して、予約されていない環境変数を上書きできます。以下に、これらの一般的な変数のいくつかを列挙します。

- PATH — オペレーティングシステムが実行ファイルとスクリプトを検索するディレクトリのリストを指定します。コマンドがコマンドラインで実行されると、システムはコマンドを見つけて実行するために PATH のディレクトリを順番にチェックします。Apache Airflow でカスタムオペレータやタスクを作成する場合、外部スクリプトや実行可能ファイルに依存することがあるかもしれません。これらのファイルを含むディレクトリが PATH の変数で指定されていない場合、システムはそれらを見つけることができないため、タスクは実行に失敗します。適切なディレクトリを PATH に追加することで、Apache Airflow タスクは必要な実行可能ファイルを見つけて実行できるようになります。
- PYTHONPATH — Python インタープリターが、インポートされたモジュールやパッケージを検索するディレクトリを決定するために使用されます。デフォルトの検索パスに追加できるディレクトリのリストです。これにより、インタプリタは標準ライブラリに含まれていない、またはシステムディレクトリにインストールされていない Python ライブラリを見つけて読み込むことができ

ます。この変数を使用してモジュールとカスタム Python パッケージを追加し、DAG で使用します。

- LD_LIBRARY_PATH — Linux の動的リンカーとローダーが共有ライブラリを検索してロードするために使用する環境変数。共有ライブラリを含むディレクトリのリストを指定し、デフォルトのシステムライブラリディレクトリの前に検索します。この変数を使用してカスタムバイナリを指定します。
- CLASSPATH — Java ランタイム環境 (JRE) と Java 開発キット (JDK) によって使用され、実行時に Java クラス、ライブラリ、リソースを検索してロードします。コンパイルされた Java コードを含むディレクトリ、JAR ファイル、および ZIP アーカイブのリストです。

Amazon MWAA での DAG の取り扱い

Amazon Managed Workflows for Apache Airflow 環境でダイレクト非循環グラフ (DAG) を実行するには、環境に接続されている Amazon S3 ストレージバケットにファイルをコピーし、Amazon MWAA コンソール上の DAG とサポートファイルの場所を Amazon MWAA に知らせます。Amazon MWAA は、ワーカー、スケジューラー、ウェブサーバー間で DAG を同期します。このガイドでは、Amazon MWAA 環境に DAG を追加または更新する方法、およびカスタムプラグインと Python 依存関係をインストールする方法について説明します。

トピック

Amazon S3 バケットの概要

Amazon MWAA 環境の Amazon S3 バケットには、パブリックアクセスがブロックされている必要があります。デフォルトでは、Amazon S3 のリソースバケット、オブジェクト、関連サブリソース (例: 設定や 設定) などのすべてのリソースはプライベートです。

- リソース所有者 (バケットを作成した AWS アカウント) のみが、リソースにアクセスできます。リソースの所有者 (管理者など) は、アクセス制御ポリシーを作成することにより、他のユーザーにアクセス許可を付与することもできます。
- 設定するアクセスポリシーには、Amazon S3 バケットに DAG、`requirements.txt` のカスタムプラグイン、および `plugins.zip` のPython 依存関係を追加する権限が必要です。必要なアクセス権限を含むポリシーの例については、「AmazonMWAAFullConsoleAccess」を参照してください。

Amazon MWAA 環境の Amazon S3 バケットでは、バージョニング管理が有効になっている必要があります。Amazon S3 バケットのバージョニング管理が有効になっている場合、新しいバージョンが作成されるたびに、新しいコピーが作成されます。

- バージョニングは、Amazon S3 バケットの `plugins.zip` のカスタムプラグインと `requirements.txt` の Python 依存関係に対して有効になっています。
- Amazon S3 バケットでこれらのファイルが更新されるたびに、Amazon MWAA コンソールで、`plugins.zip` と `requirements.txt` のバージョンを指定する必要があります。

DAG の追加と更新

有向非巡回グラフ (DAG) は、DAG の構造をコードとして定義する Python ファイル内で定義されます。AWS CLI および Amazon S3 コンソールを使用して、DAG をご利用の環境にアップロードできます。このページでは、Amazon S3 バケット内の `dags` フォルダを使用して、Amazon Managed Workflows for Apache Airflow 環境に Apache Airflow DAG を追加または更新する手順について説明します。

セクション

前提条件

このページのステップを完了するには、以下のものが必要です。

- 権限 — AWS アカウントには、管理者から、ご使用の環境の「AmazonMWAAFullConsoleAccess」アクセスコントロールポリシーへのアクセス権限が付与されている必要があります。さらに、Amazon MWAA 環境には、その環境で使用される AWS のリソースへのアクセスを「実行ロール」で許可されている必要があります。
- アクセス — 依存関係をウェブサーバーに直接インストールするためにパブリックリポジトリにアクセスする必要がある場合は、パブリックネットワークのウェブサーバーアクセスが環境に設定されている必要があります。詳細については、「the section called “Apache Airflow のアクセスモード”」を参照してください。

- Amazon S3 設定 — `plugins.zip` で DAG、カスタムプラグイン、および `requirements.txt` で Python の依存関係を保存するために使用される「Amazon S3 バケット」は、Public Access Blocked と Versioning Enabledで構成する必要があります。

使用方法

有向非巡回グラフ (DAG) は、DAG の構造をコードとして定義する単一の Python ファイル内で定義されます。その構成は以下の通りである：

- 「DAG」 の定義。
- DAG の実行方法と実行する「タスク」 を説明する「オペレーター」。
- タスクを実行する順序を記述する「オペレータ関係」。

Amazon MWAA 環境で Apache Airflow プラットフォームを実行するには、DAG 定義をストレージバケット内の `dags` フォルダにコピーする必要があります。たとえば、ストレージバケット内の DAG フォルダは以下のようになっている場合があります。

Example DAG フォルダー

```
dags/
# dag_def.py
```

Amazon MWAA は、Amazon S3 バケットの新規オブジェクトと変更されたオブジェクトを 30 秒ごとに Amazon MWAA スケジューラとワーカーコンテナの `/usr/local/airflow/dags` フォルダに自動的に同期し、ファイルタイプに関係なく Amazon S3 ソースのファイル階層を維持します。新しい DAG が Apache Airflow UI に表示されるまでの時間は、`scheduler.dag_dir_list_interval` によって制御されます。既存の DAG への変更は、次の「DAG 処理ループ」 で取り込まれます。

> **Note**
>
> DAG フォルダーには `airflow.cfg` 設定ファイルを含める必要はありません。Amazon MWAA コンソールからデフォルトの Apache Airflow 設定を上書きできます。詳細については、「Amazon MWAA での Apache Airflow 構成オプションの使用」を参照してください。

v2 での変更点

- 新規:オペレータ、フック、エグゼキューター。Apache Airflow v1と Apache Airflow v2 の間で、Amazon MWAA の `plugins.zip` に指定する DAG 内のインポート文やカスタムプラグインが変更されています。例えば、Apache Airflow v1の `from airflow.contrib.hooks.aws_hook import AwsHook` は、Apache Airflow v2 では `from airflow.providers.amazon.aws.hooks.base_aws import AwsBaseHook` に変更されています。詳細については、Apache Airflow リファレンスガイドの「Python API リファレンス」 を参照してください。

Amazon MWAA CLI ユーティリティを使用した DAG のテスト

- コマンドラインインターフェイス (CLI) ユーティリティは、Amazon Managed Workflows for Apache Airflow (Amazon MWAA) 環境をローカルに複製します。
- CLI は、Amazon MWAA のプロダクションイメージに似た Docker コンテナイメージをローカルでビルドします。これにより、Amazon MWAA にデプロイする前に、ローカルの Apache Airflow 環境を実行して DAG、カスタムプラグイン、依存関係を開発およびテストできます。
- CLI を実行するには、GitHub の「aws-mwaa-local-runner」 を参照してください。

Amazon S3 への DAG コードのアップロード

Amazon S3 コンソールまたは AWS Command Line Interface (AWS CLI) を使用して、Amazon S3 バケットに DAG コードをアップロードできます。以下の手順では、コード (`.py`)をAmazon S3バケット内の `dags` という名前のフォルダーにアップロードすることを前提としています。

AWS CLI を使用する場合

AWS Command Line Interface (AWS CLI) は、コマンドラインシェルでコマンドを使用して AWS サービスとやり取りするためのオープンソースツールです。このページのステップを完了するには、以下のものが必要です。

- 「AWS CLI - バージョン 2 のインストール」
- 「AWS CLI - `aws configure` によるクイック設定」

AWS CLI を使用してアップロードするには

1. 以下のコマンドを使って、Amazon S3 バケットをすべてリストアップします

```
aws s3 ls
```

2. 以下のコマンドを使用して、ご使用の環境の Amazon S3 バケット内のファイルとフォルダを一覧表示します。

```
aws s3 ls s3://YOUR_S3_BUCKET_NAME
```

3. 以下のコマンドは、dag_def.py ファイルを dags フォルダにアップロードします。

```
aws s3 cp dag_def.py s3://YOUR_S3_BUCKET_NAME/dags/
```

Amazon S3 バケットに dags という名前のフォルダがまだ存在しない場合、このコマンドは dags フォルダを作成し、新しいフォルダに名前が dag_def.py のファイルをアップロードします。

Amazon S3 コンソールの使用

Amazon S3 コンソールは、Amazon S3 バケット内のリソースを作成および管理できるウェブベースのユーザーインターフェイスです。以下の手順では、DAGs フォルダーが dags という名前であると仮定しています。

Amazon S3 コンソールを使ってアップロードするには

1. Amazon MWAA コンソールで「環境ページ」 を開きます。
2. 環境を選択します。
3. Amazon S3コンソールの「DAG コード in S3」ペインでDAG コード内のS3バケット リンクを選択して、ストレージバケットを開きます。
4. dags フォルダを選択します。
5. [アップロード] を選択します。
6. [ファイルの追加] を選択します。
7. dag_def.py のローカルコピーを選択し、[アップロード] を選択します。

Amazon MWAA コンソールで DAG フォルダへのパスを指定する (初回)

以下の手順では、dags という名前の Amazon S3 バケット上のフォルダへのパスを指定していると仮定します。

1. Amazon MWAA コンソールで「環境ページ」を開きます。
2. DAG を実行する環境を選択します。
3. [Edit] (編集) を選択します。
4. [Amazon S3 ペインの DAG コード] で、[DAG フォルダ] のフィールドの横にある [S3 を参照] を選択します。
5. dags フォルダを選択します。
6. [選択] を選択します。
7. [次へ]、[環境の更新] を選択します。

Apache Airflow UI での変更の表示

Apache Airflow へのログイン

Apache Airflow UI を表示するためには、AWS Identity and Access Management (IAM) で AWS アカウントに対して Apache Airflow UI アクセスポリシー: AmazonMWAAWebServer アクセス のアクセス許可が必要です。

Apache Airflow UI にアクセスするには

1. Amazon MWAA コンソールで「環境ページ」を開きます。
2. 環境を選択します。
3. [Airflow UI を開く] を選択します。

次のステップ

- GitHub の「aws-mwaa-local-runner」を使用して、DAG、カスタムプラグイン、Python の依存関係をローカルでテストします。

カスタムプラグインのインストール

Amazon Managed Workflows for Apache Airflow は Apache Airflow の組み込みプラグインマネージャーをサポートしているため、カスタム Apache Airflow オペレータ、フック、センサー、またはインターフェイスを使用できます。このページでは、`plugins.zip` ファイルを使用して、Amazon MWAA 環境に Apache Airflow カスタムプラグインをインストールする手順について説明します。

目次

前提条件

このページのステップを完了するには、以下のものが必要です。

- 権限 — AWS アカウントには、管理者から、ご使用の環境の「AmazonMWAAFullConsoleAccess」アクセスコントロールポリシーへのアクセス権限が付与されている必要があります。さらに、Amazon MWAA 環境には、その環境で使用される AWS のリソースへのアクセスを「実行ロール」で許可されている必要があります。
- アクセス — 依存関係をウェブサーバーに直接インストールするためにパブリックリポジトリにアクセスする必要がある場合は、パブリックネットワークのウェブサーバーアクセスが環境に設定されている必要があります。詳細については、「the section called “Apache Airflow のアクセスモード”」を参照してください。
- Amazon S3 設定 — `plugins.zip` で DAG、カスタムプラグイン、および `requirements.txt` で Python の依存関係を保存するために使用される「Amazon S3 バケット」は、Public Access Blocked と Versioning Enabledで構成する必要があります。

使用方法

カスタムプラグインを環境で実行するには、次の 3 つのことを行う必要があります。

1. `plugins.zip` ファイルをローカルに作成します。
2. `plugins.zip` のファイルを Amazon S3 のバケットにアップロードしてください。
3. Amazon MWAA コンソールの [プラグインファイル] フィールドに、このファイルのバージョンを指定します。

> **Note**
>
> これが初めて `plugins.zip` を Amazon S3 バケットにアップロードする場合、Amazon MWAA コンソールでファイルへのパスも指定する必要があります。1 回だけこのステップを行ってください。

v2 での変更点

- 新規:オペレータ、フック、エグゼキューター。Apache Airflow v1と Apache Airflow v2 の間で、Amazon MWAA の `plugins.zip` に指定する DAG 内のインポート文やカスタムプラグインが変更されています。例えば、Apache Airflow v1の `from airflow.contrib.hooks.aws_hook import AwsHook` は、Apache Airflow v2 では `from airflow.providers.amazon.aws.hooks.base_aws import AwsBaseHook` に変更されて

います。詳細については、Apache Airflow リファレンスガイドの「Python API リファレンス」を参照してください。

- 新規:プラグインへのインポート。`airflow.{operators,sensors,hooks}.<plugin_name>`を使用して追加されたプラグインのオペレータ、センサー、フックをインポートすることはもはやサポートされていません。これらの拡張モジュールは、通常の Python モジュールとしてインポートする必要があります。v2 以降では、これらを DAG ディレクトリに配置し、.airflowignore ファイルを作成して使用して DAG としての解析から除外する方法が推奨されます。詳細については、Apache Airflow リファレンスガイドの「モジュール管理」と「カスタムオペレータの作成」を参照してください。

カスタムプラグイン数

Apache Airflow の組み込みプラグインマネージャは、単にファイルを`$AIRFLOW_HOME/plugins`フォルダにドロップすることで外部の機能をコアに統合できます。これにより、カスタムの Apache Airflow オペレータ、フック、センサー、またはインターフェースを使用できます。次のセクションでは、ローカル開発環境におけるフラットでネストされたディレクトリ構造の例と、plugins.zip 内のディレクトリ構造を決定する import ステートメントの例を示します。

カスタムプラグインのディレクトリとサイズの制限

Apache Airflow スケジューラとワーカーは、AWS で管理されるFargateコンテナが `/usr/local/airflow/plugins/*` での環境でスタートアップする際に、カスタムプラグインを探します。

- ディレクトリ構造。(/*での) ディレクトリ構造は、`plugins.zip` ファイルの内容に基づいています。例えば、`plugins.zip` に `operators` ディレクトリがトップレベルディレクトリとして含まれている場合、そのディレクトリは環境の `/usr/local/airflow/plugins/operators` に抽出されます。
- サイズ制限。1 GB 未満の `plugins.zip` ファイルをお勧めします。`plugins.zip` ファイルのサイズが大きいほど、環境でのスタートアップ時間が長くなります。Amazon MWAA は `plugins.zip` ファイルのサイズを明示的に制限していませんが、10 分以内に依存関係をインストールできない場合、Fargate サービスはタイムアウトし、環境を安定した状態にロールバックしようとします。

> **Note**
>
> Apache Airflow v1.10.12 または Apache Airflow v2.0.2 を使用する環境では、Amazon MWAA は Apache Airflow ウェブサーバー上のアウトバウンドトラフィックを制限し、プラグイ

ンや Python の依存関係をウェブサーバーに直接インストールすることを許可していません。Apache Airflow v2.2.2 以降、Amazon MWAA はプラグインと依存関係をウェブサーバーに直接インストールできるようになりました。

カスタムプラグイン数

次のセクションでは、Apache Airflow リファレンスガイドのサンプルコードが 使用して、ローカル開発環境を構築する方法を示します。

plugins.zip でフラットなディレクトリ構造を使用する例

Apache Airflow v2

次の例は、Apache Airflow v2向けのフラットなディレクトリ構造を持つ `plugins.zip` ファイルを示しています。

Example Python VirtualEnvOperator plugins.zip を含むフラットディレクトリ

次の例では、Apache Airflow Python VirtualEnv オペレータ用のカスタムプラグインを作成する の PythonVirtualEnvOperator カスタムプラグインの plugins.zip ファイルの最上位ツリーを示します。

```
### virtual_python_plugin.py
```

Example plugins/virtual_python_plugin.py

以下の例は、Python VirtualEnvOperator カスタムプラグインを示しています。

```
"""
Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the "Software"), to deal in
the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of
the Software, and to permit persons to whom the Software is furnished to do so.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS
FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
```

```
COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER
IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
"""
from airflow.plugins_manager import AirflowPlugin
import airflow.utils.python_virtualenv
from typing import List

def _generate_virtualenv_cmd(tmp_dir: str, python_bin: str, system_site_packages:
 bool) -> List[str]:
    cmd = ['python3','/usr/local/airflow/.local/lib/python3.7/site-packages/
virtualenv', tmp_dir]
    if system_site_packages:
        cmd.append('--system-site-packages')
    if python_bin is not None:
        cmd.append(f'--python={python_bin}')
    return cmd

airflow.utils.python_virtualenv._generate_virtualenv_cmd=_generate_virtualenv_cmd

class VirtualPythonPlugin(AirflowPlugin):
    name = 'virtual_python_plugin'
```

Apache Airflow v1

次の例は、Apache Airflow v1 用のフラットなディレクトリ構造の `plugins.zip` ファイルを示しています。

Example Python VirtualEnvOperator plugins.zip を含むフラットディレクトリ

次の例では、Apache Airflow Python VirtualEnv オペレータ用のカスタムプラグインを作成する の PythonVirtualEnvOperator カスタムプラグインの plugins.zip ファイルの最上位ツリーを示します。

```
### virtual_python_plugin.py
```

Example plugins/virtual_python_plugin.py

以下の例は、Python VirtualEnvOperator カスタムプラグインを示しています。

```
from airflow.plugins_manager import AirflowPlugin
from airflow.operators.python_operator import PythonVirtualenvOperator
```

```
def _generate_virtualenv_cmd(self, tmp_dir):
    cmd = ['python3','/usr/local/airflow/.local/lib/python3.7/site-packages/
virtualenv', tmp_dir]
    if self.system_site_packages:
        cmd.append('--system-site-packages')
    if self.python_version is not None:
        cmd.append('--python=python{}'.format(self.python_version))
    return cmd
PythonVirtualenvOperator._generate_virtualenv_cmd=_generate_virtualenv_cmd

class EnvVarPlugin(AirflowPlugin):
    name = 'virtual_python_plugin'
```

plugins.zip のネストされたディレクトリ構造を使用する例

Apache Airflow v2

次の例は、Apache Airflow v2向けの hooks、operators、および sensors ディレクトリのためにそれぞれ異なるディレクトリを持つ plugins.zip ファイルを示しています。

Example plugins.zip

```
__init__.py
my_airflow_plugin.py
hooks/
|-- __init__.py
|-- my_airflow_hook.py
operators/
|-- __init__.py
|-- my_airflow_operator.py
|-- hello_operator.py
sensors/
|-- __init__.py
|-- my_airflow_sensor.py
```

次の例は、カスタムプラグインを使用する DAG (DAGs フォルダー) のインポートステートメントを示しています。

Example dags/your_dag.py

```
from airflow import DAG
```

```
from datetime import datetime, timedelta
from operators.my_airflow_operator import MyOperator
from sensors.my_airflow_sensor import MySensor
from operators.hello_operator import HelloOperator

default_args = {
 'owner': 'airflow',
 'depends_on_past': False,
 'start_date': datetime(2018, 1, 1),
 'email_on_failure': False,
 'email_on_retry': False,
 'retries': 1,
 'retry_delay': timedelta(minutes=5),
}


with DAG('customdag',
   max_active_runs=3,
   schedule_interval='@once',
   default_args=default_args) as dag:

 sens = MySensor(
  task_id='taskA'
 )

 op = MyOperator(
  task_id='taskB',
  my_field='some text'
 )

 hello_task = HelloOperator(task_id='sample-task', name='foo_bar')



 sens >> op >> hello_task
```

Example plugins/my_airflow_plugin.py

```
from airflow.plugins_manager import AirflowPlugin
from hooks.my_airflow_hook import *
from operators.my_airflow_operator import *

class PluginName(AirflowPlugin):
```

```
    name = 'my_airflow_plugin'

    hooks = [MyHook]
    operators = [MyOperator]
    sensors = [MySensor]
```

次の例は、カスタムプラグインファイルに必要な各インポートステートメントを示しています。

Example hooks/my_airflow_hook.py

```
from airflow.hooks.base import BaseHook


class MyHook(BaseHook):

    def my_method(self):
        print("Hello World")
```

Example sensors/my_airflow_sensor.py

```
from airflow.sensors.base import BaseSensorOperator
from airflow.utils.decorators import apply_defaults


class MySensor(BaseSensorOperator):

    @apply_defaults
    def __init__(self,
                 *args,
                 **kwargs):
        super(MySensor, self).__init__(*args, **kwargs)

    def poke(self, context):
        return True
```

Example operators/my_airflow_operator.py

```
from airflow.operators.bash import BaseOperator
from airflow.utils.decorators import apply_defaults
from hooks.my_airflow_hook import MyHook
```

```
class MyOperator(BaseOperator):

    @apply_defaults
    def __init__(self,
                 my_field,
                 *args,
                 **kwargs):
        super(MyOperator, self).__init__(*args, **kwargs)
        self.my_field = my_field

    def execute(self, context):
        hook = MyHook('my_conn')
        hook.my_method()
```

Example operators/hello_operator.py

```
from airflow.models.baseoperator import BaseOperator
from airflow.utils.decorators import apply_defaults

class HelloOperator(BaseOperator):

    @apply_defaults
    def __init__(
            self,
            name: str,
            **kwargs) -> None:
        super().__init__(**kwargs)
        self.name = name

    def execute(self, context):
        message = "Hello {}".format(self.name)
        print(message)
        return message
```

「Amazon MWAA CLIユーティリティを使用したカスタムプラグインのテスト」の手順に従い、次に「,plugins.zipファイルの作成」でpluginsディレクトリ内のコンテンツを圧縮してください。例えば、cd plugins です。

Apache Airflow v1

次の例は、Apache Airflow v1.10.12 向けの hooks、operators、および sensors ディレクトリのためにそれぞれ異なるディレクトリを持つ plugins.zip ファイルを示しています。

Example plugins.zip

```
__init__.py
my_airflow_plugin.py
hooks/
    |-- __init__.py
    |-- my_airflow_hook.py
operators/
    |-- __init__.py
    |-- my_airflow_operator.py
    |-- hello_operator.py
sensors/
    |-- __init__.py
    |-- my_airflow_sensor.py
```

次の例は、カスタムプラグインを使用する DAG (DAGs フォルダー) のインポートステートメントを示しています。

Example dags/your_dag.py

```
from airflow import DAG
from datetime import datetime, timedelta
from operators.my_operator import MyOperator
from sensors.my_sensor import MySensor
from operators.hello_operator import HelloOperator

default_args = {
 'owner': 'airflow',
 'depends_on_past': False,
 'start_date': datetime(2018, 1, 1),
 'email_on_failure': False,
 'email_on_retry': False,
 'retries': 1,
 'retry_delay': timedelta(minutes=5),
}


with DAG('customdag',
   max_active_runs=3,
   schedule_interval='@once',
   default_args=default_args) as dag:

 sens = MySensor(
```

```
 task_id='taskA'
)

op = MyOperator(
 task_id='taskB',
 my_field='some text'
)

hello_task = HelloOperator(task_id='sample-task', name='foo_bar')



sens >> op >> hello_task
```

Example plugins/my_airflow_plugin.py

```
from airflow.plugins_manager import AirflowPlugin
from hooks.my_airflow_hook import *
from operators.my_airflow_operator import *
from utils.my_utils import *

class PluginName(AirflowPlugin):

    name = 'my_airflow_plugin'

    hooks = [MyHook]
    operators = [MyOperator]
    sensors = [MySensor]
```

次の例は、カスタムプラグインファイルに必要な各インポートステートメントを示しています。

Example hooks/my_airflow_hook.py

```
from airflow.hooks.base_hook import BaseHook


class MyHook(BaseHook):

    def my_method(self):
        print("Hello World")
```

Example sensors/my_airflow_sensor.py

```
from airflow.sensors.base_sensor_operator import BaseSensorOperator
from airflow.utils.decorators import apply_defaults


class MySensor(BaseSensorOperator):

    @apply_defaults
    def __init__(self,
                 *args,
                 **kwargs):
        super(MySensor, self).__init__(*args, **kwargs)

    def poke(self, context):
        return True
```

Example operators/my_airflow_operator.py

```
from airflow.operators.bash_operator import BaseOperator
from airflow.utils.decorators import apply_defaults
from hooks.my_hook import MyHook


class MyOperator(BaseOperator):

    @apply_defaults
    def __init__(self,
                 my_field,
                 *args,
                 **kwargs):
        super(MyOperator, self).__init__(*args, **kwargs)
        self.my_field = my_field

    def execute(self, context):
        hook = MyHook('my_conn')
        hook.my_method()
```

Example operators/hello_operator.py

```
from airflow.models.baseoperator import BaseOperator
from airflow.utils.decorators import apply_defaults
```

```
class HelloOperator(BaseOperator):

    @apply_defaults
    def __init__(
            self,
            name: str,
            **kwargs) -> None:
        super().__init__(**kwargs)
        self.name = name

    def execute(self, context):
        message = "Hello {}".format(self.name)
        print(message)
        return message
```

「Amazon MWAA CLIユーティリティを使用したカスタムプラグインのテスト」の手順に従い、次に「,plugins.zipファイルの作成」でpluginsディレクトリ内のコンテンツを圧縮してください。例えば、cd plugins です。

plugins.zip ファイルの作成

以下の手順では、plugins.zip ファイルをローカルで作成する場合に推奨される手順について説明します。

ステップ 1: Amazon MWAA CLI ユーティリティを使用してカスタムプラグインをテストする

- コマンドラインインターフェイス (CLI) ユーティリティは、Amazon Managed Workflows for Apache Airflow (Amazon MWAA) 環境をローカルに複製します。
- CLI は、Amazon MWAA のプロダクションイメージに似た Docker コンテナイメージをローカルでビルドします。これにより、Amazon MWAA にデプロイする前に、ローカルの Apache Airflow 環境を実行して DAG、カスタムプラグイン、依存関係を開発およびテストできます。
- CLI を実行するには、GitHub の「aws-mwaa-local-runner」を参照してください。

ステップ 2: plugins.zip ファイルを作成する

ビルトインの ZIP アーカイブユーティリティやその他の ZIP ユーティリティ (7zip など) を使用して.zip ファイルを作成できます。

> **Note**
>
> Windows OS 用のビルトイン ZIP ユーティリティは、.zip ファイルの作成時にサブフォルダを追加する場合があります。Amazon S3 バケットにアップロードする前に plugins.zip ファイルの内容を確認して、ディレクトリが追加されていないことを確認することをお勧めします。

1. ディレクトリをローカルの Airflow プラグインディレクトリに変更します。例:

```
myproject$ cd plugins
```

2. 次のコマンドを実行して、コンテンツに実行権限があることを確認します (macOS と Linux のみ)。

```
plugins$ chmod -R 755 .
```

3. pluginsフォルダ内のコンテンツを ZIP 圧縮します。

```
plugins$ zip -r plugins.zip .
```

plugins.zip を Amazon S3 にアップロードします。

Amazon S3 コンソールまたは AWS Command Line Interface (AWS CLI) を使用して、Amazon S3 バケットにplugins.zipファイルをアップロードできます。

AWS CLI を使用する場合

AWS Command Line Interface (AWS CLI) は、コマンドラインシェルでコマンドを使用して AWS サービスとやり取りするためのオープンソースツールです。このページのステップを完了するには、以下のものが必要です。

- 「AWS CLI - バージョン 2 のインストール」
- 「AWS CLI - aws configure によるクイック設定」

AWS CLI を使用してアップロードするには

1. コマンドプロンプトで、plugins.zip ファイルが保存されているディレクトリに移動します。例:

```
cd plugins
```

2. 以下のコマンドを使って、Amazon S3 バケットをすべてリストアップします

```
aws s3 ls
```

3. 以下のコマンドを使用して、ご使用の環境の Amazon S3 バケット内のファイルとフォルダを一覧表示します。

```
aws s3 ls s3://YOUR_S3_BUCKET_NAME
```

4. 環境の Amazon S3 バケットに plugins.zip ファイルをアップロードするには、次のコマンドを使用します。

```
aws s3 cp plugins.zip s3://YOUR_S3_BUCKET_NAME/plugins.zip
```

Amazon S3 コンソールの使用

Amazon S3 コンソールは、Amazon S3 バケット内のリソースを作成および管理できるウェブベースのユーザーインターフェイスです。

Amazon S3 コンソールを使ってアップロードするには

1. Amazon MWAA コンソールで「環境ページ」を開きます。
2. 環境を選択します。
3. Amazon S3コンソールの「DAG コード in S3」ペインでDAG コード内のS3バケット リンクを選択して、ストレージバケットを開きます。
4. 「アップロード」を選択します。
5. 「ファイルの追加」を選択します。
6. plugins.zip のローカルコピーを選択し、[アップロード] を選択します。

環境へのカスタムプラグインのインストール

このセクションでは、plugins.zip ファイルへのパスを指定し、zip ファイルが更新されるたびに plugins.zip ファイルのバージョンを指定することで、Amazon S3 バケットにアップロードしたカスタムプラグインをインストールする方法について説明します。

Amazon MWAA コンソールで **plugins.zip** へのパスを指定する（初回）

これが初めて `plugins.zip` を Amazon S3 バケットにアップロードする場合、Amazon MWAA コンソールでファイルへのパスも指定する必要があります。1 回だけこのステップを行ってください。

1. Amazon MWAA コンソールで「環境ページ」を開きます。
2. 環境を選択します。
3. [Edit] (編集) を選択します。
4. 「Amazon S3 の DAG コード」ペインで、「プラグインファイル - オプション」フィールドの横にある「S3 を参照」を選択します。
5. Amazon S3 バケットの「`plugins.zip`」ファイルを選択します。
6. [選択] を選択します。
7. 「次へ」→「環境の更新」を選択します。

Amazon MWAA コンソールで**plugins.zip**のバージョンを指定する。

新しいバージョンの `plugins.zip` を Amazon S3 バケットにアップロードするたびに、Amazon MWAA コンソールで `plugins.zip` ファイルのバージョンを指定する必要があります。

1. Amazon MWAA コンソールで「環境ページ」を開きます。
2. 環境を選択します。
3. [Edit] (編集) を選択します。
4. 「Amazon S3 の DAG コードペイン」で、ドロップダウンリストから `plugins.zip` のバージョンを選択します。
5. [Next] (次へ) を選択します。

plugins.zip のユースケースの例

- Apache Hive と Hadoop を使ったカスタムプラグイン でカスタムプラグインを作成する方法について学びます。

- Python 仮想環境オペレータにパッチを適用するカスタムプラグイン でカスタムプラグインを作成する方法について学びます。
- Oracle によるカスタムプラグイン でカスタムプラグインを作成する方法について学びます。
- the section called “DAG のタイムゾーンの変更” でカスタムプラグインを作成する方法について学びます。

次のステップ

- GitHub の aws-mwaa-local-runner を使用して、DAG、カスタムプラグイン、Python の依存関係をローカルでテストします。

Python 依存関係のインストール

Python 依存関係とは、Amazon Managed Workflows for Apache Airflow 環境の Apache Airflow バージョンの Apache Airflow ベースインストールに含まれていないパッケージまたはディストリビューションです。このページでは、Amazon S3 バケット内の`requirements.txt`ファイルを使用して Apache Airflow Python の依存関係を Amazon MWAA 環境にインストールする手順について説明します。

目次

前提条件

このページのステップを完了するには、以下のものが必要です。

- アクセス許可 - AWS アカウントには、管理者から環境の AmazonMWAAFullConsole アクセスコントロールポリシーへのアクセス権が付与されている必要があります。さらに、Amazon MWAA 環境が使用する AWS リソースにアクセスするには、実行ロールで Amazon MWAA 環境を許可する必要があります。
- アクセス — 依存関係をウェブサーバーに直接インストールするためにパブリックリポジトリにアクセスする必要がある場合は、パブリックネットワークのウェブサーバーアクセスが環境に設定されている必要があります。詳細については、「the section called “Apache Airflow のアクセスモード”」を参照してください。
- Amazon S3 設定 — `plugins.zip` で DAG、カスタムプラグイン、および `requirements.txt` で Python の依存関係を保存するために使用される「Amazon S3 バケット」は、Public Access Blocked と Versioning Enabledで構成する必要があります。

仕組み

Amazon MWAA では、Python の依存関係はすべて、`requirements.txt` ファイルを Amazon S3 バケットにアップロードし、ファイルを更新するたびにAmazon MWAA コンソールでファイルのバージョンを指定することでインストールされます。Amazon MWAA は `pip3 install -r requirements.txt` を実行して、Python の依存関係を Apache Airflowスケジューラおよび各ワーカーにインストールします。

Python の依存関係をお使いの環境で実行するには、次の 3 つのことを行う必要があります。

1. `requirements.txt` ファイルをローカルに作成します。
2. ローカルの `requirements.txt` を Amazon S3 バケットにアップロードします。
3. Amazon MWAA コンソールの [要件ファイル] フィールドに、このファイルのバージョンを指定します。

> **Note**
>
> これが初めて `requirements.txt` を作成してAmazon S3 バケットにアップロードする場合、Amazon MWAA コンソールでファイルへのパスも指定する必要があります。1 回だけこのステップを行ってください。

Python の依存関係の概要

Apache Airflow のエクストラやその他の Python 依存関係は、Python パッケージインデックス (PyPi.org)、Python wheels (`.whl`)、またはプライベート PyPi/PEP-503 準拠リポジトリでホストされている Python 依存関係から環境にインストールできます。

Python 依存関係の場所とサイズ制限

Apache Airflow スケジューラとワーカーは、 の環境の AWSマネージド Fargate コンテナで起動時にカスタムプラグインを探します`/usr/local/airflow/plugins`。

- サイズ制限。1 GB未満の合計サイズを持つライブラリを参照する `requirements.txt` ファイルをお勧めします。Amazon MWAA がインストールする必要のあるライブラリが多いほど、環境でのスタートアップ時間が長くなります。Amazon MWAA はインストールするライブラリのサイズを明示的に制限していませんが、10 分以内に依存関係をインストールできない場合、Fargate サービスはタイムアウトし、環境を安定した状態にロールバックしようとします。

requirements.txt ファイルの作成

以下の手順では、requirements.txt ファイルをローカルで作成する場合に推奨される手順について説明します。

ステップ 1: Amazon MWAA CLI ユーティリティを使用して Python の依存関係をテストする

- コマンドラインインターフェイス (CLI) ユーティリティは、Amazon Managed Workflows for Apache Airflow (Amazon MWAA) 環境をローカルに複製します。
- CLI は、Amazon MWAA のプロダクションイメージに似た Docker コンテナイメージをローカルでビルドします。これにより、Amazon MWAA にデプロイする前に、ローカルの Apache Airflow 環境を実行して DAG、カスタムプラグイン、依存関係を開発およびテストできます。

- CLI を実行するには、 の aws-mwaa-local-runner を参照してください GitHub。

ステップ 2: `requirements.txt` を作成する

次のセクションでは、`requirements.txt` ファイルの「Python Package インデックス」から Python 依存関係を指定する方法について説明します。

Apache Airflow v2

1. ローカルでテストします。`requirements.txt` ファイルを作成する前に、ライブラリを繰り返し追加してパッケージとバージョンの適切な組み合わせを見つけてください。Amazon MWAA CLI ユーティリティを実行するには、「」の「aws-mwaa-local-runner」を参照してください GitHub。
2. Apache Airflow パッケージのエクストラを確認してください。Amazon MWAA で Apache Airflow v2 にインストールされているパッケージのリストを表示するには、 GitHub ウェブサイトの「Amazon MWAA local runnerrequirements.txt」を参照してください。
3. 制約ステートメントを追加します。Apache Airflow v2 環境用の制約ファイルを `requirements.txt` ファイルの先頭に追加します。Apache Airflow の制約ファイルには、Apache Airflow のリリース時点で利用可能なプロバイダーのバージョンが指定されています。

   Apache Airflow v2.7.2 から、要件ファイルには `--constraint` ステートメントを含める必要があります。制約を指定しない場合、要件に記載されているパッケージが使用している Apache Airflow のバージョンと互換性があることを確認するため、Amazon MWAA はお客様に代わって制約を指定します。

   次の例では、*`{environment-version}`* を環境のバージョン番号に置き換え、*`{Python-version}`* を環境と互換性のある Python のバージョンに置き換えます。

   Apache Airflow 環境と互換性のある Python のバージョンについては、「Apache Airflow バージョン」を参照してください。

   ```
   --constraint "https://raw.githubusercontent.com/apache/airflow/
   constraints-{Airflow-version}/constraints-{Python-version}.txt"
   ```

   制約ファイルが `xyz==1.0` パッケージが環境内の他のパッケージと互換性がないと判断した場合、`pip3 install` は環境に互換性のないライブラリがインストールされるのを防ぐために失敗しま す。パッケージのインストールに失敗した場合、各 Apache Airflow コンポー

ネント (スケジューラ、ワーカー、ウェブサーバー) のエラーログを、対応するログストリームで CloudWatch Logs に表示できます。ログタイプの詳細については、「the section called "Airflow ログの表示"」を参照してください。

4. Apache Airflow パッケージ。パッケージエクストラとバージョン (==)を追加します。これにより、同じ名前で異なるバージョンのパッケージが環境にインストールされるのを防ぐことができます。

```
apache-airflow[package-extra]==2.5.1
```

5. Python ライブラリ パッケージ名とバージョン (==) を requirements.txt ファイルに追加します。これにより、PyPi.org からの将来の重大な更新が自動的に適用されないようにできます。

```
library == version
```

Example Boto3 と psycopg2-binary

この例は、デモンストレーションのみを目的としています。boto と psycopg2 のバイナリライブラリは Apache Airflow v2 のベースインストールに含まれており、requirements.txt ファイルで指定する必要はありません。

```
boto3==1.17.54
boto==2.49.0
botocore==1.20.54
psycopg2-binary==2.8.6
```

バージョンなしでパッケージが指定されている場合、Amazon MWAA は PyPi.org から最新バージョンのパッケージをインストールします。このバージョンは、お客様の requirements.txt 内の他のパッケージと競合する可能性があります。

Apache Airflow v1

1. ローカルでテストします。requirements.txt ファイルを作成する前に、ライブラリを繰り返し追加してパッケージとバージョンの適切な組み合わせを見つけてください。Amazon MWAA CLI ユーティリティを実行するには、「」の「aws-mwaa-local-runner」を参照してください GitHub。

2. Airflow パッケージのエクストラを確認してください。https://raw.githubusercontent.com/apache/airflow/constraints-1.10.12/constraints-3.7.txt で Apache Airflow v1.10.12 で利用できるパッケージのリストが確認してください。
3. 制約ファイルを追加します。Apache Airflow v1.10.12 の制約ファイルを `requirements.txt` ファイルの先頭に追加します。制約ファイルが `xyz==1.0` パッケージが環境内の他のパッケージと互換性がないと判断した場合、`pip3 install` は環境に互換性のないライブラリがインストールされるのを防ぐために失敗しま す。

```
--constraint "https://raw.githubusercontent.com/apache/airflow/
constraints-1.10.12/constraints-3.7.txt"
```

4. アパッチ・エアフロー v1.10.12 パッケージ。「Airflow パッケージエクストラ」と Apache Airflow v1.10.12 バージョン (==) を追加します。これにより、同じ名前で異なるバージョンのパッケージが環境にインストールされるのを防ぐことができます。

```
apache-airflow[package]==1.10.12
```

Example Secure Shell (SSH)

次の例では、 `requirements.txt` ファイルは、Apache Airflow v1.10.12 用 SSH をインストールします。

```
apache-airflow[ssh]==1.10.12
```

5. Python ライブラリ パッケージ名とバージョン (==) を `requirements.txt` ファイルに追加します。これにより、PyPi.org からの将来の重大な更新が自動的に適用されないようにできます。

```
library == version
```

Example Boto3

次の例では、`requirements.txt` ファイルは、Apache Airflow v1.10.12 用の Boto3 ライブラリをインストールします。

```
boto3 == 1.17.4
```

バージョンなしでパッケージが指定されている場合、Amazon MWAA は PyPi.org から最新バージョンのパッケージをインストールします。このバージョンは、お客様の `requirements.txt` 内の他のパッケージと競合する可能性があります。

`requirements.txt` を Amazon S3 にアップロードします。

Amazon S3 コンソールまたは AWS Command Line Interface（ AWS CLI) を使用して、Amazon S3 バケットに`requirements.txt`ファイルをアップロードできます。

の使用 AWS CLI

AWS Command Line Interface（ AWS CLI) は、コマンドラインシェルのコマンドを使用して AWS サービスとやり取りできるオープンソースツールです。このページのステップを完了するには、以下のものが必要です。

- AWS CLI – バージョン 2 をインストールします。
- AWS CLI – を使用したクイック設定`aws configure`。

を使用してアップロードするには AWS CLI

1. 以下のコマンドを使って、Amazon S3 バケットをすべてリストアップします

```
aws s3 ls
```

2. 以下のコマンドを使用して、ご使用の環境の Amazon S3 バケット内のファイルとフォルダを一覧表示します。

```
aws s3 ls s3://YOUR_S3_BUCKET_NAME
```

3. 以下のコマンドにより、`requirements.txt` ファイルが Amazon S3 にアップロードされます。

```
aws s3 cp requirements.txt s3://YOUR_S3_BUCKET_NAME/requirements.txt
```

Amazon S3 コンソールの使用

Amazon S3 コンソールは、Amazon S3 バケット内のリソースを作成および管理できるウェブベースのユーザーインターフェイスです。

Amazon S3 コンソールを使ってアップロードするには

1. Amazon MWAA コンソールで「環境ページ」を開きます。
2. 環境を選択します。
3. Amazon S3コンソールの「DAG コード in S3」ペインでDAG コード内のS3バケット リンクを選択して、ストレージバケットを開きます。
4. 「アップロード」を選択します。
5. 「ファイルの追加」を選択します。
6. `requirements.txt` のローカルコピーを選択し、[アップロード] を選択します。

環境への Python 依存関係のインストール

このセクションでは、requirements.txt ファイルへのパスを指定し、更新されるたびに requirements.txt ファイルのバージョンを指定することで、Amazon S3 バケットにアップロードした依存関係をインストールする方法について説明します。

Amazon MWAA コンソールで **`requirements.txt`** へのパスを指定する（初回）

これが初めて `requirements.txt` を作成してAmazon S3 バケットにアップロードする場合、Amazon MWAA コンソールでファイルへのパスも指定する必要があります。1 回だけこのステップを行ってください。

1. Amazon MWAA コンソールで「環境ページ」を開きます。
2. 環境を選択します。
3. [編集] を選択します。
4. Amazon S3 ペインの DAG コードで、[要件ファイル-オプション] フィールドの横にある [S3 を参照] を選択します。
5. Amazon S3 バケット上の`requirements.txt`ファイルを選択します。
6. [選択] を選択します。
7. 「次へ」→「環境の更新」を選択します。

環境の更新が完了した直後に、新しいパッケージの使用を開始できます。

Amazon MWAA コンソールで**requirements.txt**のバージョンを指定する。

新しいバージョンの requirements.txt を Amazon S3 バケットにアップロードするたびに、Amazon MWAA コンソールで requirements.txt ファイルのバージョンを指定する必要があります。

1. Amazon MWAA コンソールで「環境ページ」を開きます。
2. 環境を選択します。
3. [編集] を選択します。
4. Amazon S3 の DAG コードペインで、ドロップダウンリストから requirements.txt のバージョンを選択します。
5. 「次へ」→「環境の更新」を選択します。

環境の更新が完了した直後に、新しいパッケージの使用を開始できます。

requirements.txtの口グを表示する

ワークフローのスケジュール設定と dags フォルダーの解析を行うスケジューラーの Apache Airflow ログを表示できます。次の手順では、Amazon MWAA コンソールでスケジューラのロググループを開き、Logs コンソールで Apache Airflow CloudWatch ログを表示する方法について説明します。

requirements.txt のログを表示するには

1. Amazon MWAA コンソールで「環境ページ」を開きます。
2. 環境を選択します。
3. モニタリングペインで Airflow スケジューラーロググループを選択します。
4. [ログストリーム] requirements_install_ipのログを選択します。
5. /usr/local/airflow/.local/bin で環境にインストールされたパッケージのリストが表示されるはずです。例:

```
Collecting appdirs==1.4.4 (from -r /usr/local/airflow/.local/bin (line 1))
Downloading https://files.pythonhosted.org/
packages/3b/00/2344469e2084fb28kjdsfiuyweb47389789vxbmnbjhsdgf5463acd6cf5e3db69324/
appdirs-1.4.4-py2.py3-none-any.whl
Collecting astroid==2.4.2 (from -r /usr/local/airflow/.local/bin (line 2))
```

6. パッケージのリストを確認し、インストール中にエラーが発生したパッケージがないか確認してください。何か問題が発生した場合、以下のようなエラーが表示されることがあります。

```
2021-03-05T14:34:42.731-07:00
No matching distribution found for LibraryName==1.0.0 (from -r /usr/local/
airflow/.local/bin (line 4))
No matching distribution found for LibraryName==1.0.0 (from -r /usr/local/
airflow/.local/bin (line 4))
```

次のステップ

- で DAGs 、カスタムプラグイン、Python の依存関係をローカルでテストします GitHub。

Amazon S3 でファイルの削除

このページでは、Amazon Managed Workflows for Apache Airflow 環境の Amazon S3 バケットでのバージョニングの仕組みと、DAG、`plugins.zip` または `requirements.txt` ファイルを削除する手順について説明します。

目次

前提条件

このページのステップを完了するには、以下のものが必要です。

- 権限 — AWS アカウントには、管理者から、ご使用の環境の「AmazonMWAAFullConsoleAccess」 アクセスコントロールポリシーへのアクセス権限が付与されている必要があります。さらに、Amazon MWAA 環境には、その環境で使用される AWS のリソースへのアクセスを「実行ロール」で許可されている必要があります。
- アクセス — 依存関係をウェブサーバーに直接インストールするためにパブリックリポジトリにアクセスする必要がある場合は、パブリックネットワークのウェブサーバーアクセスが環境に設定されている必要があります。詳細については、「the section called “Apache Airflow のアクセスモード”」を参照してください。
- Amazon S3 設定 — `plugins.zip` で DAG、カスタムプラグイン、および `requirements.txt` で Python の依存関係を保存するために使用される「Amazon S3 バケット」は、Public Access Blocked と Versioning Enabledで構成する必要があります。

バージョン管理の概要

Amazon S3 バケット内の `requirements.txt` および `plugins.zip` はバージョニングされています。オブジェクトの Amazon S3 バケットバージョニングが有効になっていて、アーティファクト (plugins.zip など) が Amazon S3 バケットから削除されても、ファイルは完全には削除されません。Amazon S3 でアーティファクトが削除されるたびに、「I'm not here (ここにはいません)」という404 (オブジェクトが見つかりません) エラー/0k ファイルのファイルの新しいコピーが作成されます。Amazon S3 ではこれを「削除マーカー」と呼んでいます。削除マーカーは、他のオブジェクトと同じように、キー名 (またはキー) とバージョンIDを持つファイルの「ヌル」バージョンです。

Amazon S3 バケットのストレージコストを削減するために、ファイルバージョンと削除マーカーを定期的に削除することをお勧めします。「最新でない」(以前の) ファイルバージョンを完全に削除するには、ファイルのバージョンを削除し、そのバージョンの削除マーカーを削除する必要があります。

使用方法

Amazon MWAA は 30 秒ごとにお客様の Amazon S3 バケットに対して同期オペレーションを実行します。これにより、Amazon S3 バケット内のすべての DAG 削除が Fargate コンテナの Airflow イメージと同期されます。

plugins.zip および requirements.txt ファイルについては、Amazon MWAA がカスタムプラグインと Python 依存関係を使用して Fargate コンテナの新しい Airflow イメージをビルドするときに、環境の更新後にのみ変更が行われます。もし requirements.txt または plugins.zip のファイルの現在のバージョンを削除し、削除されたファイルに新しいバージョンを提供せずに環境を更新した場合、更新は「Unable to read version {version} of file {file}」などのエラーメッセージと共に失敗します。

Amazon S3 で DAG を削除する

DAG ファイル (.py) はバージョン管理されていないため、Amazon S3 コンソールで直接削除できます。次のステップでは、Amazon S3 バケット内の DAG を削除する方法について説明します。

DAG を削除するには

1. Amazon MWAA コンソールで「環境ページ」を開きます。
2. 環境を選択します。
3. Amazon S3コンソールの「DAG コード in S3」ペインでDAG コード内のS3バケット リンクを選択して、ストレージバケットを開きます。
4. dags フォルダを選択します。
5. DAG を選択し、[削除] します。
6. [オブジェクトを削除] する場合、delete を入力してください。
7. [Delete objects] (オブジェクトの削除) を選択します。

> **Note**
>
> Apache Airflow は DAG の実行履歴を保存します。Apache Airflow で DAG を実行した後は、Apache Airflow で削除するまで、ファイルのステータスに関係なく、Airflow DAG リストに残ります。Apache Airflow 内の DAG を削除するには、リンク列の下にある赤い「削除」ボタンを選択します。

「現在の」requirements.txt または plugins.zip を環境から削除する

現在、plugins.zip や requirements.txt を追加した後に環境から削除する方法はありませんが、現在対応中です。当面の回避策は、それぞれ空のテキストまたは zip ファイルを指すことです。

「最新でない」(以前の) requirements.txt または plugins.zip バージョンを削除する

Amazon S3 バケット内の `requirements.txt` および `plugins.zip` ファイルは、Amazon MWAA でバージョニングされています。Amazon S3 バケット上のこれらのファイルを完全に削除する場合は、オブジェクトの現在のバージョン (121212) (plugins.zip など) を取得し、バージョンを削除してから、ファイルバージョンの削除マーカーを削除する必要があります。

Amazon S3 コンソールで「最新でない」(以前の) ファイルバージョンを削除することもできますが、それでも以下のオプションのいずれかを使用して削除マーカーを削除する必要があります。

- オブジェクトバージョンを取得するには、Amazon S3 ガイドの「バージョニングが有効なバケットからのオブジェクトバージョンの取得」を参照してください。
- オブジェクトバージョンを削除するには、Amazon S3 ガイドの「バージョニングが有効なバケットからのオブジェクトバージョンの削除」を参照してください。
- 削除マーカーを削除するには、Amazon S3 ガイドの「削除マーカーの管理」を参照してください。

ライフサイクルを使用して「最新ではない」(以前の) バージョンを削除し、マーカーを自動的に削除する

Amazon S3 バケットのライフサイクルポリシーを設定して、Amazon S3 バケット内の「最新でない」(以前の) バージョンの plugins.zip ファイルと requirements.txt ファイルを特定の日数が経過した後に削除するか、期限切れのオブジェクトの削除マーカーを削除できます。

1. Amazon MWAA コンソールで「環境ページ」を開きます。
2. 環境を選択します。
3. [Amazon S3 内の DAG コード] で、Amazon S3 バケットを選択します。
4. [ライフサイクルルールを作成] を選択します。

requirements.txt の「最新でない」バージョンを削除してマーカーを自動的に削除するライフサイクルポリシーの例

次の例は、「最新ではない」バージョンの requirements.txt ファイルとその削除マーカーを 30 日後に完全に削除するライフサイクルルールを作成する方法を示しています。

1. Amazon MWAA コンソールで「環境ページ」を開きます。
2. 環境を選択します。
3. [Amazon S3 内の DAG コード] で、Amazon S3 バケットを選択します。
4. [ライフサイクルルールを作成] を選択します。
5. [ライフサイクルルール] の名前に `Delete previous requirements.txt versions and delete markers after thirty days` を入力してください。
6. [プレフィックス] では、[要件]。
7. [ライフサイクルルールアクション] で、[以前のバージョンのオブジェクトを完全に削除する] と [期限切れの削除マーカーまたは不完全なマルチパートアップロードを削除] を選択します。
8. [オブジェクトが前のバージョンになるまでの日数] に、「`30`」を入力してください。
9. [期限切れオブジェクト削除マーカー] で [期限切れオブジェクト削除マーカーを削除] を選択します。オブジェクトは 30 日後に完全に削除されます。

次のステップ

- Amazon S3 削除マーカーの詳細については、「削除マーカーの管理」を参照してください。
- Amazon S3 ライフサイクルの詳細については、「期限切れオブジェクト」を参照してください。

ネットワーク

このガイドでは、Amazon MWAA 環境に必要な Amazon VPC ネットワーク設定について説明します。

セクション

Amazon MWAA でのネットワーキングについて

Amazon VPC は、 AWS アカウントにリンクされた仮想ネットワークです。仮想インフラストラクチャとネットワークトラフィックのセグメンテーションをきめ細かく制御できるため、クラウドセキュリティと動的なスケーリングが可能になります。このページでは、Apache Airflow 環境の Amazon マネージドワークフローをサポートするために必要な、パブリックルーティングまたはプライベートルーティングを使用する Amazon VPC インフラストラクチャについて説明します。

目次

規約

パブリックルーティング

インターネットにアクセスできる Amazon VPC ネットワーク。

プライベート・ルーティング

インターネットにアクセスできない Amazon VPC ネットワーク。

サポート対象

次の表では、Amazon MWAA がサポートする Amazon VPC の種類について説明しています。

Amazon VPC タイプ	サポート	
環境を作成しようとしているアカウントが所有する Amazon VPC。	はい	
複数の AWS アカウントが AWS リソースを作成する共有 Amazon VPC。	はい	

VPC インフラストラクチャの概要

Amazon MWAA 環境を作成すると、Amazon MWAA は、お客様が環境に合わせて選択した Apache Airflow アクセスモードに基づいて、お客様の環境用に 1 つから 2 つの VPC エンドポイントを作成します。これらのエンドポイントは、Amazon VPC 内の Elastic Network Interfaces (ENI) とプライベート IP アドレスで表されます。これらのエンドポイントが作成されると、これらの IPs 宛てのトラフィックは、環境で使用される対応する AWS サービスにプライベートまたはパブリックにルーティングされます。

次のセクションでは、トラフィックをインターネット経由でパブリックに、または Amazon VPC 内でプライベートにルーティングするために必要な Amazon VPC インフラストラクチャについて説明します。

インターネット経由のパブリックルーティング

このセクションでは、パブリックルーティングを使用する環境の Amazon VPC インフラストラクチャについて説明します。次の VPC インフラストラクチャが必要です。

- 「1 つのVPCセキュリティグループ」。VPCセキュリティ・グループは仮想ファイアウォールとして機能し、インスタンスのイングレス (受信) とイグレス (送信) のネットワーク・トラフィックを制御します。
  - 最大 5 つのセキュリティグループを指定できます。
  - セキュリティグループは、自己参照型のインバウンドルールをセキュリティグループ自身に指定する必要があります。
  - セキュリティグループはすべてのトラフィックのアウトバウンドルール (`0.0.0.0/0`) を指定する必要があります。
  - セキュリティグループは、自己参照ルールに含まれるすべてのトラフィックを許可する必要があります。例えば (推奨) 全アクセス自己参照型セキュリティグループの例 です。
  - セキュリティグループは、オプションでトラフィックをさらに制限することができ、HTTPS ポートレンジ443とTCPポートレンジ5432を指定してポート範囲を設定できます。例えば、(オプション)ポート5432 へのインバウンド・アクセスを制限するセキュリティグループの例 と (オプション)ポート443へのインバウンド・アクセスを制限するセキュリティ・グループの例 です。
- 2 つのパブリックサブネット。パブリックサブネットは、インターネットゲートウェイへのルートが含まれているルートテーブルに関連付けられているサブネットです。
  - 2 つのパブリックサブネットが必要です。これにより、Amazon MWAA は、一方のコンテナに障害が発生した場合に、もう一方のアベイラビリティーゾーンでお客様の環境用の新しいコンテナイメージを構築できます。
  - サブネットは異なるアベイラビリティゾーンになければなりません。例えば、`us-east-1a` や `us-east-1b` などです。
  - サブネットは Elastic IP アドレス (EIP) を使用して NAT ゲートウェイ (または NAT インスタンス) にルーティングする必要があります。
  - サブネットは、インターネット行きのトラフィックをインターネットゲートウェイに誘導するルートテーブルを持つ必要があります。
- 2 つのプライベートサブネット。プライベート・サブネットとは、インターネットゲートウェイへのルートを持つルート・テーブルと関連付けられて「いない」サブネットのことです。

  - 2 つのプライベートサブネットが必要です。これにより、Amazon MWAA は、一方のコンテナに障害が発生した場合に、もう一方のアベイラビリティーゾーンでお客様の環境用の新しいコンテナイメージを構築できます。
  - サブネットは異なるアベイラビリティゾーンになければなりません。例えば、`us-east-1a` や `us-east-1b` などです。
  - サブネットには NAT デバイス (ゲートウェイまたはインスタンス)へのルートテーブルが 必要です。
  - サブネットは、インターネットゲートウェイにルーティングしては「ならない」です。
- 「ネットワークアクセス制御リスト (ACL) 」。NACL は、サブネットレベルでインバウンドトラフィックとアウトバウンドトラフィックを (許可または拒否ルールを使用して) 管理します。
  - NACL には、すべてのトラフィックを許可するインバウンドルール (`0.0.0.0/0`) が必要です。
  - NACL には、すべてのトラフィックを拒否するアウトバウンドルール (`0.0.0.0/0`) が必要です。
  - 例えば (推奨) ACL の例 です。
- 2 つの NAT ゲートウェイ (または NAT インスタンス)。NAT デバイスは、プライベートサブネットのインスタンスからインターネットまたは他の AWS サービスにトラフィックを転送し、応答をインスタンスにルーティングします。
  - NAT デバイスはパブリックサブネットに接続する必要があります。(パブリックサブネットごとに 1 つの NAT デバイス)
  - NAT デバイスには、各パブリックサブネットに Elastic IPv4 アドレス (EIP) がアタッチされている必要があります。
- 「一つのインターネットゲートウェイ」。インターネットゲートウェイは、Amazon VPC をインターネットおよびその他の AWS サービスに接続します。
  - インターネットゲートウェイが Amazon VPC に接続されていなければならなりません。

インターネットにアクセスできないプライベートルーティング

このセクションでは、プライベートルーティングを使用する環境の Amazon VPC インフラストラクチャについて説明します。次の VPC インフラストラクチャが必要です。

- 「1 つのVPCセキュリティグループ」。VPCセキュリティ・グループは仮想ファイアウォールとして機能し、インスタンスのイングレス (受信) とイグレス (送信) のネットワーク・トラフィックを制御します。
  - 最大 5 つのセキュリティグループを指定できます。

  - セキュリティグループは、自己参照型のインバウンドルールをセキュリティグループ自身に指定する必要があります。
  - セキュリティグループはすべてのトラフィックのアウトバウンドルール (0.0.0.0/0) を指定する必要があります。
  - セキュリティグループは、自己参照ルールに含まれるすべてのトラフィックを許可する必要があります。例えば (推奨) 全アクセス自己参照型セキュリティグループの例 です。
  - セキュリティグループは、オプションでトラフィックをさらに制限することができ、HTTPS ポートレンジ443とTCPポートレンジ5432を指定してポート範囲を設定できます。例えば、(オプション)ポート5432 へのインバウンド・アクセスを制限するセキュリティグループの例 と (オプション)ポート443へのインバウンド・アクセスを制限するセキュリティ・グループの例 です。
- 2 つのプライベートサブネット。プライベート・サブネットとは、インターネットゲートウェイへのルートを持つルート・テーブルと関連付けられて「いない」サブネットのことです。
  - 2 つのプライベートサブネットが必要です。これにより、Amazon MWAA は、一方のコンテナに障害が発生した場合に、もう一方のアベイラビリティーゾーンでお客様の環境用の新しいコンテナイメージを構築できます。
  - サブネットは異なるアベイラビリティゾーンになければなりません。例えば、us-east-1a や us-east-1b などです。
  - サブネットには VPC エンドポイントへのルートテーブルが必要です。
  - サブネットには NAT デバイス (ゲートウェイまたはインスタンス) へのルートテーブルやインターネットゲートウェイがあってはなりません。
- 「ネットワークアクセス制御リスト (ACL) 」。NACL は、サブネットレベルでインバウンドトラフィックとアウトバウンドトラフィックを (許可または拒否ルールを使用して) 管理します。
  - NACL には、すべてのトラフィックを許可するインバウンドルール (0.0.0.0/0) が必要です。
  - NACL には、すべてのトラフィックを拒否するアウトバウンドルール (0.0.0.0/0) が必要です。
  - 例えば (推奨) ACL の例 です。
- ローカルルートテーブル。ローカル・ルート・テーブルは、VPC 内の通信のためのデフォルト・ルートです。
  - ローカルルートテーブルはプライベートサブネットに関連付けられている必要があります。
  - ローカルルートテーブルを使用して、VPC 内のインスタンスが自ネットワークと通信できるようにする必要があります。例えば、 を使用して Apache Airflow ウェブサーバー の VPC イン

ターフェイスエンドポイント AWS Client VPN にアクセスする場合、ルートテーブルは VPC エンドポイントにルーティングする必要があります。

- 環境で使用される各 AWS サービスの VPC エンドポイント、および Amazon MWAA 環境と同じ AWS リージョンと Amazon VPC 内の Apache Airflow VPC エンドポイント。
  - 環境で使用される各 AWS サービスの VPC エンドポイントと Apache Airflow の VPC エンドポイント。例えば (必須) VPC エンドポイント です。
  - VPC エンドポイントではプライベート DNS が有効になっている必要があります。
  - VPC エンドポイントは、環境の 2 つのプライベートサブネットに関連付けられている必要があります。
  - VPC エンドポイントは、環境のセキュリティグループに関連付けられている必要があります。
  - 各エンドポイントの VPC エンドポイントポリシーは、環境で使用される AWS サービスへのアクセスを許可するように設定する必要があります。例えば (推奨) すべてのアクセスを許可する VPC エンドポイントポリシーの例 です。
  - Amazon S3 用の VPC エンドポイントポリシーは、バケットアクセスを許可するように設定する必要があります。例えば (推奨) バケットアクセスを許可する Amazon S3 ゲートウェイエンドポイントポリシーの例 です。

Amazon VPC と Apache エアフローアクセスモードのユースケース例

このセクションでは、Amazon VPC でのネットワークアクセスのさまざまなユースケースと、Amazon MWAA コンソールで選択すべき Apache Airflow ウェブサーバーアクセスモードについて説明します。

インターネットアクセスが許可されている-新しい Amazon VPC ネットワーク

VPC 内のインターネットアクセスが組織で許可されており、ユーザーがインターネット経由で Apache Airflow ウェブサーバーにアクセスできるようにしたい場合:

1. インターネットにアクセスできる Amazon VPC ネットワークを作成します。
2. Apache Airflow ウェブサーバー用のパブリックネットワークアクセスモードの環境を作成します。
3. 推奨される内容: Amazon VPC インフラストラクチャ、Amazon S3 バケット、Amazon MWAA 環境を同時に作成する AWS CloudFormation クイックスタートテンプレートを使用することをお勧めします。詳細については、「Amazon Managed Workflows for Apache Airflow のクイックスタートチュートリアル」を参照してください。

VPC 内のインターネットアクセスが組織で許可されており、Apache Airflow ウェブサーバーへのアクセスを VPC 内のユーザーに制限したい場合:

1. インターネットにアクセスできる Amazon VPC ネットワークを作成します。
2. Apache Airflow ウェブサーバーの VPC インターフェースエンドポイントにコンピューターからアクセスするメカニズムを作成します。
3. Apache Airflow ウェブサーバー用のプライベートネットワークアクセスモードの環境を作成します。
4. 推奨事項:

    a. の Amazon MWAA コンソールオプション 1: Amazon MWAA コンソールで VPC ネットワークを作成する、または の AWS CloudFormation テンプレートを使用することをお勧めしますオプション 2: インターネットにアクセス可能な Amazon VPC ネットワークの作成。

    b. を使用して、 の Apache Airflow ウェブサーバー AWS Client VPN へのアクセスを設定することをお勧めしますチュートリアル: AWS Client VPN を使用したプライベートネットワークアクセスの設定。

インターネットアクセスは許可されていません-新しい Amazon VPC ネットワーク

VPC 内のインターネットアクセスが組織によって許可されていない場合:

1. インターネットにアクセスせずに Amazon VPC ネットワークを作成します。
2. Apache Airflow ウェブサーバーの VPC インターフェースエンドポイントにコンピューターからアクセスするメカニズムを作成します。
3. 環境で使用される AWS サービスごとに VPC エンドポイントを作成します。
4. Apache Airflow ウェブサーバー用のプライベートネットワークアクセスモードの環境を作成します。
5. 推奨事項:

    a. AWS CloudFormation テンプレートを使用して、インターネットアクセスのない Amazon VPC と、 で Amazon MWAA が使用する各 AWS サービスの VPC エンドポイントを作成することをお勧めしますオプション 3: インターネットにアクセスせずに Amazon VPC ネットワークを作成する。

    b. を使用して、 の Apache Airflow ウェブサーバー AWS Client VPN へのアクセスを設定することをお勧めしますチュートリアル: AWS Client VPN を使用したプライベートネットワークアクセスの設定。

インターネットアクセスは許可されていません-既存の Amazon VPC ネットワーク

もし組織があなたのVPCでのインターネットアクセスを許可していない場合、かつ既に必要な Amazon VPCネットワークがインターネットアクセスなしに設定されている場合:

1. 環境で使用される AWS サービスごとに VPC エンドポイントを作成します。
2. Apache エアフロー用の VPC エンドポイントを作成します。
3. Apache Airflow ウェブサーバーの VPC インターフェースエンドポイントにコンピューターからアクセスするメカニズムを作成します。
4. Apache Airflow ウェブサーバー用のプライベートネットワークアクセスモードの環境を作成します。
5. 推奨事項:

    a. Amazon MWAA で使用される各 AWS サービスに必要な VPC エンドポイントと、 の Apache Airflow に必要な VPC エンドポイントを作成してアタッチすることをお勧めしますAmazon VPC に必要な VPC サービスエンドポイントをプライベートルーティングで作成する。

    b. を使用して、 の Apache Airflow ウェブサーバー AWS Client VPN へのアクセスを設定することをお勧めしますチュートリアル: AWS Client VPN を使用したプライベートネットワークアクセスの設定。

Amazon MWAA の VPC のセキュリティ

このページでは、Apache Airflow 用 Amazon マネージドワークフロー環境の保護に使用される Amazon VPC コンポーネントと、これらのコンポーネントに必要な設定について説明します。

目次

  - (オプション)ポート443へのインバウンド・アクセスを制限するセキュリティ・グループの例
- VPC エンドポイントポリシー (プライベートルーティングのみ)
  - (推奨) すべてのアクセスを許可する VPC エンドポイントポリシーの例
  - (推奨) バケットアクセスを許可する Amazon S3 ゲートウェイエンドポイントポリシーの例

用語

パブリックルーティング

インターネットにアクセスできる Amazon VPC ネットワーク。

プライベート・ルーティング

インターネットにアクセスできない Amazon VPC ネットワーク。

セキュリティの概要

セキュリティグループとアクセスコントロールリスト (ACL) は、指定されたルールを使用して Amazon VPC のサブネットとインスタンス全体のネットワークトラフィックを制御する方法を提供します。

- サブネットとの間のネットワークトラフィックは、アクセス制御リスト(ACL)によって制御することができます。必要な ACL は 1 つだけで、同じ ACL を複数の環境で使用できます。
- インスタンスに出入りするネットワークトラフィックは、Amazon VPC セキュリティグループによって制御できます。　1 つの環境につき 1 ~ 5 つのセキュリティグループを使用できます。
- インスタンスとの間で送受信されるネットワークトラフィックは、VPC エンドポイントポリシーでも制御できます。組織がAmazon VPC 内のインターネットアクセスを許可しておらず、プライベートルーティングで Amazon VPC ネットワークを使用している場合は、「AWS VPC エンドポイントと Apache Airflow VPC エンドポイント」には VPC エンドポイントポリシーが必要です。

ネットワークアクセスコントロールリスト (ACL)

「ネットワークアクセスコントロールリスト (ACL)」は、サブネットレベルでインバウンドまたはアウトバウンドのトラフィックを許可または拒否ルールで管理できます。ACL はステートレスです。つまり、インバウンドルールとアウトバウンドルールは、別々に明示的に指定する必要があります。VPC ネットワーク内のインスタンスに出入りできるネットワークトラフィックの種類を指定するために使用されます。

すべての Amazon VPC には、インバウンドトラフィックとアウトバウンドトラフィックを許可するデフォルト ACL があります。デフォルト ACL ルールを編集することも、カスタム ACL を作成してサブネットにアタッチすることもできます。サブネットには常に 1 つの ACL しかアタッチできませんが、1 つの ACL を複数のサブネットにアタッチできます。

(推奨) ACL の例

次の例は、パブリックルーティングまたはプライベートルーティングを使用する Amazon VPC の Amazon VPC で使用できるインバウンドおよびアウトバウンド ACL ルールを示しています。

ルール番号	タイプ	プロトコル	ポート範囲	送信元	許可/拒否
100	すべての IPv4 トラフィック	すべて	すべて	0.0.0.0/0	許可
*	すべての IPv4 トラフィック	すべて	すべて	0.0.0.0/0	拒否

VPC セキュリティグループ

「VPC セキュリティ・グループ」は、インスタンスレベルでネットワーク・トラフィックを制御する仮想ファイアウォールの役割を果たします。セキュリティグループはステートフルです。つまり、インバウンド接続が許可されると、リプレイが許可されます。VPC ネットワーク内のインスタンスから許可されるネットワークトラフィックのタイプを指定するために使用されます。

すべての Amazon VPC にはデフォルトのセキュリティグループがあります。デフォルトでは、インバウンドルールはありません。すべてのアウトバウンドトラフィックを許可するアウトバウンドルールがあります。デフォルトのセキュリティグループルールを編集するか、カスタムセキュリティグループを作成して Amazon VPC にアタッチできます。Amazon MWAA では、NAT ゲートウェイにトラフィックを誘導するインバウンドルールとアウトバウンドルールを設定する必要があります。

(推奨) 全アクセス自己参照型セキュリティグループの例

以下は、パブリックルーティングまたはプライベートルーティングを持つAmazon VPCのすべてのトラフィックを許可するインバウンドセキュリティグループルールを示しています。この例のセキュリティグループは、自己参照規則です。

タイプ	プロトコル	ソースタイプ	ソース		
すべてのトラフィック	すべて	すべて	sg-0909e8e81919 / my-mwaa-vpc-security-group		

次の例は、アウトバウンドセキュリティグループのルールを示しています。

タイプ	プロトコル	ソースタイプ	ソース		
すべてのトラフィック	すべて	すべて	0.0.0.0/0		

(オプション)ポート5432 へのインバウンド・アクセスを制限するセキュリティグループの例

次の例は、お客様の環境の Amazon Aurora PostgreSQL メタデータデータベース (Amazon MWAA が所有) のポート 5432 でのすべての HTTPS トラフィックを許可するインバウンドセキュリティグループのルールを示しています。

> **Note**
>
> このルールを使用してトラフィックを制限する場合は、ポート 443 で TCP トラフィックを許可するルールをもう 1 つ追加する必要があります。

タイプ	プロトコル	ポート範囲	ソースタイプ	ソース	
カスタム TCP	TCP	5432	カスタム	sg-0909e8e81919 / my-mwaa-vpc-security-group	

(オプション)ポート443へのインバウンド・アクセスを制限するセキュリティ・グループの例

次の例は、Apache Airflow Web サーバーのポート 443 上のすべての TCP トラフィックを許可するインバウンドセキュリティグループのルールを示しています。

タイプ	プロトコル	ポート範囲	ソースタイプ	ソース	
HTTPS	TCP	443	カスタム	sg-0909e8e81919 / my-mwaa-vpc-security-group	

VPC エンドポイントポリシー (プライベートルーティングのみ)

VPC エンドポイント (AWS PrivateLink) ポリシーは、プライベートサブネットからの AWS サービスへのアクセスを制御します。VPC エンドポイントポリシーは、VPC ゲートウェイまたはインターフェイスのエンドポイントにアタッチする IAM リソースポリシーです。このセクションでは、各 VPC エンドポイントの VPC エンドポイントポリシーに必要な権限について説明します。

すべての AWS サービスへのフルアクセスを許可する VPC エンドポイントごとに VPC インターフェイスエンドポイントポリシーを使用し、アクセス AWS 許可のためにのみ実行ロールを使用することをお勧めします。

(推奨) すべてのアクセスを許可する VPC エンドポイントポリシーの例

次の例は、プライベートルーティングを使用する Amazon VPC の VPC インターフェイスエンドポイントポリシーを示しています。

```
{
    "Statement": [
        {
            "Action": "*",
            "Effect": "Allow",
            "Resource": "*",
            "Principal": "*"
        }
    ]
```

```
}
```

(推奨) バケットアクセスを許可する Amazon S3 ゲートウェイエンドポイントポリシーの例

以下の例は、プライベートルーティングを使用する Amazon VPC の Amazon ECR オペレーションに必要な Amazon S3 バケットへのアクセスを提供する VPC ゲートウェイエンドポイントポリシーを示しています。DAG とサポートファイルが保存されているバケットに加えて、Amazon ECR イメージを取得にも必要です。

```
{
  "Statement": [
    {
      "Sid": "Access-to-specific-bucket-only",
      "Principal": "*",
      "Action": [
        "s3:GetObject"
      ],
      "Effect": "Allow",
      "Resource": ["arn:aws:s3:::prod-region-starport-layer-bucket/*"]
    }
  ]
}
```

Amazon MWAA のサービス固有の Amazon VPC エンドポイントへのアクセスの管理

VPC エンドポイント (AWS PrivateLink) を使用すると、インターネットゲートウェイ、NAT デバイス、VPN、またはファイアウォールプロキシを必要と AWS せずに、VPC を でホストされているサービスにプライベートに接続できます。これらのエンドポイントは、VPC 内のインスタンスと AWS サービス間の通信を可能にする、水平方向にスケーラブルで可用性の高い仮想デバイスです。このページでは、Amazon MWAA によって作成された VPC エンドポイントと、Apache Airflow 用の Amazon マネージドワークフローでプライベートネットワークアクセスモードを選択した場合に Apache Airflow ウェブサーバーの VPC エンドポイントにアクセスする方法について説明します。

目次

- パブリックネットワークアクセスモード
- プライベートネットワークアクセスモード
- 他の AWS サービスを使用するためのアクセス許可
- VPC エンドポイントの表示
  - Amazon VPC コンソールで VPC エンドポイントを表示する
  - Apache Airflow ウェブサーバーとその VPC エンドポイントのプライベート IP アドレスの識別
- Apache Airflow ウェブサーバーの VPC エンドポイントへのアクセス (プライベートネットワークアクセス)
  - の使用 AWS Client VPN
  - Linux 拠点ホストを使用する
  - Load Balancer の使用 (上級)

料金

- AWS PrivateLink 料金表

VPC エンドポイントの概要

Amazon MWAA 環境を作成すると、Amazon MWAA はお客様の環境用に 1 つから 2 つの VPC エンドポイントを作成します。これらのエンドポイントは、Amazon VPC 内の Elastic Network Interfaces (ENI) とプライベート IP アドレスで表されます。これらのエンドポイントが作成されると、これらの IPs 宛てのトラフィックは、環境で使用される対応する AWS サービスにプライベートまたはパブリックにルーティングされます。

パブリックネットワークアクセスモード

Apache Airflow ウェブサーバーでパブリックネットワークアクセスモードを選択した場合、ネットワークトラフィックはインターネット経由でパブリックにルーティングされます。

- Amazon MWAA は Amazon Aurora PostgreSQL メタデータデータベース用の VPC インターフェイスエンドポイントを作成します。エンドポイントは、プライベートサブネットにマッピングされたアベイラビリティーゾーンに作成され、他の AWS アカウントから独立しています。
- その後、Amazon MWAA はプライベートサブネットの IP アドレスをインターフェイスエンドポイントにバインドします。これは、Amazon VPC の各アベイラビリティーゾーンから単一の IP をバインドするというベストプラクティスをサポートするように設計されています。

プライベートネットワークアクセスモード

Apache Airflow ウェブサーバーでプライベートネットワークアクセスモードを選択した場合、ネットワークトラフィックは Amazon VPC 内でプライベートにルーティングされます。

- Amazon MWAA は、Apache Airflow ウェブサーバー用の VPC インターフェイスエンドポイントと、Amazon Aurora PostgreSQL メタデータデータベース用のインターフェイスエンドポイントを作成します。エンドポイントは、プライベートサブネットにマッピングされたアベイラビリティーゾーンに作成され、他の AWS アカウントから独立しています。
- その後、Amazon MWAA はプライベートサブネットの IP アドレスをインターフェイスエンドポイントにバインドします。これは、Amazon VPC の各アベイラビリティーゾーンから単一の IP をバインドするというベストプラクティスをサポートするように設計されています。

他の AWS サービスを使用するためのアクセス許可

インターフェイスエンドポイントは、 AWS Identity and Access Management (IAM) の環境の実行ロールを使用して、環境で使用される AWS リソースに対するアクセス許可を管理します。環境に対してより多くの AWS サービスが有効になっているため、各サービスでは、環境の実行ロールを使用してアクセス許可を設定する必要があります。権限を追加するには、Amazon MWAA 実行ロールを参照してください。

Apache Airflow ウェブサーバーのプライベートネットワークアクセスモードを選択した場合は、VPC エンドポイントポリシーで各エンドポイントのアクセス権限も許可する必要があります。詳細については、「the section called “VPC エンドポイントポリシー (プライベートルーティングのみ)”」を参照してください。

VPC エンドポイントの表示

このセクションでは、Amazon MWAA によって作成された VPC エンドポイントを表示する方法と、Apache Airflow VPC エンドポイントのプライベート IP アドレスを識別する方法について説明します。

Amazon VPC コンソールで VPC エンドポイントを表示する

以下のセクションでは、Amazon MWAA によって作成された VPC エンドポイント、および Amazon VPC にプライベートルーティングを使用している場合に作成した VPC エンドポイントを表示する手順を示します。

VPC エンドポイントを表示するには

1. Amazon VPC コンソールで「Endpoints ページ」 を開きます。
2. AWS リージョンセレクタを使用して、リージョンを選択します。
3. Amazon MWAA によって作成された VPC インターフェイスエンドポイントと、Amazon VPC でプライベートルーティングを使用している場合は作成した VPC エンドポイントが表示されます。

プライベートルーティングを使用する Amazon VPC に必要な VPC サービスエンドポイントの詳細については、「Amazon VPC に必要な VPC サービスエンドポイントをプライベートルーティングで作成する」を参照してください。

Apache Airflow ウェブサーバーとその VPC エンドポイントのプライベート IP アドレスの識別

次の手順では、Apache Airflow ウェブサーバーのホスト名と VPC インターフェースエンドポイントのホスト名、およびそれらのプライベート IP アドレスを取得する方法について説明します。

1. 次の AWS Command Line Interface （ AWS CLI) コマンドを使用して、Apache Airflow ウェブサーバー のホスト名を取得します。

```
aws mwaa get-environment --name YOUR_ENVIRONMENT_NAME --query
 'Environment.WebserverUrl'
```

次のような反応が表示されるはずだ：

```
"99aa99aa-55aa-44a1-a91f-f4552cf4e2f5-vpce.c10.us-west-2.airflow.amazonaws.com"
```

2. 前のコマンドの応答で返されたホスト名で dig コマンドを実行します。例:

```
dig CNAME +short 99aa99aa-55aa-44a1-a91f-f4552cf4e2f5-vpce.c10.us-
west-2.airflow.amazonaws.com
```

次のような反応が表示されるはずだ：

```
vpce-0699aa333a0a0a0-bf90xjtr.vpce-svc-00bb7c2ca2213bc37.us-
west-2.vpce.amazonaws.com.
```

3. 次の AWS Command Line Interface （ AWS CLI) コマンドを使用して、前のコマンドのレスポンスで返された VPC エンドポイント DNS 名を取得します。例:

```
aws ec2 describe-vpc-endpoints | grep vpce-0699aa333a0a0a0-bf90xjtr.vpce-
svc-00bb7c2ca2213bc37.us-west-2.vpce.amazonaws.com.
```

次のような反応が表示されるはずだ：

```
"DnsName": "vpce-066777a0a0a0-bf90xjtr.vpce-svc-00bb7c2ca2213bc37.us-
west-2.vpce.amazonaws.com",
```

4. Apache Airflow ホスト名とその VPC エンドポイントの DNS 名に対して nslookup または dig コマンドを実行して IP アドレスを取得します。例:

```
dig +short YOUR_AIRFLOW_HOST_NAME YOUR_AIRFLOW_VPC_ENDPOINT_DNS
```

次のような反応が表示されるはずだ：

```
192.0.5.1
  192.0.6.1
```

Apache Airflow ウェブサーバーの VPC エンドポイントへのアクセス (プライベートネットワークアクセス)

Apache Airflow ウェブサーバーでプライベートネットワークアクセスモードを選択した場合は、Apache Airflow ウェブサーバーの VPC インターフェースエンドポイントにアクセスするメカニズムを作成する必要があります。これらのリソースには、Amazon MWAA 環境と同じ Amazon VPC、VPC セキュリティグループ、プライベートサブネットを使用する必要があります。

の使用 AWS Client VPN

AWS Client VPN は、オンプレミスネットワーク内の AWS リソースとリソースに安全にアクセスできるマネージドクライアントベースの VPN サービスです。OpenVPN クライアントを使用して、どこからでも安全なTLS接続を提供しまsy。

Amazon MWAAチュートリアルに従って、Client VPNを構成することをお勧めします：チュートリアル: AWS Client VPN を使用したプライベートネットワークアクセスの設定。

Linux 拠点ホストを使用する

要塞ホストは、外部ネットワーク (インターネット経由など) からプライベートネットワークにアクセスできるようにすることを目的としたサーバーです。Linux インスタンスはパブリックサブネットにあり、拠点ホストを実行している基盤となる Amazon EC2 インスタンスにアタッチされたセキュリティグループからの SSH アクセスを許可するセキュリティグループを使用してセットアップされます。

Amazon MWAAのチュートリアルに従って、Linux Bastion Hostを構成することをお勧めします：チュートリアル: Linux 踏み台ホストを使用したプライベートネットワークアクセスの設定。

Load Balancer の使用 (上級)

次のセクションでは、Application Load Balancer に適用する必要がある設定を示しています。

1. 「ターゲットグループ」。Apache Airflow ウェブサーバーのプライベート IP アドレスとその VPC インターフェイスエンドポイントを指すターゲットグループを使用する必要があります。1 つのプライベート IP アドレスだけを使用すると可用性が低下する可能性があるため、両方のプライベート IP アドレスを登録ターゲットとして指定することをお勧めします。プライベート IP アドレスを識別する方法の詳細については、the section called “Apache Airflow ウェブサーバーとその VPC エンドポイントのプライベート IP アドレスの識別”を参照してください。
2. 「ステータスコード」。ターゲットグループ設定では`200`と`302`ステータスコードを使用することをお勧めします。そうしないと、Apache Airflow ウェブサーバーの VPC エンドポイントが`302 Redirect`エラーで応答した場合に、ターゲットに異常があるとフラグが立てられる可能性があります。
3. 「HTTPS リスナー」。Apache Airflow ウェブサーバーのターゲットポートを指定する必要があります。例:

[プロトコル]	[ポート]
HTTPS	443

4. ACM 新しいドメイン。で SSL/TLS 証明書を関連付ける場合は AWS Certificate Manager、ロードバランサーの HTTPS リスナー用の新しいドメインを作成する必要があります。
5. ACM証明書リージョン。で SSL/TLS 証明書を関連付ける場合は AWS Certificate Manager、環境と同じ AWS リージョンにアップロードする必要があります。例:

- Example 証明書をアップロードするリージョン

```
aws acm import-certificate --certificate fileb://Certificate.pem --certificate-
chain fileb://CertificateChain.pem --private-key fileb://PrivateKey.pem --
region us-west-2
```

Amazon VPC に必要な VPC サービスエンドポイントをプライベートルーティングで作成する

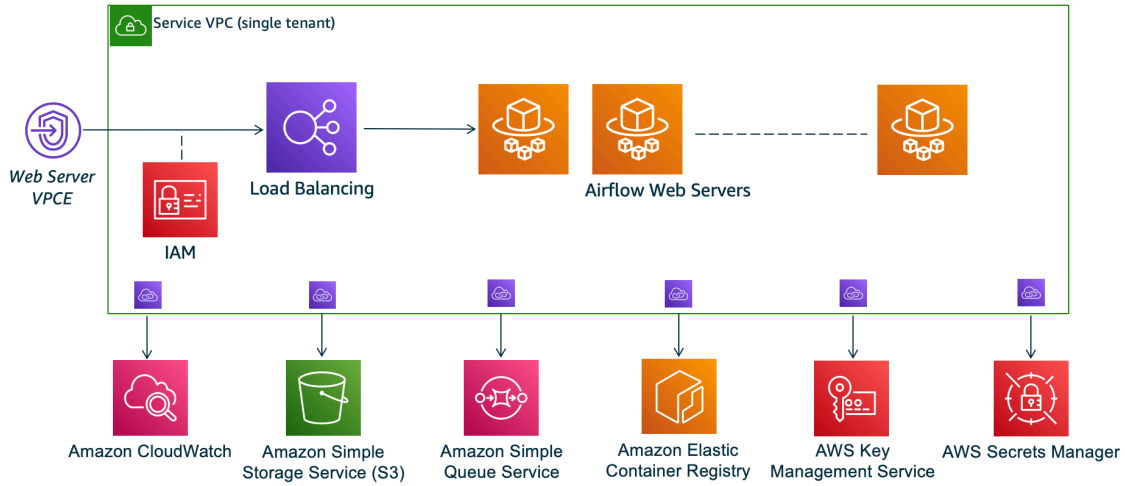
インターネットにアクセスできない既存の Amazon VPC ネットワークでは、Apache Airflow 用の Amazon マネージドワークフローで Apache Airflow を使用するには、追加の VPC サービスエンドポイント (AWS PrivateLink) が必要です。このページでは、Amazon MWAA が使用する AWS サービスに必要な VPC エンドポイント、Apache Airflow に必要な VPC エンドポイント、およびプライベートルーティングを使用して VPC エンドポイントを作成して既存の Amazon VPC にアタッチする方法について説明します。

目次

料金

- AWS PrivateLink 料金表

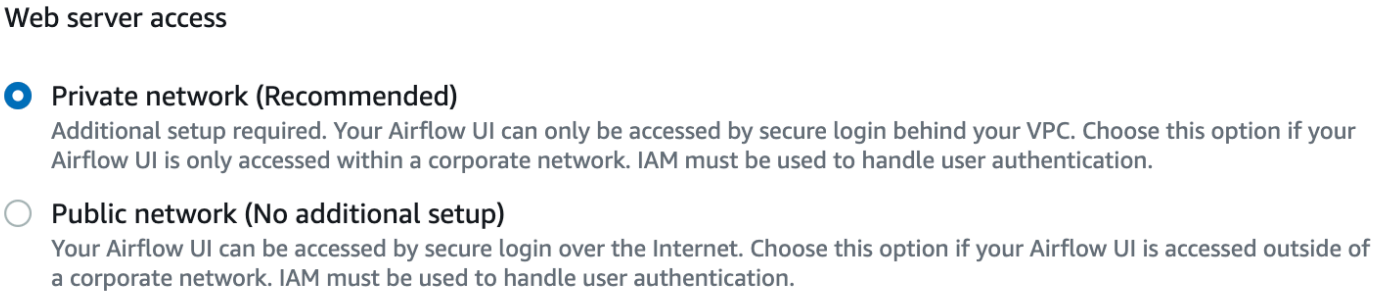
プライベートネットワークとプライベートルーティング



プライベートネットワークアクセスモードは、Apache Airflow UI へのアクセスを、「お使いの環境のIAM ポリシー」へのアクセスが許可されている Amazon VPC 内のユーザーに制限します。

プライベートなウェブサーバーアクセスを持つ環境を作成する場合、すべての依存関係を Python Wheel アーカイブ (`.whl`) にパッケージ化し、その後、`requirements.txt` で `.whl` を参照する必要があります。wheel を使用して依存関係をパッケージ化およびインストールする手順については、「Python wheel を使用した依存関係の管理」を参照してください。。

以下の画像は、Amazon MWAA コンソールの [プライベートネットワーク] オプションの場所を示しています。



- プライベートルーティング。「インターネットにアクセスできない Amazon VPC」は VPC 内のネットワークトラフィックを制限します。このページでは、Amazon VPC にインターネットアクセスがなく、環境で使用されるサービスごとに AWS VPC エンドポイントが必要であり、Amazon

MWAA 環境と同じ AWS リージョンと Amazon VPC の Apache Airflow 用の VPC エンドポイントが必要であることを前提としています。

(必須) VPC エンドポイント

次のセクションでは、インターネットにアクセスできない Amazon VPC に必要な VPC エンドポイントを示しています。Apache Airflow に必要な VPC エンドポイントを含め、Amazon MWAA が使用する各 AWS サービスの VPC エンドポイントを一覧表示します。

```
com.amazonaws.YOUR_REGION.s3
com.amazonaws.YOUR_REGION.monitoring
com.amazonaws.YOUR_REGION.ecr.dkr
com.amazonaws.YOUR_REGION.ecr.api
com.amazonaws.YOUR_REGION.logs
com.amazonaws.YOUR_REGION.sqs
com.amazonaws.YOUR_REGION.kms
com.amazonaws.YOUR_REGION.airflow.api
com.amazonaws.YOUR_REGION.airflow.env
com.amazonaws.YOUR_REGION.airflow.ops
```

必要な VPC エンドポイントのアタッチ

このセクションでは、Amazon VPC に必要な VPC エンドポイントをプライベートルーティングでアタッチする手順について説明します。

AWS サービスに必要な VPC エンドポイント

次のセクションでは、環境で使用される AWS サービスの VPC エンドポイントを既存の Amazon VPC にアタッチする手順を示します。

VPC エンドポイントをプライベートサブネットにアタッチするには

1. Amazon VPC コンソールで「Endpoints ページ」を開きます。
2. AWS リージョンセレクタを使用して、リージョンを選択します。
3. Amazon S3 のエンドポイントを作成する：
   a. [エンドポイントの作成] を選択します。
   b. [属性でフィルタリングまたはキーワードで検索] テキストフィールドに **.s3** と入力し、キーボードの [Enter] を押します。

c. [ゲートウェイ] タイプのリストにあるサービスエンドポイントを選択することをお勧めします。

例えば、次のようになります: **com.amazonaws.us-west-2.s3 amazon Gateway**

d. [VPC] 内の環境の Amazon VPC を選択してください。

e. 異なるアベイラビリティーゾーンにある 2 つのプライベートサブネットが選択されていることと、[DNS 名を有効化] を選択してそのプライベート DNS が有効になっていることを確認します。

f. 環境の Amazon VPC セキュリティグループを選択します。

g. [ポリシー] で [フルアクセス] を選択します。

h. [エンドポイントの作成] を選択します。

4. Amazon ECR の最初のエンドポイントを作成する：

a. [エンドポイントの作成] を選択します。

b. [属性でフィルタリングまたはキーワードで検索] テキストフィールドに **.ecr.dkr** と入力し、キーボードの [Enter] を押します。

c. サービスのエンドポイントを選択します。

d. [VPC] 内の環境の Amazon VPC を選択してください。

e. 異なるアベイラビリティーゾーンにある 2 つのプライベートサブネットが選択され、[DNS 名を有効化] が有効になっていることを確認します。

f. 環境の Amazon VPC セキュリティグループを選択します。

g. [ポリシー] で [フルアクセス] を選択します。

h. [エンドポイントの作成] を選択します。

5. Amazon ECR 用の 2 つ目のエンドポイントを作成する：

a. [エンドポイントの作成] を選択します。

b. [属性でフィルタリングまたはキーワードで検索] テキストフィールドに **.ecr.api** と入力し、キーボードの [Enter] を押します。

c. サービスのエンドポイントを選択します。

d. [VPC] 内の環境の Amazon VPC を選択してください。

e. 異なるアベイラビリティーゾーンにある 2 つのプライベートサブネットが選択され、[DNS 名を有効化] が有効になっていることを確認します。

f. 環境の Amazon VPC セキュリティグループを選択します。

g. [ポリシー] で [フルアクセス] を選択します。

h. [エンドポイントの作成] を選択します。

6. CloudWatch ログのエンドポイントを作成します。

a. [エンドポイントの作成] を選択します。

b. [属性でフィルタリングまたはキーワードで検索] テキストフィールドに **.logs** と入力し、キーボードの [Enter] を押します。

c. サービスのエンドポイントを選択します。

d. [VPC] 内の環境の Amazon VPC を選択してください。

e. 異なるアベイラビリティーゾーンにある 2 つのプライベートサブネットが選択され、[DNS 名を有効化] が有効になっていることを確認します。

f. 環境の Amazon VPC セキュリティグループを選択します。

g. [ポリシー] で [フルアクセス] を選択します。

h. [エンドポイントの作成] を選択します。

7. CloudWatch モニタリング用のエンドポイントを作成します。

a. [エンドポイントの作成] を選択します。

b. [属性でフィルタリングまたはキーワードで検索] テキストフィールドに **.monitoring** と入力し、キーボードの [Enter] を押します。

c. サービスのエンドポイントを選択します。

d. [VPC] 内の環境の Amazon VPC を選択してください。

e. 異なるアベイラビリティーゾーンにある 2 つのプライベートサブネットが選択され、[DNS 名を有効化] が有効になっていることを確認します。

f. 環境の Amazon VPC セキュリティグループを選択します。

g. [ポリシー] で [フルアクセス] を選択します。

h. [エンドポイントの作成] を選択します。

8. Amazon SQS のエンドポイントを作成する：

a. [エンドポイントの作成] を選択します。

b. [属性でフィルタリングまたはキーワードで検索] テキストフィールドに **.sqs** と入力し、キーボードの [Enter] を押します。

c. サービスのエンドポイントを選択します。

d. [VPC] 内の環境の Amazon VPC を選択してください。

e. 異なるアベイラビリティーゾーンにある 2 つのプライベートサブネットが選択され、[DNS 名を有効化] が有効になっていることを確認します。

f. 環境の Amazon VPC セキュリティグループを選択します。

g. [ポリシー] で [フルアクセス] を選択します。

h. [エンドポイントの作成] を選択します。

9. のエンドポイントを作成します AWS KMS。

a. [エンドポイントの作成] を選択します。

b. [属性でフィルタリングまたはキーワードで検索] テキストフィールドに **.kms** と入力し、キーボードの [Enter] を押します。

c. サービスのエンドポイントを選択します。

d. [VPC] 内の環境の Amazon VPC を選択してください。

e. 異なるアベイラビリティーゾーンにある 2 つのプライベートサブネットが選択され、[DNS 名を有効化] が有効になっていることを確認します。

f. 環境の Amazon VPC セキュリティグループを選択します。

g. [ポリシー] で [フルアクセス] を選択します。

h. [エンドポイントの作成] を選択します。

Apache Airflow 用の VPC エンドポイント

以下のセクションでは、Apache Airflow の VPC エンドポイントを既存の Amazon VPC にアタッチする手順を示しています。

VPC エンドポイントをプライベートサブネットにアタッチするには

1. Amazon VPC コンソールで「Endpoints ページ」を開きます。
2. AWS リージョンセレクタを使用して、リージョンを選択します。
3. Apache Airflow API のエンドポイントを作成します。

a. [エンドポイントの作成] を選択します。

b. [属性でフィルタリングまたはキーワードで検索] テキストフィールドに **.airflow.api** と入力し、キーボードの [Enter] を押します。

c. サービスのエンドポイントを選択します。

d. [VPC] 内の環境の Amazon VPC を選択してください。

e. 異なるアベイラビリティーゾーンにある 2 つのプライベートサブネットが選択され、[DNS 名を有効化] が有効になっていることを確認します。

f. 環境の Amazon VPC セキュリティグループを選択します。

g. [ポリシー] で [フルアクセス] を選択します。

h. [エンドポイントの作成] を選択します。

4. Apache Airflow 環境の 1 つ目のエンドポイントを作成します。

a. [エンドポイントの作成] を選択します。

b. [属性でフィルタリングまたはキーワードで検索] テキストフィールドに **`.airflow.env`** と入力し、キーボードの [Enter] を押します。

c. サービスのエンドポイントを選択します。

d. [VPC] 内の環境の Amazon VPC を選択してください。

e. 異なるアベイラビリティーゾーンにある 2 つのプライベートサブネットが選択され、[DNS 名を有効化] が有効になっていることを確認します。

f. 環境の Amazon VPC セキュリティグループを選択します。

g. [ポリシー] で [フルアクセス] を選択します。

h. [エンドポイントの作成] を選択します。

5. Apache Airflow オペレーション用の 2 つ目のエンドポイントを作成します。

a. [エンドポイントの作成] を選択します。

b. [属性でフィルタリングまたはキーワードで検索] テキストフィールドに **`.airflow.ops`** と入力し、キーボードの [Enter] を押します。

c. サービスのエンドポイントを選択します。

d. [VPC] 内の環境の Amazon VPC を選択してください。

e. 異なるアベイラビリティーゾーンにある 2 つのプライベートサブネットが選択され、[DNS 名を有効化] が有効になっていることを確認します。

f. 環境の Amazon VPC セキュリティグループを選択します。

g. [ポリシー] で [フルアクセス] を選択します。

h. [エンドポイントの作成] を選択します。

(オプション) Amazon S3 VPC インターフェイスエンドポイントのプライベート IP アドレスを有効にする

Amazon S3 インターフェイスエンドポイントはプライベート DNS をサポートしていません。S3 エンドポイントのリクエストは引き続きパブリック IP アドレスに解決されます。S3 アドレスをプライベート IP アドレスに解決するには、S3 リージョンエンドポイントの「Route 53 にプライベートホストゾーン」を追加する必要があります。

Route 53 の使用

このセクションでは、Route 53 を使用して S3 インターフェイスエンドポイントのプライベート IP アドレスを有効にする手順について説明します。

1. Amazon S3 VPC インターフェイスエンドポイント (s3.eu-west-1.amazonaws.com など) 用のプライベートホストゾーンを作成し、Amazon VPC に関連付けます。
2. Amazon S3 VPC インターフェイスエンドポイント (s3.eu-west-1.amazonaws.com など) の ALIAS A レコードを作成します。このレコードは VPC インターフェイスエンドポイントの DNS 名に関連付けられます。
3. Amazon S3 インターフェイスエンドポイント (*. s3.eu-west-1.amazonaws.com など) に対して、VPC インターフェースエンドポイントの DNS 名に解決する ALIAS A ワイルドカードレコードを作成します。

カスタム DNS を使用する VPC

Amazon VPC がカスタム DNS ルーティングを使用している場合は、CNAME レコードを作成して DNS リゾルバー (Route 53 ではなく、通常は DNS サーバーを実行している EC2 インスタンス) に変更を加える必要があります。例:

```
Name: s3.us-west-2.amazonaws.com
Type: CNAME
Value:  *.vpce-0f67d23e37648915c-e2q2e2j3.s3.us-west-2.vpce.amazonaws.com
```

Amazon MWAA での独自の Amazon VPC エンドポイントの管理

Amazon MWAA は Amazon VPC エンドポイントを使用して、Apache Airflow 環境の設定に必要なさまざまなAWSサービスと統合します。独自のエンドポイントの管理には、主に次の 2 つのユースケースがあります。

1. これは、 を使用して複数のAWSアカウントを管理し、リソースを共有するときにAWS Organizations、共有 Amazon VPC に Apache Airflow 環境を作成できることを意味します。
2. エンドポイントを使用する特定のリソースにアクセス許可を絞り込むことで、より制限の厳しいアクセスポリシーを使用しましょう。

独自の VPC エンドポイントを管理する場合は、環境 RDS for PostgreSQL データベースと環境ウェブサーバー用に独自のエンドポイントを作成する責任があります。

Amazon MWAA が Apache Airflow をクラウドにデプロイする方法の詳細については、「Amazon MWAA アーキテクチャ図」を参照してください。

共有 Amazon VPC での環境の作成

AWS Organizations を使用してリソースを共有する複数のAWSアカウントを管理する場合は、Amazon MWAA でカスタマーマネージド VPC エンドポイントを使用して、組織内の別のアカウントと環境リソースを共有できます。

共有 VPC アクセスを設定すると、メインの Amazon VPC を所有するアカウント (所有者) は、Amazon MWAA に必要な 2 つのプライベートサブネットを、同じ組織に属する他のアカウント (参加者) と共有します。これらのサブネットを共有する参加者アカウントは、共有 Amazon VPC 内の環境を表示、作成、変更、削除できます。

組織内のアカウントとして機能し、Amazon VPC リソースを所有`Owner`する`Root`アカウント と、同じ組織のメンバー`Participant`である参加者アカウント があるとします。が と共有している Amazon VPC に新しい Amazon MWAA `Participant`を作成すると`Owner`、Amazon MWAA はまずサービス VPC リソースを作成し、次に最大 72 時間 `PENDING`状態になります。

環境ステータスが から `CREATING`に変わると`PENDING`、 に代わって動作するプリンシパルによって必要なエンドポイント`Owner`が作成されます。これを行うために、Amazon MWAA は Amazon MWAA コンソールにデータベースとウェブサーバーのエンドポイントを一覧表示します。`GetEnvironment` API アクションを呼び出してサービスエンドポイントを取得することもできます。

> **Note**
>
> リソースの共有に使用する Amazon VPC がプライベート Amazon VPC である場合は、「」で説明されている手順を完了する必要がありますthe section called “VPC エンドポイントへのアクセスの管理”。このトピックでは、Amazon ECR、Amazon ECS、Amazon SQS な

ど、 がAWS統合する他のAWSサービスに関連するさまざまな Amazon VPC エンドポイントの設定について説明します。これらのサービスは、クラウドで Apache Airflow 環境を運用および管理するために不可欠です。

前提条件

共有 VPC に Amazon MWAA 環境を作成する前に、次のリソースが必要です。

- Amazon VPC を所有するAWSアカウントとしてOwner使用されるアカウント。
- ルート としてMyOrganization作成されたAWS Organizations組織単位。
- 下の 2 番目のAWSアカウント MyOrganizationでParticipant、新しい環境を作成する参加者アカウントを提供します。

さらに、Amazon VPC でリソースを共有するときは、所有者と参加者の責任とアクセス許可を理解しておくことをお勧めします。

Amazon VPC を作成する

まず、所有者アカウントと参加者アカウントが共有する新しい Amazon VPC を作成します。

1. を使用してコンソールにサインインOwnerし、AWS CloudFormationコンソールを開きます。次のテンプレートを使用してスタックを作成します。このスタックは、Amazon VPC を含む多数のネットワークリソースと、このシナリオで 2 つのアカウントが共有するサブネットをプロビジョニングします。

```
AWSTemplateFormatVersion: "2010-09-09"
Description: >-
  This template deploys a VPC, with a pair of public and private subnets spread
  across two Availability Zones. It deploys an internet gateway, with a default
  route on the public subnets. It deploys a pair of NAT gateways (one in each
  AZ), and default routes for them in the private subnets.
Parameters:
  EnvironmentName:
    Description: An environment name that is prefixed to resource names
    Type: String
    Default: mwaa-
  VpcCIDR:
    Description: Please enter the IP range (CIDR notation) for this VPC
    Type: String
```

```
    Default: 10.192.0.0/16
  PublicSubnet1CIDR:
    Description: >-
      Please enter the IP range (CIDR notation) for the public subnet in the
      first Availability Zone
    Type: String
    Default: 10.192.10.0/24
  PublicSubnet2CIDR:
    Description: >-
      Please enter the IP range (CIDR notation) for the public subnet in the
      second Availability Zone
    Type: String
    Default: 10.192.11.0/24
  PrivateSubnet1CIDR:
    Description: >-
      Please enter the IP range (CIDR notation) for the private subnet in the
      first Availability Zone
    Type: String
    Default: 10.192.20.0/24
  PrivateSubnet2CIDR:
    Description: >-
      Please enter the IP range (CIDR notation) for the private subnet in the
      second Availability Zone
    Type: String
    Default: 10.192.21.0/24
Resources:
  VPC:
    Type: 'AWS::EC2::VPC'
    Properties:
      CidrBlock: !Ref VpcCIDR
      EnableDnsSupport: true
      EnableDnsHostnames: true
      Tags:
        - Key: Name
          Value: !Ref EnvironmentName
  InternetGateway:
    Type: 'AWS::EC2::InternetGateway'
    Properties:
      Tags:
        - Key: Name
          Value: !Ref EnvironmentName
  InternetGatewayAttachment:
    Type: 'AWS::EC2::VPCGatewayAttachment'
    Properties:
```

```
      InternetGatewayId: !Ref InternetGateway
      VpcId: !Ref VPC
  PublicSubnet1:
    Type: 'AWS::EC2::Subnet'
    Properties:
      VpcId: !Ref VPC
      AvailabilityZone: !Select
        - 0
        - !GetAZs ''
      CidrBlock: !Ref PublicSubnet1CIDR
      MapPublicIpOnLaunch: true
      Tags:
        - Key: Name
          Value: !Sub '${EnvironmentName} Public Subnet (AZ1)'
  PublicSubnet2:
    Type: 'AWS::EC2::Subnet'
    Properties:
      VpcId: !Ref VPC
      AvailabilityZone: !Select
        - 1
        - !GetAZs ''
      CidrBlock: !Ref PublicSubnet2CIDR
      MapPublicIpOnLaunch: true
      Tags:
        - Key: Name
          Value: !Sub '${EnvironmentName} Public Subnet (AZ2)'
  PrivateSubnet1:
    Type: 'AWS::EC2::Subnet'
    Properties:
      VpcId: !Ref VPC
      AvailabilityZone: !Select
        - 0
        - !GetAZs ''
      CidrBlock: !Ref PrivateSubnet1CIDR
      MapPublicIpOnLaunch: false
      Tags:
        - Key: Name
          Value: !Sub '${EnvironmentName} Private Subnet (AZ1)'
  PrivateSubnet2:
    Type: 'AWS::EC2::Subnet'
    Properties:
      VpcId: !Ref VPC
      AvailabilityZone: !Select
        - 1
```

```
      - !GetAZs ''
    CidrBlock: !Ref PrivateSubnet2CIDR
    MapPublicIpOnLaunch: false
    Tags:
      - Key: Name
        Value: !Sub '${EnvironmentName} Private Subnet (AZ2)'
NatGateway1EIP:
  Type: 'AWS::EC2::EIP'
  DependsOn: InternetGatewayAttachment
  Properties:
    Domain: vpc
NatGateway2EIP:
  Type: 'AWS::EC2::EIP'
  DependsOn: InternetGatewayAttachment
  Properties:
    Domain: vpc
NatGateway1:
  Type: 'AWS::EC2::NatGateway'
  Properties:
    AllocationId: !GetAtt NatGateway1EIP.AllocationId
    SubnetId: !Ref PublicSubnet1
NatGateway2:
  Type: 'AWS::EC2::NatGateway'
  Properties:
    AllocationId: !GetAtt NatGateway2EIP.AllocationId
    SubnetId: !Ref PublicSubnet2
PublicRouteTable:
  Type: 'AWS::EC2::RouteTable'
  Properties:
    VpcId: !Ref VPC
    Tags:
      - Key: Name
        Value: !Sub '${EnvironmentName} Public Routes'
DefaultPublicRoute:
  Type: 'AWS::EC2::Route'
  DependsOn: InternetGatewayAttachment
  Properties:
    RouteTableId: !Ref PublicRouteTable
    DestinationCidrBlock: 0.0.0.0/0
    GatewayId: !Ref InternetGateway
PublicSubnet1RouteTableAssociation:
  Type: 'AWS::EC2::SubnetRouteTableAssociation'
  Properties:
    RouteTableId: !Ref PublicRouteTable
```

```
      SubnetId: !Ref PublicSubnet1
  PublicSubnet2RouteTableAssociation:
    Type: 'AWS::EC2::SubnetRouteTableAssociation'
    Properties:
      RouteTableId: !Ref PublicRouteTable
      SubnetId: !Ref PublicSubnet2
  PrivateRouteTable1:
    Type: 'AWS::EC2::RouteTable'
    Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: !Sub '${EnvironmentName} Private Routes (AZ1)'
  DefaultPrivateRoute1:
    Type: 'AWS::EC2::Route'
    Properties:
      RouteTableId: !Ref PrivateRouteTable1
      DestinationCidrBlock: 0.0.0.0/0
      NatGatewayId: !Ref NatGateway1
  PrivateSubnet1RouteTableAssociation:
    Type: 'AWS::EC2::SubnetRouteTableAssociation'
    Properties:
      RouteTableId: !Ref PrivateRouteTable1
      SubnetId: !Ref PrivateSubnet1
  PrivateRouteTable2:
    Type: 'AWS::EC2::RouteTable'
    Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: !Sub '${EnvironmentName} Private Routes (AZ2)'
  DefaultPrivateRoute2:
    Type: 'AWS::EC2::Route'
    Properties:
      RouteTableId: !Ref PrivateRouteTable2
      DestinationCidrBlock: 0.0.0.0/0
      NatGatewayId: !Ref NatGateway2
  PrivateSubnet2RouteTableAssociation:
    Type: 'AWS::EC2::SubnetRouteTableAssociation'
    Properties:
      RouteTableId: !Ref PrivateRouteTable2
      SubnetId: !Ref PrivateSubnet2
  SecurityGroup:
    Type: 'AWS::EC2::SecurityGroup'
```

```
    Properties:
      GroupName: mwaa-security-group
      GroupDescription: Security group with a self-referencing inbound rule.
      VpcId: !Ref VPC
  SecurityGroupIngress:
    Type: 'AWS::EC2::SecurityGroupIngress'
    Properties:
      GroupId: !Ref SecurityGroup
      IpProtocol: '-1'
      SourceSecurityGroupId: !Ref SecurityGroup
Outputs:
  VPC:
    Description: A reference to the created VPC
    Value: !Ref VPC
  PublicSubnets:
    Description: A list of the public subnets
    Value: !Join
      - ','
      - - !Ref PublicSubnet1
        - !Ref PublicSubnet2
  PrivateSubnets:
    Description: A list of the private subnets
    Value: !Join
      - ','
      - - !Ref PrivateSubnet1
        - !Ref PrivateSubnet2
  PublicSubnet1:
    Description: A reference to the public subnet in the 1st Availability Zone
    Value: !Ref PublicSubnet1
  PublicSubnet2:
    Description: A reference to the public subnet in the 2nd Availability Zone
    Value: !Ref PublicSubnet2
  PrivateSubnet1:
    Description: A reference to the private subnet in the 1st Availability Zone
    Value: !Ref PrivateSubnet1
  PrivateSubnet2:
    Description: A reference to the private subnet in the 2nd Availability Zone
    Value: !Ref PrivateSubnet2
  SecurityGroupIngress:
    Description: Security group with self-referencing inbound rule
    Value: !Ref SecurityGroupIngress
```

2. 新しい Amazon VPC リソースがプロビジョニングされたら、AWS Resource Access Manager コンソールに移動し、リソース共有の作成を選択します。

3. と共有できる利用可能なサブネットのリストから、最初のステップで作成したサブネットを選択しますParticipant。

環境を作成します。

カスタマー管理の Amazon VPC エンドポイントを使用して Amazon MWAA 環境を作成するには、次のステップを実行します。

1. を使用してサインインしParticipant、Amazon MWAA コンソールを開きます。ステップ 1: 詳細を指定して、新しい環境の Amazon S3 バケット、DAG フォルダ、依存関係を指定します。詳細については、「 の開始方法」を参照してください。
2. 詳細設定の構成ページのネットワーク で、共有 Amazon VPC からサブネットを選択します。
3. 「エンドポイント管理」で、ドロップダウンリストから「CUSTOMER」を選択します。
4. ページの残りのオプションのデフォルトのままにして、確認と作成 ページの環境の作成 を選択します。

環境は CREATING状態から始まり、 に変わりますPENDING。環境が の場合PENDING、コンソールを使用してデータベースエンドポイントサービス名とウェブサーバーエンドポイントサービス名 (プライベートウェブサーバーをセットアップした場合) を書き留めます。

Amazon MWAA コンソールを使用して新しい環境を作成する場合。Amazon MWAA は、必要なインバウンドルールとアウトバウンドルールを使用して新しいセキュリティグループを作成します。セキュリティグループ ID を書き留めます。

次のセクションOwnerでは、 はサービスエンドポイントとセキュリティグループ ID を使用して、共有 Amazon VPC に新しい Amazon VPC エンドポイントを作成します。

Amazon VPC エンドポイントを作成する

次のステップを実行して、環境に必要な Amazon VPC エンドポイントを作成します。

1. Owner、 AWS Management Consoleを開いて にサインインしますhttps://console.aws.amazon.com/vpc/。
2. 左側のナビゲーションパネルからセキュリティグループを選択し、次のインバウンドルールとアウトバウンドルールを使用して、共有 Amazon VPC に新しいセキュリティグループを作成します。

	タイプ	プロトコル	ソースタイプ	ソース
インバウンド	すべてのトラフィック	すべて	すべて	環境セキュリティグループ
アウトバウンド	すべてのトラフィック	すべて	すべて	0.0.0.0/0

> **Warning**
>
> Owner アカウントは、新しい環境から共有 Amazon VPC へのトラフィックを許可するセキュリティグループをOwnerアカウントに設定する必要があります。これを行うには、 で新しいセキュリティグループを作成するかOwner、既存のセキュリティグループを編集します。

3. エンドポイント を選択し、前の手順のエンドポイントサービス名を使用して、環境データベースとウェブサーバー (プライベートモードの場合) の新しいエンドポイントを作成します。共有 Amazon VPC、環境に使用したサブネット、環境のセキュリティグループを選択します。

成功すると、環境は PENDING から にCREATING、最後に に変わりますAVAILABLE。の場合AVAILABLE、Apache Airflow コンソールにサインインできます。

共有 Amazon VPC のトラブルシューティング

共有 Amazon VPC で環境を作成するときに発生する問題を解決するには、次のリファレンスを使用します。

PENDING ステータスCREATE_FAILEDが 以降の環境

- Participant を使用して、 Ownerがサブネットを と共有していることを確認しますAWS Resource Access Manager。
- データベースとウェブサーバーの Amazon VPC エンドポイントが、環境に関連付けられた同じサブネットに作成されていることを確認します。
- エンドポイントで使用されるセキュリティグループが、環境に使用されるセキュリティグループからのトラフィックを許可していることを確認します。Owner アカウントは、

のセキュリティグループを Participantとして参照するルールを作成します*account-number*/*security-group-id*。

タイプ	プロトコル	ソースタイプ	ソース
すべてのトラフィック	すべて	すべて	*123456789012* /*sg-0909e8e81919*

詳細については、「所有者と参加者の責任とアクセス許可」を参照してください。

環境が **PENDING**ステータスのままになる

各 VPC エンドポイントのステータスを検証して、 であることを確認しますAvailable。プライベートウェブサーバーで環境を設定する場合は、ウェブサーバーのエンドポイントも作成する必要があります。環境が にスタックしている場合はPENDING、プライベートウェブサーバーのエンドポイントが欠落している可能性があります。

受信**The Vpc Endpoint Service '*vpce-service-name*' does not exist**エラー

次のエラーが表示された場合は、共有 VPC を所有するアカウントにエンドポイントを作成するOwnerアカウントを確認します。

```
ClientError: An error occurred (InvalidServiceName) when calling the
 CreateVpcEndpoint operation:

The Vpc Endpoint Service 'vpce-service-name' does not exist
```

Apache Airflow のための Amazon Managed Workflows のチュートリアル

このガイドには、Amazon Managed Workflows for Apache Airflow 環境の使用と設定に関する step-by-step チュートリアルが含まれています。

トピック

チュートリアル: AWS Client VPN を使用したプライベートネットワークアクセスの設定

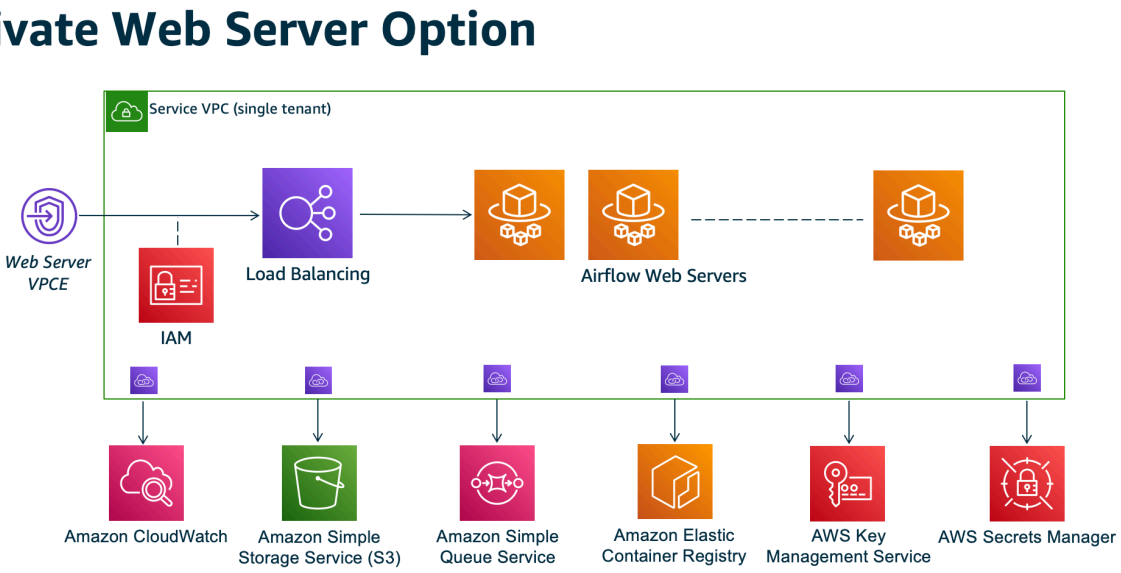
このチュートリアルでは、Amazon Managed Workflows for Apache Airflow 環境用に、コンピュータから Apache Airflow ウェブサーバーへの VPN トンネルを作成する手順を説明します。VPN トンネルを介してインターネットに接続するには、まず AWS Client VPN エンドポイントを作成する必要があります。設定が完了すると、クライアント VPN エンドポイントは VPN サーバーとして機能し、コンピューターから VPC 内のリソースへの安全な接続を可能にします。次に、「AWS Client VPN デスクトップ用」を使用してコンピューターから Client VPN に接続します。

セクション

- ステップ 6: クライアント VPN に認証進入ルールを追加する
- ステップ 7 : クライアント VPN のエンドポイント設定ファイルをダウンロードします
- ステップ 8: AWS Client VPN に接続する
- 次のステップ

プライベートネットワーク

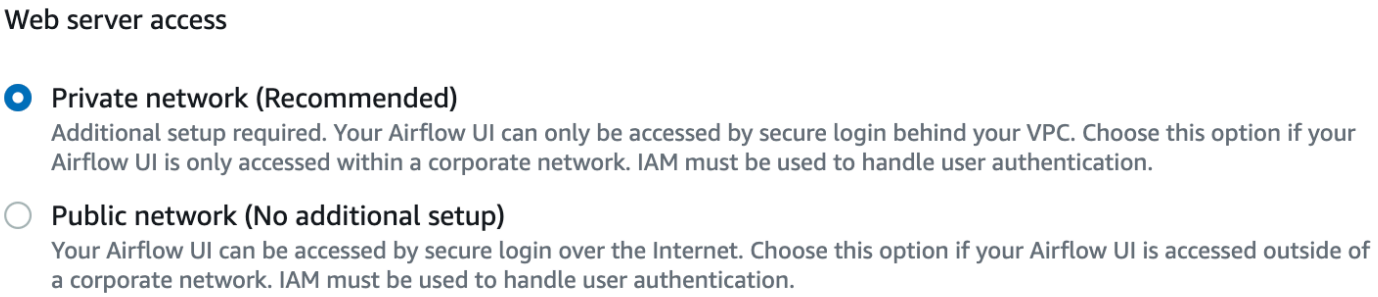
このチュートリアルでは、Apache Airflow ウェブサーバーの [プライベートネットワーク] アクセスモードを選択していることを前提としています。



プライベートネットワークアクセスモードは、Apache Airflow UI へのアクセスを、「お使いの環境のIAM ポリシー」へのアクセスが許可されている Amazon VPC 内のユーザーに制限します。

プライベートなウェブサーバーアクセスを持つ環境を作成する場合、すべての依存関係を Python Wheel アーカイブ (`.whl`) にパッケージ化し、その後、`requirements.txt` で `.whl` を参照する必要があります。wheel を使用して依存関係をパッケージ化およびインストールする手順については、「Python wheel を使用した依存関係の管理」を参照してください。。

以下の画像は、Amazon MWAA コンソールの [プライベートネットワーク] オプションの場所を示しています。



ユースケース

このチュートリアルは、Amazon MWAA 環境を作成する前でも後でも使用できます。環境と同じ Amazon VPC、VPC セキュリティグループ、プライベートサブネットを使用する必要があります。Amazon MWAA 環境を作成した後にこのチュートリアルを使用する場合、手順を完了したら Amazon MWAA コンソールに戻り、Apache Airflow ウェブサーバーのアクセスモードを [プライベートネットワーク] に変更できます。

開始する前に

1. ユーザー許可を確認してください。AWS Identity and Access Management (IAM) のアカウントに VPC リソースを作成および管理するための十分な権限があることを確認してください。
2. Amazon MWAA VPC を使用してください。このチュートリアルでは、クライアント VPN を既存の VPC に関連付けることを前提としています。Amazon VPC は、Amazon MWAA 環境と同じ AWS のリージョンにある必要があり、かつ 2 つのプライベートサブネットを持っている必要があります。Amazon VPC を作成していない場合、オプション 3: インターネットにアクセスせずに Amazon VPC ネットワークを作成する で AWS CloudFormation テンプレートを使用してください。

目的

このチュートリアルでは、以下の作業を行います。

1. 既存の Amazon VPC に対して、AWS CloudFormation テンプレートを使用して AWS Client VPN エンドポイントを作成します。
2. サーバーおよびクライアントの証明書とキーを生成し、その後、Amazon MWAA 環境と同じ AWS のリージョンにある AWS Certificate Manager にサーバー証明書とキーをアップロードします。

3. クライアント VPN のクライアント VPN エンドポイント設定ファイルをダウンロードして変更し、このファイルを使用して、デスクトップ用クライアント VPN を使用して接続するための VPN プロファイルを作成します。

(オプション) ステップ 1: VPC、CIDR ルール、VPC セキュリティを特定する

次のセクションでは、Amazon VPC、VPC セキュリティグループの ID を検索する方法と、以降のステップでクライアント VPN を作成するために必要な CIDR ルールを特定する方法について説明します。

CIDR ルールを特定してください。

次のセクションでは、クライアント VPN を作成するために必要な CIDR ルールを特定する方法を示します。

クライアント VPN の CIDR を識別するには

1. Amazon VPC コンソールの「Amazon VPC ページ」を開きます。
2. ナビゲーションバーのリージョンセレクターを使用して、Amazon MWAA 環境と同じ AWS リージョンを選択します。
3. Amazon VPC を選択してください。
4. プライベートサブネットの CIDR が以下のようになっていると仮定します。

   - プライベートサブネット 1: 10.192.10.0 /24
   - プライベートサブネット 2: 10.192.11.0 /24

   Amazon VPC の CIDR が 10.192.0.0 /16 の場合、クライアント VPN に指定するクライアント IPv4 CIDR は 10.192.0.0 /22 になります。
5. この CIDR 値と VPC ID の値を後続のステップのために保存してください。

VPC とセキュリティグループを特定する

次のセクションでは、クライアント VPN を作成するために必要な Amazon VPC とセキュリティグループの ID を確認する方法を示します。

> **Note**
>
> 複数のセキュリティグループを使用している可能性があります。後続のステップでは、VPC のセキュリティグループをすべて指定する必要があります。

セキュリティグループを特定するには

1. Amazon VPC コンソールの「セキュリティグループページ」を開きます。
2. ナビゲーションバーのリージョンセレクターを使って、AWS リージョンを選択します。
3. VPC ID で Amazon VPC を探し、その VPC に関連付けられているセキュリティグループを特定します。
4. セキュリティグループと VPC の ID を後続のステップのために保存します。

ステップ 2: サーバーとクライアントの証明書を作成する

クライアント VPN エンドポイントは、1024 ビットおよび 2048 ビットの RSA キーサイズのみサポートしています。以下のセクションでは、OpenVPN easy-rsa を使ってサーバーとクライアントの証明書と鍵を生成し、AWS Command Line Interface (AWS CLI) を使って証明書を ACM にアップロードする方法を示します。

クライアント証明書を作成するには

1. 以下の簡単な手順に従って、「クライアントの認証と承認:相互認証」から証明書を作成し、ACM にアップロードするための AWS CLI を使用してください。
2. これらの手順では、サーバーおよびクライアント証明書をアップロードする際に、AWS CLI コマンドで Amazon MWAA 環境と同じ AWS リージョンを指定する必要があります。ここでは、これらのコマンドでリージョンを指定する方法を例で示します。

    a. Example サーバー証明書のリージョン

```
aws acm import-certificate --certificate fileb://server.crt --private-key
 fileb://server.key --certificate-chain fileb://ca.crt --region us-west-2
```

b. Example クライアント証明書のリージョン

```
aws acm import-certificate --certificate fileb://client1.domain.tld.crt
 --private-key fileb://client1.domain.tld.key --certificate-chain fileb://
ca.crt --region us-west-2
```

c. これらの手順の後、サーバー証明書およびクライアント証明書の ARN に関する AWS CLI のレスポンスで返された値を保存してください。これらの ARN を使用して、クライアント VPN を作成するための AWS CloudFormation テンプレートに ARN を指定します。

3. この手順では、クライアント証明書とプライベートキーがコンピューターに保存されます。ここでは、これらの認証情報の確認場所の例。

a. Example macOS 上

macOS では、内容は `/Users/youruser/custom_folder` に保存されます。このディレクトリのすべての (`ls -a`) 内容を一覧表示すると、次のような内容が表示されるはずです。

```
.
..
ca.crt
client1.domain.tld.crt
client1.domain.tld.key
server.crt
server.key
```

b. これらの手順の後、クライアント証明書の内容または場所を `client1.domain.tld.crt` に保存し、プライベートキーの場所を `client1.domain.tld.key` にメモしてください。これらの値をクライアント VPN の設定ファイルに追加します。

ステップ 3: AWS CloudFormation テンプレートをローカルに保存する

次のセクションには、クライアント VPN を作成するための AWS CloudFormation テンプレートが含まれています。Amazon MWAA 環境と同じ Amazon VPC、VPC セキュリティグループ、およびプライベートサブネットを指定する必要があります。

- 次のテンプレートの内容をコピーし、`mwaa_vpn_client.yaml` としてローカルに保存します。「テンプレートをダウンロードする」こともできます。

以下の値を置き換えます。

- **YOUR_CLIENT_ROOT_CERTIFICATE_ARN** — ClientRootCertificateChainArn の「client1.domain.tld」証明書のARN。
- **YOUR_SERVER_CERTIFICATE_ARN** — ServerCertificateArn 内のサーバー証明書の ARN。
- ClientCidrBlock のクライアント IPv4 CIDR ルール。10.192.0.0/22 の CIDR ルールが提供されています。
- VpcId の Amazon VPC ID 。vpc-010101010101 の VPC が提供されています。
- SecurityGroupIds 内の VPC セキュリティグループ ID。sg-0101010101 のセキュリティグループが提供されています。

```
AWSTemplateFormatVersion: 2010-09-09
Description: This template deploys a VPN Client Endpoint.
Resources:
  ClientVpnEndpoint:
    Type: 'AWS::EC2::ClientVpnEndpoint'
    Properties:
      AuthenticationOptions:
        - Type: "certificate-authentication"
          MutualAuthentication:
            ClientRootCertificateChainArn: "YOUR_CLIENT_ROOT_CERTIFICATE_ARN"
      ClientCidrBlock: 10.192.0.0/22
      ClientConnectOptions:
        Enabled: false
      ConnectionLogOptions:
        Enabled: false
      Description: "MWAA Client VPN"
      DnsServers: []
      SecurityGroupIds:
        - sg-0101010101
      SelfServicePortal: ''
      ServerCertificateArn: "YOUR_SERVER_CERTIFICATE_ARN"
      SplitTunnel: true
      TagSpecifications:
        - ResourceType: "client-vpn-endpoint"
          Tags:
          - Key: Name
            Value: MWAA-Client-VPN
```

```
    TransportProtocol: udp
    VpcId: vpc-010101010101
    VpnPort: 443
```

> **Note**
>
> ご使用の環境で複数のセキュリティグループを使用している場合は、次の形式で複数のセキュリティグループを指定できます。
>
> ```
> SecurityGroupIds:
>     - sg-0112233445566778b
>     - sg-0223344556677889f
> ```

ステップ 4: クライアント VPN AWS CloudFormation スタックを作成する

AWS Client VPN を作成するには

1. AWS CloudFormation コンソールを開きます。
2. [テンプレートは準備できている]、[テンプレートファイルをアップロードする] を選択します。
3. [ファイルのを選択] を選択し、`mwaa_vpn_client.yaml` ファイルを選択します。
4.
5. [次へ]、[次へ] を選択します。
6. アクノレッジメントを選択し、[スタックを作成] を選択します。

ステップ 5: サブネットをクライアント VPN に関連付ける

プライベートサブネットを AWS Client VPN に関連付けるには

1. Amazon VPC コンソールを開きます。
2. [クライアント VPN エンドポイント] ページを選択します。
3. クライアント VPN を選択し、次に [アソシエーション] タブの [アソシエイト] を選択します。
4. ドロップダウンリストから次の項目を選択します。

    - [VPC] 内の Amazon VPC。

- [関連付けるサブネットを選択] にあるプライベートサブネットの 1 つ。

5. [関連付ける] を選択します。

> **Note**
>
> VPC とサブネットがクライアント VPN に関連付けられるまでに数分かかります。

ステップ 6: クライアント VPN に認証進入ルールを追加する

VPC の CIDR ルールを使用する認証進入ルールをクライアント VPN に追加する必要があります。Active Directory グループまたは SAML ベースのアイデンティティプロバイダー (IdP) から特定のユーザーまたはグループを承認する場合は、Client VPN ガイドの「承認ルール」を参照してください。

CIDR を AWS Client VPN に追加するには

1. Amazon VPC コンソールを開きます。
2. [クライアント VPN エンドポイント] ページを選択します。
3. クライアント VPN を選択し、[承認] タブの [進入を許可] を選択します。
4. 次を指定します:
   - [有効化する送信先ネットワーク] にある Amazon VPC の CIDR ルール。例:

     ```
     10.192.0.0/16
     ```

   - [アクセスを付与する対象] で、[すべてのユーザーにアクセスを許可する] を選択します。
   - [説明] で、分かりやすい名前を入力します。
5. [認可ルールを追加する] を選択します。

> **Note**
>
> Amazon VPC のネットワークコンポーネントによっては、ネットワークアクセスコントロールリスト (NACL) へのこの承認進入ルールが必要な場合もあります。

ステップ 7：クライアント VPN のエンドポイント設定ファイルをダウンロードします

設定ファイルをダウンロードするには

1. 「Client VPN エンドポイント設定ファイルのダウンロード」にある Client VPN 設定ファイルをダウンロードするには、以下の簡単な手順に従ってください。
2. このステップでは、クライアント VPN エンドポイントの DNS 名の前に文字列を追加するように求められます。例を示します。

   - Example エンドポイント DNS 名

     クライアント VPN エンドポイントの DNS 名が次のようになっている場合

     ```
     remote cvpn-endpoint-0909091212aaee1.prod.clientvpn.us-west-1.amazonaws.com 443
     ```

     クライアント VPN エンドポイントを識別する文字列を次のように追加できます。

     ```
     remote mwaavpn.cvpn-endpoint-0909091212aaee1.prod.clientvpn.us-
     west-1.amazonaws.com 443
     ```

3. これらの手順では、クライアント証明書の内容を新しい <cert></cert> のタグの間に追加し、プライベートキーの内容を新しい <key></key> のタグの間に追加するように求められます。例を示します。

   a. コマンドプロンプトを開き、ディレクトリをクライアント証明書とプライベートキーの場所に変更します。

   b. Example macOS client1.domain.tld.crt

      macOS で client1.domain.tld.crt ファイルの内容を表示するには、cat client1.domain.tld.crt を使用できます。

      ターミナルから値をコピーし、downloaded-client-config.ovpn に貼り付けます。以下のようになります。

      ```
      ZZZ1111dddaBBB
      -----END CERTIFICATE-----
      </ca>
      <cert>
      ```

```
-----BEGIN CERTIFICATE-----
YOUR client1.domain.tld.crt
-----END CERTIFICATE-----
</cert>
```

c. Example macOS client1.domain.tld.key

`client1.domain.tld.key` の内容を表示するには、`cat client1.domain.tld.key` を使用できます。

ターミナルから値をコピーし、`downloaded-client-config.ovpn` に貼り付けます。以下のようになります。

```
ZZZ1111dddaBBB
-----END CERTIFICATE-----
</ca>
<cert>
-----BEGIN CERTIFICATE-----
YOUR client1.domain.tld.crt
-----END CERTIFICATE-----
</cert>
<key>
-----BEGIN CERTIFICATE-----
YOUR client1.domain.tld.key
-----END CERTIFICATE-----
</key>
```

ステップ 8: AWS Client VPN に接続する

AWS Client VPN のクライアントは無料で提供されています。コンピュータを AWS Client VPN に直接接続して、エンドツーエンドの VPN 体験をすることができます。

クライアント VPN に接続するには

1. 「AWS Client VPN for Desktop」をダウンロードしてインストールします。
2. AWS Client VPN を開きます。
3. VPN クライアントメニューで [ファイル]、[管理対象プロファイル] を選択します。
4. [プロファイルを追加] を選択し、`downloaded-client-config.ovpn` を選択します。
5. [表示名] にわかりやすい名前を入力します。

6. [プロフィールを追加] を選択し、[完了] を選択します。
7. [Connect] (接続) を選択します。

クライアント VPN に接続したら、Amazon VPC 内のすべてのリソースを表示するには、他の VPN との接続を切断する必要があります。

> **Note**
>
> 接続する前に、クライアントを終了して再起動する必要がある場合があります。

次のステップ

- Amazon Managed Workflows for Apache Airflowを使い始める で Amazon MWAA 環境を作成する方法を学びます。クライアント VPN と同じ AWS のリージョンに、クライアント VPN と同じ VPC、プライベートサブネット、セキュリティグループを使用して環境を作成する必要があります。

チュートリアル: Linux 踏み台ホストを使用したプライベートネットワークアクセスの設定

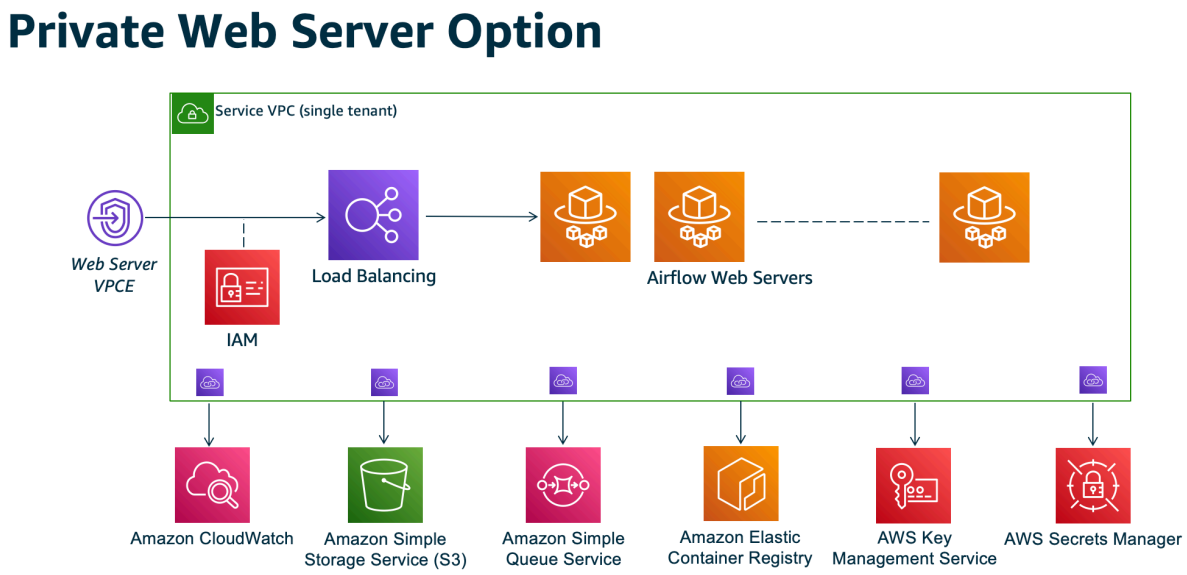
このチュートリアルでは、Amazon Managed Workflows for Apache Airflow 環境用に、コンピュータから Apache Airflow ウェブサーバーへの SSH トンネルを作成する手順を説明します。Amazon MWAA 環境が既に作成されていることを前提としています。セットアップが完了すると、Linux 踏み台ホストはジャンプサーバーとして機能し、コンピュータから VPC 内のリソースに安全に接続できるようになります。次に、SOCKS プロキシ管理アドオンを使用してブラウザのプロキシ設定を制御し、Apache Airflow UI にアクセスします。

セクション

- ステップ 2: ssh トンネルを作成する
- ステップ 3:踏み台セキュリティグループをインバウンドルールとして設定する
- ステップ 4: Apache Airflow URL をコピーします。
- ステップ 5: プロキシ設定を行う
- ステップ 6: Apache Airflow UI を開きます。
- 次のステップ

プライベートネットワーク

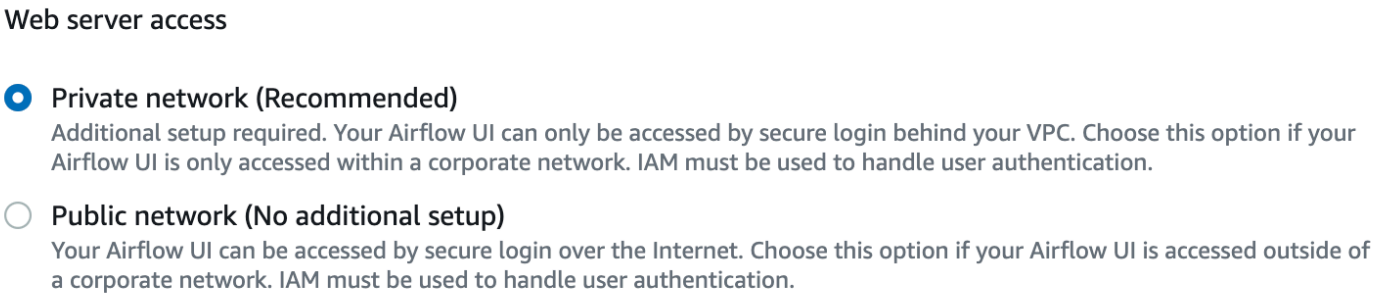
このチュートリアルでは、Apache Airflow ウェブサーバーの [プライベートネットワーク] アクセスモードを選択していることを前提としています。



プライベートネットワークアクセスモードは、Apache Airflow UI へのアクセスを、「お使いの環境のIAM ポリシー」へのアクセスが許可されている Amazon VPC 内のユーザーに制限します。

プライベートなウェブサーバーアクセスを持つ環境を作成する場合、すべての依存関係を Python Wheel アーカイブ (`.whl`) にパッケージ化し、その後、`requirements.txt` で `.whl` を参照する必要があります。wheel を使用して依存関係をパッケージ化およびインストールする手順については、「Python wheel を使用した依存関係の管理」を参照してください。。

以下の画像は、Amazon MWAA コンソールの [プライベートネットワーク] オプションの場所を示しています。



ユースケース

このチュートリアルは、Amazon MWAA 環境を作成した後でも使用できます。環境と同じ Amazon VPC、VPC セキュリティグループ、パブリックサブネットを使用する必要があります。

開始する前に

1. ユーザー許可を確認してください。 AWS Identity and Access Management (IAM) のアカウントには、VPC リソースを作成および管理するための十分なアクセス許可があることを確認してください。
2. Amazon MWAA VPC を使用してください。このチュートリアルでは、踏み台ホストを既存の VPC に関連付けることを想定としています。Amazon VPC は Amazon MWAA 環境と同じリージョンにあり、「VPCネットワークを作成する」で定義されているように 2 つのプライベートサブネットを備えている必要があります。
3. SSH キーを作成します。仮想サーバーに接続するには、Amazon MWAA 環境と同じリージョンに Amazon EC2 SSH キー ([.pem]) を作成する必要があります。SSH キーがない場合は、Amazon EC2 ユーザーガイド」の「キーペアの作成またはインポート」を参照してください。

目的

このチュートリアルでは、以下の作業を行います。

1. 「既存の VPC AWS CloudFormation のテンプレート」を使用して Linux 踏み台ホストインスタンスを作成します。
2. ポート 22 の進入ルールを使用して、踏み台インスタンスのセキュリティグループへのインバウンドトラフィックを承認します。

3. Amazon MWAA 環境のセキュリティグループから踏み台インスタンスのセキュリティグループへのインバウンドトラフィックを承認します。
4. 踏み台インスタンスへの SSH トンネルを作成します。
5. Firefox ブラウザの FoxyProxy アドオンをインストールして設定し、Apache Airflow UI を表示します。

ステップ 1:踏み台インスタンスを作成する

次のセクションでは、 AWS CloudFormation コンソールでAWS CloudFormation 既存の VPC のテンプレートを使用して linux 踏み台インスタンスを作成する手順について説明します。

Linux 踏み台ホストを作成するには

1. AWS CloudFormation コンソールでデプロイクイックスタートページを開きます。
2. ナビゲーションバーのリージョンセレクターを使用して、Amazon MWAA 環境と同じ AWS リージョンを選択します。
3. [次へ] をクリックします。
4. [スタック名] テキストフィールドに、`mwaa-linux-bastion` などの名前を入力します。
5. [パラメーター]、[ネットワーク設定] ペインで、以下のオプションを選択します。
    a. Amazon MWAA 環境の [VPC ID]を選択してください。
    b. Amazon MWAA 環境の [パブリックサブネット 1 ID] を選択します。
    c. Amazon MWAA 環境の [パブリックサブネット 2 ID] を選択します。
    d. [許可された踏み台外部アクセス CIDR] に、可能な限り狭いアドレス範囲 (内部 CIDR 範囲など) を入力します。

    > **Note**
    > 範囲を特定する最も簡単な方法は、パブリックサブネットと同じ CIDR 範囲を使用することです。例えば、VPCネットワークを作成するページの AWS CloudFormation テンプレート内のパブリックサブネットは `10.192.10.0/24`と です`10.192.11.0/24`。

6. [Amazon EC2 設定] ペインで、以下を選択します。
    a. [キーペア名] のドロップダウンリストで SSH キーを選択します。

b. [踏み台ホスト名] に名前を入力します。

c. [TCPフォワーディング] には [true] を選択します。

> **Warning**
>
> このステップでは TCP 転送を [true] に設定する必要があります。そうしないと、次のステップで SSH トンネルを作成できません。

7. [次へ]、[次へ] を選択します。

8. アクノレッジメントを選択し、[スタックを作成] を選択します。

Linux 踏み台ホストのアーキテクチャの詳細については、AWS 「クラウド上の Linux 踏み台ホスト: アーキテクチャ」を参照してください。

ステップ 2: ssh トンネルを作成する

Linux の踏み台への ssh トンネルを作成する方法については、以下のステップで示します。SSH トンネルは、ローカル IP アドレスから Linux 踏み台へのリクエストを受信するため、Linux 踏み台の TCP フォワーディングは前のステップで `true` に設定されました。

macOS/Linux

コマンドラインでトンネルを作成するには

1. Amazon EC2 のコンソールで「インスタンス」ページを開きます。
2. インスタンスを選択します。
3. [Public IPv4 DNS] のアドレスをコピーします。例えば `ec2-4-82-142-1.compute-1.amazonaws.com` です。
4. コマンドプロンプトで、SSH キーが保存されているディレクトリに移動します。
5. 次のコマンドを実行して、踏み台インスタンスに ssh を使用して接続します。サンプル値を `mykeypair.pem` の SSH キー名に置き換えてください。

```
ssh -i mykeypair.pem -N -D 8157 ec2-user@YOUR_PUBLIC_IPV4_DNS
```

Windows (PuTTY)

PuTTY を使用してトンネルを作成するには

1. Amazon EC2 のコンソールで「インスタンス」ページを開きます。
2. インスタンスを選択します。
3. [Public IPv4 DNS] のアドレスをコピーします。例えば `ec2-4-82-142-1.compute-1.amazonaws.com` です。
4. 「PuTTY」を開き、[セッション] を選択します。
5. [ホスト名] にホスト名を ec2-user@ *YOUR_PUBLIC_IPV4_DNS* と入力し、[ポート] を 22 と入力します。
6. [SSH] タブを展開し、[認証] を選択します。[認証用のプライベートキーファイル] で、ローカルの「ppk」ファイルを選択します。
7. SSH で [トンネル] タブを選択し、[動的] オプションと [自動] オプションを選択します。
8. [送信元ポート] に 8157 ポート (またはその他の未使用のポート) を追加し、[送信先] ポートは空白のままにします。[追加] を選択します。
9. [セッション] タブを選択し、セッション名を入力します。例えば「`SSH Tunnel`」のようにです。
10. [保存]、[開く] を選択します。

> **Note**
>
> パブリックキーのパスフレーズの入力が必要な場合があります。

> **Note**
>
> `Permission denied (publickey)` エラーが表示された場合は、AWSSupport-TroubleshootSSH ツールを使用し、このオートメーションを実行する (コンソール) を選択して SSH セットアップのトラブルシューティングを行うことをお勧めします。

ステップ 3:踏み台セキュリティグループをインバウンドルールとして設定する

サーバーへのアクセスとサーバーからの通常のインターネットアクセスは、それらのサーバーにアタッチされた特別なメンテナンスセキュリティグループで許可されています。以下の手順では、踏み台セキュリティグループを環境の VPC セキュリティグループへのインバウンドトラフィックソースとして設定する方法について説明します。

1. Amazon MWAA コンソールで「環境ページ」を開きます。
2. 環境を選択します。
3. [ネットワーク] ペインで [VPC セキュリティグループ] を選択します。
4. [Edit inbound rules] (インバウンドルールの編集) を選択します。
5. [ルールを追加] を選択します。
6. [ソース] ドロップダウンリストから VPC セキュリティグループ ID を選択します。
7. 残りのオプションは空白のままにするか、デフォルト値に設定します。
8. [Save Rules] (ルールの保存) を選択します。

ステップ 4: Apache Airflow URL をコピーします。

次のステップでは、Amazon MWAA コンソールを開いて URL を Apache Airflow UI にコピーする方法について説明します。

1. Amazon MWAA コンソールで「環境ページ」を開きます。
2. 環境を選択します。
3. [Airflow UI] の URL をコピーして、以降のステップで使用できるようにします。

ステップ 5: プロキシ設定を行う

ダイナミックポートフォワーディングによる SSH トンネルを使用する場合、SOCKS プロキシ管理アドオンを使用して、ブラウザでプロキシ設定を管理する必要があります。例えば、Chromium `--proxy-server`の機能を使用してブラウザセッションを開始したり、Mozilla FireFox ブラウザで FoxyProxy 拡張機能を使用したりできます。

オプション 1： ローカルポート転送を使用して SSH トンネルをセットアップします。

SOCKS プロキシを使いたくない場合は、ローカルポート転送を使って SSH トンネルをセットアップすることができます。次のコマンド例では、ローカルポート 8157 でトラフィックを転送して Amazon EC2 ResourceManagerウェブインターフェイスにアクセスします。

1. 新しいコマンドプロンプトウィンドウを開きます。
2. 以下のコマンドを入力し、SSH トンネルを開きます。

```
ssh -i mykeypair.pem -N -L 8157:YOUR_VPC_ENDPOINT_ID-
vpce.YOUR_REGION.airflow.amazonaws.com:443
 ubuntu@YOUR_PUBLIC_IPV4_DNS.YOUR_REGION.compute.amazonaws.com
```

   -L はローカルポート転送の使用を意味し、ノードのローカルウェブサーバーの特定リモートポートへのデータ転送に使用するローカルポートを指定できます。

3. ブラウザに `http://localhost:8157/` を入力してください。

> **Note**
>
> `https://localhost:8157/` を使用する必要がある場合があります。

オプション 2: コマンドラインによるプロキシ

ほとんどのウェブブラウザでは、コマンドラインまたは設定パラメータを使用してプロキシを設定できます。たとえば、Chromium では、次のコマンドを使用してブラウザを起動できます。

```
chromium --proxy-server="socks5://localhost:8157"
```

これにより、前のステップで作成した SSH トンネルを使用してリクエストをプロキシするブラウザセッションが開始されます。プライベート Amazon MWAA 環境 URL (https:// を使用) は、次のように開くことができます。

```
https://YOUR_VPC_ENDPOINT_ID-vpce.YOUR_REGION.airflow.amazonaws.com/home.
```

オプション 3: for FoxyProxy Mozilla Firefox を使用するプロキシ

次の例は、Mozilla Firefox. FoxyProxy provides の FoxyProxy 標準 (バージョン 7.5.1) 設定を示しています。これにより、Apache Airflow UI で使用されるドメインに対応するパターンと一致する URL についてプロキシサーバーを使用できます。

1. Firefox で、FoxyProxy 標準拡張ページを開きます。
2. [Firefox に追加] を選択します。
3. [追加] を選択します。
4. ブラウザのツールバー FoxyProxy のアイコンを選択し、オプション を選択します。
5. 次のコードをコピーし、`mwaa-proxy.json` という名前でローカルに保存します。*YOUR_HOST_NAME* のサンプル値を [Apache Airflow URL] に置き換えてください。

```
{
  "e0b7kh1606694837384": {
    "type": 3,
    "color": "#66cc66",
    "title": "airflow",
    "active": true,
    "address": "localhost",
    "port": 8157,
    "proxyDNS": false,
    "username": "",
    "password": "",
    "whitePatterns": [
      {
        "title": "airflow-ui",
        "pattern": "YOUR_HOST_NAME",
        "type": 1,
        "protocols": 1,
        "active": true
      }
    ],
    "blackPatterns": [],
    "pacURL": "",
    "index": -1
  },
  "k20d21508277536715": {
    "active": true,
    "title": "Default",
    "notes": "These are the settings that are used when no patterns match a URL.",
```

```
    "color": "#0055E5",
    "type": 5,
    "whitePatterns": [
      {
        "title": "all URLs",
        "active": true,
        "pattern": "*",
        "type": 1,
        "protocols": 1
      }
    ],
    "blackPatterns": [],
    "index": 9007199254740991
  },
  "logging": {
    "active": true,
    "maxSize": 500
  },
  "mode": "patterns",
  "browserVersion": "82.0.3",
  "foxyProxyVersion": "7.5.1",
  "foxyProxyEdition": "standard"
}
```

6. FoxyProxy 6.0 以降の設定のインポートペインで、設定のインポートを選択し、`mwaa-proxy.json`ファイルを選択します。
7. [OK] をクリックします。

ステップ 6: Apache Airflow UI を開きます。

Apache Airflow UI を開く方法については、以下のステップで示します。

1. Amazon MWAA コンソールで「環境ページ」を開きます。
2. [Airflow UI を開く] を選択します。

次のステップ

- Apache Airflow CLI コマンドリファレンス の踏み台ホストへの SSH トンネルで Airflow CLI コマンドを実行する方法について説明します。

- DAG コードを Amazon S3 バケットにアップロードする方法については、DAG の追加と更新 で示します。

チュートリアル: Amazon MWAA ユーザーのアクセスを DAG のサブセットに制限する

Amazon MWAA は IAM プリンシパルを 1 つ以上の Apache Airflow の「デフォルトロール」にマッピングすることで、環境へのアクセスを管理します。次のチュートリアルでは、個々の Amazon MWAA ユーザーが特定の DAG または一連の DAG の表示と操作のみに制限する方法を示しています。

> **Note**
>
> このチュートリアルのステップは、IAM ロールを想定できる限り、フェデレーションアクセスを使用して完了できます。

トピック

前提条件

このチュートリアルのステップを完了するには、以下が必要です。

- 「複数の DAG を使用する Amazon MWAA 環境」
- アクセスAdministratorAccess許可Adminを持つ IAM プリンシパル、および DAG アクセスを制限できるプリンシパルMWAAUserとしての IAM ユーザー 。管理者ロールの詳細については、「IAM ユーザーガイド」の「管理者ジョブ機能」を参照してください。

> **Note**
>
> アクセス許可ポリシーを IAM ユーザーに直接アタッチしないでください。Amazon MWAA リソースに一時的にアクセスするために引き受けることができる IAM ロールの設定をお勧めします。

- AWS Command Line Interface バージョン 2 がインストールされました。

ステップ 1:デフォルトの **Public** Apache Airflow ロールを使用して、IAM プリンシパルに Amazon MWAA ウェブサーバーへのアクセス権を付与します。

を使用してアクセス許可を付与するには AWS Management Console

1. Admin ロールを使用して AWS アカウントにサインインし、IAM コンソール を開きます。
2. 左側のナビゲーションペインで [ユーザー] を選択し、ユーザーテーブルから Amazon MWAA IAM ユーザーを選択します。
3. ユーザー詳細ページの [概要] で [アクセス許可] タブを選択し、[アクセス許可ポリシー] を選択してカードを展開し、[アクセス許可の追加] を選択します。
4. [権限の付与] セクションで、[既存のポリシーの直接添付] を選択し、[ポリシーの作成] を選択して、独自のカスタム権限ポリシーを作成して添付します。
5. [ポリシーの作成] ページで [JSON] を選択し、次の JSON アクセス権限ポリシーをコピーしてポリシーエディターに貼り付けます。このポリシーは、デフォルトの Public Apache Airflow ロールを持つユーザーにウェブサーバーアクセスを許可します。

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

 "arn:aws:airflow:YOUR_REGION:YOUR_ACCOUNT_ID:role/YOUR_ENVIRONMENT_NAME/Public"
            ]
        }
```

```
    ]
}
```

ステップ 2:新しい Apache Airflow カスタムロールを作成する

Apache Airflow UI を使用して新しいロールを作成するには

1. 管理者の IAM ロールを使用して、「Amazon MWAA コンソール」を開き、環境の Apache Airflow UI を起動します。
2. 上部のナビゲーションペインで [セキュリティ] にカーソルを合わせてドロップダウンリストを開き、[ロールを一覧表示] を選択してデフォルトの Apache Airflow ロールを表示します。
3. ロールリストから [ユーザー] を選択し、ページ上部の [アクション] を選択してドロップダウンを開きます。[ロールをコピー] を選択し、[OK] を確認します。

   > **Note**
   >
   > [Ops] ロールまたは [ビューワー] ロールをコピーして、それぞれ許可するアクセス権を増やしたり減らしたりします。

4. テーブルで作成した新しいロールを探し、[レコードを編集] を選択します。
5. [ロールを編集] ページで以下を実行します。

   - [名前] には、テキストフィールドにロールの新しい名前を入力します。例えば **`Restricted`** です。
   - アクセス許可 のリストで、 `can read on DAGs`と を削除し`can edit on DAGs`、アクセス許可を付与する DAGs のセットの読み取りおよび書き込みアクセス許可を追加します。たとえば、DAG である `example_dag.py` の場合、**`can read on DAG:`*`example_dag`*** と **`can edit on DAG:`*`example_dag`*** を追加してください。

   [保存] を選択します。これで、Amazon MWAA 環境で使用可能な DAG のサブセットへのアクセスを制限する新しいロールができたはずです。これで、既存の Apache Airflow ユーザーにこのロールを割り当てることができるようになりました。

ステップ 3:作成したロールを Amazon MWAA ユーザーに割り当てます。

新しいロールを割り当てるには

1. MWAAUser の アクセス資格情報を使用して、以下の CLI コマンドを実行して環境のウェブサーバー URL を取得してください。

   ```
   $ aws mwaa get-environment --name YOUR_ENVIRONMENT_NAME | jq
    '.Environment.WebserverUrl'
   ```

   成功すれば、次のような出力が表示されます。

   ```
   "ab1b2345-678a-90a1-a2aa-34a567a8a901.c13.us-west-2.airflow.amazonaws.com"
   ```

2. にMWAAUserサインインしたら AWS Management Console、新しいブラウザウィンドウを開き、次の URI にアクセスします。Webserver-URL を自分の情報に置き換えます。

   ```
   https://<Webserver-URL>/home
   ```

   成功した場合、まだ MWAAUser が Apache Airflow UI へのアクセス許可を得ていないため、Forbidden のエラーページが表示されます。

3. にAdminサインインしたら AWS Management Console、Amazon MWAA コンソールを再度開き、環境の Apache Airflow UI を起動します。

4. UI ダッシュボードから [セキュリティ] ドロップダウンを展開し、今度は [ユーザーを一覧表示] を選択します。

5. ユーザーテーブルで新しい Apache Airflow ユーザーを探し、[レコードを編集] を選択します。ユーザーの名前は、次のパターンで IAM ユーザーの名前と一致します：user/*mwaa-user*。

6. [ユーザーの編集] ページの [ロール] セクションで、作成した新しいカスタムロールを追加し、[保存] を選択します。

   > **Note**
   >
   > [姓] フィールドは必須ですが、スペースがあれば十分です。

   IAM Public プリンシパルは Apache Airflow UI MWAAUser にアクセスする権限を付与し、新しいロールは DAG を確認するために必要な追加の権限を付与します。

> **Important**
>
> Apache Airflow UI を使用して追加された IAM によって承認されていない 5 つのデフォルトロール (`Admin` など) は、次回のユーザーログイン時に削除されます。

次のステップ

- Amazon MWAA 環境へのアクセス管理の詳細と、環境ユーザーに使用できる JSON IAM ポリシーのサンプルを確認するには、「the section called “Amazon MWAA 環境へのアクセス”」を参照してください。

関連リソース

- 「アクセスコントロール」(Apache Airflow ドキュメント) — デフォルトの Apache Airflow ロールの詳細については、Apache Airflow ドキュメンテーションウェブサイトをご覧ください。

チュートリアル: Amazon MWAA での独自の環境エンドポイントの管理を自動化する

AWS Organizations を使用してリソースを共有する複数のAWSアカウントを管理する場合、Amazon MWAA では独自の Amazon VPC エンドポイントを作成および管理できます。つまり、環境に必要なリソースにのみアクセスできるようにする、より厳格なセキュリティポリシーを使用できます。

共有 Amazon VPC で環境を作成すると、メインの Amazon VPC を所有するアカウント (所有者) は、Amazon MWAA に必要な 2 つのプライベートサブネットを、同じ組織に属する他のアカウント (参加者) と共有します。これらのサブネットを共有する参加者アカウントは、共有 VPC 内の環境を表示、作成、変更、削除できます。

共有 VPC またはポリシー制限のある Amazon VPC で環境を作成すると、Amazon MWAA はまずサービス VPC リソースを作成し、次に最大 72 時間 `PENDING`状態に入ります。

環境ステータスが から `CREATING`に変わると`PENDING`、Amazon MWAA は 状態の変更に関する Amazon EventBridge 通知を送信します。これにより、所有者アカウントは、Amazon MWAA コンソールまたは API からのエンドポイントサービス情報に基づいて、またはプログラムによって、参加者に代わって必要なエンドポイントを作成できます。以下では、Lambda 関数と Amazon MWAA

状態変更通知をリッスンする EventBridge ルールを使用して、新しい Amazon VPC エンドポイントを作成します。

ここでは、環境と同じ Amazon VPC に新しいエンドポイントを作成します。共有 Amazon VPC をセットアップするには、所有者アカウントと参加者アカウントの Amazon MWAA 環境に EventBridge ルールと Lambda 関数を作成します。

トピック

前提条件

このチュートリアルの手順を完了するには、以下が必要です。

- ...

Amazon VPC を作成する

次のAWS CloudFormationテンプレートとAWS CLIコマンドを使用して、新しい Amazon VPC を作成します。テンプレートは Amazon VPC リソースを設定し、特定のキューへのアクセスを制限するようにエンドポイントポリシーを変更します。

1. AWS CloudFormation テンプレート をダウンロードし、`.yml`ファイルを解凍します。
2. 新しいコマンドプロンプトウィンドウで、テンプレートを保存したフォルダに移動し、`create-stack` を使用してスタックを作成します。`--template-body` フラグは、テンプレートへのパスを指定します。

```
$ aws cloudformation create-stack --stack-name stack-name --template-body file://
cfn-vpc-private-network.yml
```

次のセクションでは、Lambda 関数を作成します。

Lambda 関数を作成する

次の Python コードと IAM JSON ポリシーを使用して、新しい Lambda 関数と実行ロールを作成します。この関数は、プライベート Apache Airflow ウェブサーバーと Amazon SQS キューの Amazon VPC エンドポイントを作成します。Amazon MWAA は Amazon SQS を使用して、環境のスケーリング時に複数のワーカー間で Celery を使用してタスクをキューに入れます。

1. Python 関数コード をダウンロードします。
2. IAM アクセス許可ポリシー をダウンロードし、ファイルを解凍します。
3. コマンドプロンプトを開き、JSON アクセス許可ポリシーを保存したフォルダに移動します。IAM `create-role` コマンドを使用して新しいロールを作成します。

   ```
   $ aws iam create-role --role-name function-role \
   --assume-role-policy-document file://lambda-mwaa-vpce-policy.json
   ```

   AWS CLI レスポンスのロール ARN を書き留めます。次のステップでは、ARN を使用して、この新しいロールを関数の実行ロールとして指定します。
4. 関数コードを保存したフォルダに移動し、 コマンドを使用して`create-function`新しい関数を作成します。

   ```
   $ aws lambda create-function --function-name mwaa-vpce-lambda \
   --zip-file file://mwaa-lambda-shared-vpc.zip --runtime python3.8 --role
    arn:aws:iam::123456789012:role/function-role --handler lambda_handler
   ```

   AWS CLI レスポンスからの関数 ARN を書き留めます。次のステップでは、ARN を指定して、関数を新しい EventBridge ルールのターゲットとして設定します。

次のセクションでは、環境が PENDING状態になったときにこの関数を呼び出す EventBridge ルールを作成します。

EventBridge ルールを作成する

以下を実行して、Amazon MWAA 通知をリッスンし、新しい Lambda 関数をターゲットとする新しいルールを作成します。

1. コマンドを使用して EventBridge `put-rule`新しい EventBridge ルールを作成します。

   ```
   $ aws events put-rule --name "mwaa-lambda-rule" \
   ```

```
--event-pattern "{\"source\":[\"aws.airflow\"],\"detail-type\":[\"MWAA Environment
 Status Change\"]}"
```

イベントパターンは、環境ステータスが変更されるたびに Amazon MWAA が送信する通知をリッスンします。

```
{
  "source": ["aws.airflow"],
  "detail-type": ["MWAA Environment Status Change"]
}
```

2. put-targets コマンドを使用して、Lambda 関数を新しいルールのターゲットとして追加します。

```
$ aws events put-targets --rule "mwaa-lambda-rule" \
--targets "Id"="1","Arn"="arn:aws::lambda:region:123456789012:function:mwaa-vpce-
lambda"
```

カスタマー管理の Amazon VPC エンドポイントを使用して、新しい Amazon MWAA 環境を作成する準備が整いました。

Amazon MWAA 環境を作成する

Amazon MWAA コンソールを使用して、カスタマー管理の Amazon VPC エンドポイントで新しい環境を作成します。

1. Amazon MWAA コンソールを開き、環境の作成を選択します。
2. 名前には一意の名前を入力します。
3. Airflow バージョン では、最新バージョンを選択します。
4. 環境dags/で使用する などの Amazon S3 バケットと DAGsフォルダ を選択し、次へ を選択します。
5. 詳細設定の構成ページで、次の操作を行います。
   a. Virtual Private Cloud で、前のステップ で作成した Amazon VPC を選択します。
   b. ウェブサーバーアクセス で、パブリックネットワーク (インターネットアクセス) を選択します。

c. セキュリティグループ で、 で作成したセキュリティグループを選択しますAWS CloudFormation。前のステップのAWS PrivateLinkエンドポイントのセキュリティグループは自己参照であるため、環境に対して同じセキュリティグループを選択する必要があります。

d. エンドポイント管理 で、カスタマーマネージドエンドポイント を選択します。

6. 残りのデフォルト設定のままにして、次へを選択します。

7. 選択内容を確認し、環境の作成を選択します。

> **Tip**
>
> 新しい環境の設定の詳細については、「Amazon MWAA の開始方法」を参照してください。

環境が の場合`PENDING`、Amazon MWAA はルールに設定したイベントパターンに一致する通知を送信します。このルールは Lambda 関数を呼び出します。関数は通知イベントを解析し、ウェブサーバーと Amazon SQS キューに必要なエンドポイント情報を取得します。次に、Amazon VPC にエンドポイントを作成します。

エンドポイントが使用可能になると、Amazon MWAA は環境の作成を再開します。準備ができたら、環境ステータスが に変わり`AVAILABLE`、Amazon MWAA コンソールを使用して Apache Airflow ウェブサーバーにアクセスできます。

Amazon Managed Workflows for Apache Airflow

このガイドには、Amazon Managed Workflows for Apache Airflow 環境で使用できる DAG やカスタムプラグインなどのコードサンプルが含まれています。 AWS サービスで Apache Airflow を使用するその他の例については、Apache Airflow GitHub リポジトリの dags ディレクトリを参照してください。

サンプル

- DAG を使用して CLI に変数をインポートする
- SSHOperator を使用して SSH 接続の作成
- Apache Airflow Snowflake 接続において、AWS Secrets Manager でのシークレットキーを使用する
- DAG を使用して CloudWatch にカスタムメトリクスを書き込む
- Amazon MWAA 環境でのAurora PostgreSQL データベースのクリーンアップ
- Amazon S3 の CSV ファイルへの環境メタデータのエクスポート
- Apache Airflow 変数の AWS Secrets Manager におけるシークレットキーの使用
- AWS Secrets Manager の Apache Airflow 接続でのシークレットキーの使用
- Oracle でのカスタムプラグインの作成
- ランタイム環境変数を生成するカスタムプラグインを作成します
- Amazon MWAA での DAG のタイムゾーンの変更
- CodeArtifact トークンのリフレッシュ
- Apache Hive と Hadoop を使ったカスタムプラグインの作成
- Apache Airflow Python VirtualEnv オペレータ用のカスタムプラグインを作成する
- Lambda 関数を使用して DAG を呼び出す
- さまざまな Amazon MWAA 環境での DAG の呼び出し
- Amazon RDS for Microsoft SQL Server で Amazon MWAA を使用する
- Amazon MWAA と Amazon EMR を併用する
- Amazon EKS での Amazon MWAA の使用
- ECSOperator を使用して Amazon ECS に接続します。
- Amazon MWAA での dbt の使用
- AWS ブログとチュートリアル

DAG を使用して CLI に変数をインポートする

次のサンプルコードでは、Amazon Managed Workflows for Apache Airflow の CLI を使用して変数をインポートしています。

トピック

バージョン

- このページのコード例は、「Python 3.10」で Apache Airflow v2 以上と共に使用可能です。

前提条件

- このページのコード例を使用する場合、追加のアクセス許可は必要ありません。

許可

AWS アカウントには `AmazonMWAAAirflowCliAccess` ポリシーへのアクセス権が必要です。詳細については、「Apache Airflow CLI ポリシー: AmazonMWAAAirflowCli アクセス」を参照してください。

依存関係

- このコード例を Apache Airflow v2 で使用する場合、追加の依存関係は必要ありません。このコードでは、ご使用の環境にある「Apache Airflow v2 のベースインストール」を使用します。

コードサンプル

以下のサンプルコードは、3 つの入力を受け取ります： Amazon MWAA 環境の名前 (mwaa_env で)、環境の AWS リージョン (aws_region で)、インポートしたい変数が含まれているローカルファイル (var_file で)。

```
import boto3
import json
import requests
import base64
import getopt
import sys

argv = sys.argv[1:]
mwaa_env=''
aws_region=''
var_file=''

try:
    opts, args = getopt.getopt(argv, 'e:v:r:', ['environment', 'variable-
file','region'])
    #if len(opts) == 0 and len(opts) > 3:
    if len(opts) != 3:
        print ('Usage: -e MWAA environment -v variable file location and filename -r
 aws region')
    else:
        for opt, arg in opts:
            if opt in ("-e"):
                mwaa_env=arg
            elif opt in ("-r"):
                aws_region=arg
            elif opt in ("-v"):
                var_file=arg

        boto3.setup_default_session(region_name="{}".format(aws_region))
        mwaa_env_name = "{}".format(mwaa_env)

        client = boto3.client('mwaa')
        mwaa_cli_token = client.create_cli_token(
            Name=mwaa_env_name
        )

        with open ("{}".format(var_file), "r") as myfile:
```

```
            fileconf = myfile.read().replace('\n', '')

        json_dictionary = json.loads(fileconf)
        for key in json_dictionary:
            print(key, " ", json_dictionary[key])
            val = (key + " " + json_dictionary[key])
            mwaa_auth_token = 'Bearer ' + mwaa_cli_token['CliToken']
            mwaa_webserver_hostname = 'https://{0}/aws_mwaa/
cli'.format(mwaa_cli_token['WebServerHostname'])
            raw_data = "variables set {0}".format(val)
            mwaa_response = requests.post(
                mwaa_webserver_hostname,
                headers={
                    'Authorization': mwaa_auth_token,
                    'Content-Type': 'text/plain'
                    },
                data=raw_data
                )
            mwaa_std_err_message = base64.b64decode(mwaa_response.json()
['stderr']).decode('utf8')
            mwaa_std_out_message = base64.b64decode(mwaa_response.json()
['stdout']).decode('utf8')
            print(mwaa_response.status_code)
            print(mwaa_std_err_message)
            print(mwaa_std_out_message)

except:
    print('Use this script with the following options: -e MWAA environment -v variable
 file location and filename -r aws region')
    print("Unexpected error:", sys.exc_info()[0])
    sys.exit(2)
```

次のステップ

- この例の DAG コードを Amazon S3 バケットの dags フォルダにアップロードする方法については、「DAG の追加と更新」を参照してください。

SSHOperator を使用して SSH 接続の作成

次の例では、Amazon Managed Workflows for Apache Airflow 環境からリモートの Amazon EC2 インスタンスに接続するために SSHOperator をどのように使用できるかを示しています。同様の方法で、SSH アクセスを持つ任意のリモートインスタンスに接続できます。

次の例では、SSH シークレットキー (.pem)を Amazon S3 の環境の dags ディレクトリにアップロードします。次に、requirements.txt を使用して必要な依存関係をインストールし、UI で新しい Apache Airflow 接続を作成します。最後に、リモートインスタンスへの SSH 接続を作成する DAG を作成します。

トピック

Version

- このページのコード例は、「Python 3.10」で Apache Airflow v2 以上と共に使用可能です。

前提条件

このページのサンプルコードを使用するには、以下が必要です。

- 「Amazon MWAA 環境」。
- SSH シークレットキー。このコードサンプルは、Amazon MWAA 環境と同じリージョンに Amazon EC2 インスタンスと .pem があることを前提としています。キーがない場合は、Amazon EC2 ユーザーガイド」の「キーペアの作成またはインポート」を参照してください。

アクセス許可

- このページのコード例を使用する場合、追加のアクセス許可は必要ありません。

要件

次のパラメータを requirements.txt に追加して、ウェブサーバーに apache-airflow-providers-ssh パッケージをインストールしてください。環境が更新され、Amazon MWAA が依存関係を正常にインストールすると、UI に新しい [SSH] 接続タイプが表示されます。

```
-c https://raw.githubusercontent.com/apache/airflow/constraints-Airflow-version/
constraints-Python-version.txt
apache-airflow-providers-ssh
```

> **Note**
>
> -c は requirements.txt 内での制約 URL を定義します。これにより、Amazon MAA はお客様の環境に合った正しいパッケージバージョンをインストールできます。

シークレットキーを Amazon S3 にコピーする

次の AWS Command Line Interface コマンドを使用して、.pemキーを Amazon S3 の環境のdagsディレクトリにコピーします。

```
$ aws s3 cp your-secret-key.pem s3://your-bucket/dags/
```

Amazon MWAA は、.pem キーを含む dags のコンテンツをローカルの /usr/local/airflow/dags/ ディレクトリにコピーすることで、Apache Airflow はキーにアクセスできます。

新しい Apache Airflow 接続の作成

Apache Airflow UI を使用して新しい SSH 接続を作成するには

1. Amazon MWAA コンソールで「環境ページ」を開きます。
2. 環境のリストで、ご使用の環境に合った [Open Airflow UI] を選択します。
3. Apache Airflow UI ページで、上部のナビゲーションバーから [管理] を選択してドロップダウンリストを展開し、[接続] を選択します。

4. [接続リスト] ページで [+] を選択するか、[新規レコードを追加] ボタンをクリックして新しい接続を追加します。
5. [接続の追加] ページで、以下の情報を追加します。
   a. [接続 ID] には **ssh_new** と入力します。
   b. [接続タイプ] では、ドロップダウンリストから [SSH] を選択します。

      > **Note**
      >
      > [SSH] 接続タイプがリストに表示されない場合、Amazon MWAA は必要な `apache-airflow-providers-ssh` パッケージをインストールしていない可能性があります。このパッケージを含むように `requirements.txt` ファイルを更新してから、もう一度試してください。

   c. [ホスト] には、接続する Amazon EC2 インスタンスの IP アドレスを入力します。例えば **12.345.67.89** です。
   d. [ユーザー名] に、Amazon EC2 インスタンスに接続する場合は **ec2-user** を入力します。Apache Airflow に接続させたいリモートインスタンスのタイプによって、ユーザー名は異なる場合があります。
   e. [抽出] には、以下のキーと値のペアを JSON 形式で入力します。

      ```
      { "key_file": "/usr/local/airflow/dags/your-secret-key.pem" }
      ```

      このキーと値のペアは、Apache Airflow に対してシークレットキーをローカルの `/dags` ディレクトリから探すように指示します。

コードサンプル

次の DAG は `SSHOperator` を使用してターゲットの Amazon EC2 インスタンスに接続し、その後 `hostname` Linux コマンドを実行してインスタンスの名前を表示します。DAG を変更して、リモートインスタンスで任意のコマンドまたはスクリプトを実行できます。

1. ターミナルを開き、DAG コードが保存されているディレクトリに移動します。例:

   ```
   cd dags
   ```

2. 以下のコードサンプルの内容をコピーし、ローカルに `ssh.py` として保存します。

```
from airflow.decorators import dag
from datetime import datetime
from airflow.providers.ssh.operators.ssh import SSHOperator

@dag(
    dag_id="ssh_operator_example",
    schedule_interval=None,
    start_date=datetime(2022, 1, 1),
    catchup=False,
    )
def ssh_dag():
    task_1=SSHOperator(
        task_id="ssh_task",
        ssh_conn_id='ssh_new',
        command='hostname',
    )

my_ssh_dag = ssh_dag()
```

3. 次の AWS CLI コマンドを実行して DAG を環境のバケットにコピーし、Apache Airflow UI を使用して DAG をトリガーします。

```
$ aws s3 cp your-dag.py s3://your-environment-bucket/dags/
```

4. 成功した場合、ssh_operator_example DAG 内の ssh_task タスクログで次のような出力が表示されます。

```
[2022-01-01, 12:00:00 UTC] {{base.py:79}} INFO - Using connection to: id: ssh_new.
 Host: 12.345.67.89, Port: None,
Schema: , Login: ec2-user, Password: None, extra: {'key_file': '/usr/local/airflow/
dags/your-secret-key.pem'}
[2022-01-01, 12:00:00 UTC] {{ssh.py:264}} WARNING - Remote Identification Change is
 not verified. This won't protect against Man-In-The-Middle attacks
[2022-01-01, 12:00:00 UTC] {{ssh.py:270}} WARNING - No Host Key Verification. This
 won't protect against Man-In-The-Middle attacks
[2022-01-01, 12:00:00 UTC] {{transport.py:1819}} INFO - Connected (version 2.0,
 client OpenSSH_7.4)
[2022-01-01, 12:00:00 UTC] {{transport.py:1819}} INFO - Authentication (publickey)
 successful!
[2022-01-01, 12:00:00 UTC] {{ssh.py:139}} INFO - Running command: hostname
[2022-01-01, 12:00:00 UTC]{{ssh.py:171}} INFO - ip-123-45-67-89.us-
west-2.compute.internal
```

```
[2022-01-01, 12:00:00 UTC] {{taskinstance.py:1280}} INFO - Marking task as SUCCESS.
 dag_id=ssh_operator_example, task_id=ssh_task, execution_date=20220712T200914,
 start_date=20220712T200915, end_date=20220712T200916
```

Apache Airflow Snowflake 接続において、AWS Secrets Manager でのシークレットキーを使用する

以下は、Amazon Managed Workflows for Apache AirflowでApache Airflow Snowflake 接続のシークレットキーを取得するために AWS Secrets Manager を呼び出すサンプルです。AWS Secrets Manager シークレットを使用した Apache Airflow 接続の設定 でのステップを完了していることが前提となります。

トピック

バージョン

- このページのコード例は、「Python 3.10」で Apache Airflow v2 以上と共に使用可能です。

前提条件

このページのサンプルコードを使用するには、以下が必要です。

- Secrets Manager バックエンドを Apache Airflow 構成オプションとして使用する方法は、AWS Secrets Manager シークレットを使用した Apache Airflow 接続の設定 で示されています。
- AWS Secrets Manager シークレットを使用した Apache Airflow 接続の設定 に示すように、Secrets Manager のApache Airflow 接続文字列。

許可

- Secrets Manager の権限 (AWS Secrets Manager シークレットを使用した Apache Airflow 接続の設定 を参照)。

要件

このページのサンプルコードを使用するには、次の依存関係を `requirements.txt` に追加してください。詳細については、「Python 依存関係のインストール」を参照してください。

```
apache-airflow-providers-snowflake==1.3.0
```

コードサンプル

以下の手順では、Secrets Manager を呼び出してシークレットを取得する DAG コードを作成する方法について説明します。

1. コマンドプロンプトで、DAG コードが保存されているディレクトリに移動します。例:

   ```
   cd dags
   ```

2. 以下のコードサンプルの内容をコピーし、ローカルに `snowflake_connection.py` として保存します。

   ```
   """
   Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.

   Permission is hereby granted, free of charge, to any person obtaining a copy of
   this software and associated documentation files (the "Software"), to deal in
   the Software without restriction, including without limitation the rights to
   use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of
   the Software, and to permit persons to whom the Software is furnished to do so.

   THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
   IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS
   FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
   COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER
   IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
   CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
   """
   ```

```
from airflow import DAG
from airflow.providers.snowflake.operators.snowflake import SnowflakeOperator
from airflow.utils.dates import days_ago

snowflake_query = [
    """use warehouse "MY_WAREHOUSE";""",
    """select * from "SNOWFLAKE_SAMPLE_DATA"."WEATHER"."WEATHER_14_TOTAL" limit
 100;""",
]

with DAG(dag_id='snowflake_test', schedule_interval=None, catchup=False,
 start_date=days_ago(1)) as dag:
    snowflake_select = SnowflakeOperator(
        task_id="snowflake_select",
        sql=snowflake_query,
        snowflake_conn_id="snowflake_conn",
    )
```

次のステップ

- この例の DAG コードを Amazon S3 バケットの dags フォルダにアップロードする方法については、「DAG の追加と更新」を参照してください。

DAG を使用して CloudWatch にカスタムメトリクスを書き込む

次のコード例を使用して、PythonOperator を実行して Amazon MWAA 環境の OS レベルのメトリックスを取得する有向非巡回グラフ (DAG) を作成できます。その後、DAG はカスタムメトリクスとしてデータを Amazon CloudWatch に発行します。

OS レベルのカスタムメトリクスにより、環境ワーカーが仮想メモリや CPU などのリソースをどのように利用しているかをさらに把握できます。この情報を使用して、ワークロードに最適な「環境クラス」を選択できます。

トピック

- バージョン
- 前提条件
- 許可
- 依存関係

- コード例

バージョン

- このページのコード例は、「Python 3.10」で Apache Airflow v2 以上と共に使用可能です。

前提条件

このページのコード例を使用するには、以下のものが必要である：

- 「Amazon MWAA 環境」。

許可

- このページのコード例を使用する場合、追加のアクセス許可は必要ありません。

依存関係

- このページのコード例を使用するのに、追加の依存関係は必要ありません。

コード例

1. コマンドプロンプトで、DAG コードが保存されているフォルダーに移動します。例:

```
cd dags
```

2. 次のコード例の内容をコピーし、dag-custom-metrics.py としてローカルに保存します。MWAA-ENV-NAME をご使用の環境名に置き換えてください。

```
from airflow import DAG
from airflow.operators.python_operator import PythonOperator
from airflow.utils.dates import days_ago
from datetime import datetime
import os,json,boto3,psutil,socket

def publish_metric(client,name,value,cat,unit='None'):
    environment_name = os.getenv("MWAA_ENV_NAME")
```

```
    value_number=float(value)
    hostname = socket.gethostname()
    ip_address = socket.gethostbyname(hostname)
    print('writing value',value_number,'to metric',name)
    response = client.put_metric_data(
        Namespace='MWAA-Custom',
        MetricData=[
            {
                'MetricName': name,
                'Dimensions': [
                    {
                        'Name': 'Environment',
                        'Value': environment_name
                    },
                    {
                        'Name': 'Category',
                        'Value': cat
                    },
                    {
                        'Name': 'Host',
                        'Value': ip_address
                    },
                ],
                'Timestamp': datetime.now(),
                'Value': value_number,
                'Unit': unit
            },
        ]
    )
    print(response)
    return response

def python_fn(**kwargs):
    client = boto3.client('cloudwatch')

    cpu_stats = psutil.cpu_stats()
    print('cpu_stats', cpu_stats)

    virtual = psutil.virtual_memory()
    cpu_times_percent = psutil.cpu_times_percent(interval=0)

    publish_metric(client=client, name='virtual_memory_total',
 cat='virtual_memory', value=virtual.total, unit='Bytes')
```

```
    publish_metric(client=client, name='virtual_memory_available',
 cat='virtual_memory', value=virtual.available, unit='Bytes')
    publish_metric(client=client, name='virtual_memory_used', cat='virtual_memory',
 value=virtual.used, unit='Bytes')
    publish_metric(client=client, name='virtual_memory_free', cat='virtual_memory',
 value=virtual.free, unit='Bytes')
    publish_metric(client=client, name='virtual_memory_active',
 cat='virtual_memory', value=virtual.active, unit='Bytes')
    publish_metric(client=client, name='virtual_memory_inactive',
 cat='virtual_memory', value=virtual.inactive, unit='Bytes')
    publish_metric(client=client, name='virtual_memory_percent',
 cat='virtual_memory', value=virtual.percent, unit='Percent')

    publish_metric(client=client, name='cpu_times_percent_user',
 cat='cpu_times_percent', value=cpu_times_percent.user, unit='Percent')
    publish_metric(client=client, name='cpu_times_percent_system',
 cat='cpu_times_percent', value=cpu_times_percent.system, unit='Percent')
    publish_metric(client=client, name='cpu_times_percent_idle',
 cat='cpu_times_percent', value=cpu_times_percent.idle, unit='Percent')

    return "OK"


with DAG(dag_id=os.path.basename(__file__).replace(".py", ""),
 schedule_interval='*/5 * * * *', catchup=False, start_date=days_ago(1)) as dag:
    t = PythonOperator(task_id="memory_test", python_callable=python_fn,
 provide_context=True)
```

3. 以下の AWS CLI コマンドを実行して、DAG を環境のバケットにコピーし、次に Apache Airflow UI を使用して DAG をトリガーします。

```
$ aws s3 cp your-dag.py s3://your-environment-bucket/dags/
```

4. DAG が正常に実行されると、Apache Airflow ログに次のような内容が表示されるはずです。

```
[2022-08-16, 10:54:46 UTC] {{logging_mixin.py:109}} INFO -
 cpu_stats scpustats(ctx_switches=3253992384, interrupts=1964237163,
 soft_interrupts=492328209, syscalls=0)
[2022-08-16, 10:54:46 UTC] {{logging_mixin.py:109}} INFO - writing value
 16024199168.0 to metric virtual_memory_total
[2022-08-16, 10:54:46 UTC] {{logging_mixin.py:109}} INFO - {'ResponseMetadata':
 {'RequestId': 'fad289ac-aa51-46a9-8b18-24e4e4063f4d', 'HTTPStatusCode': 200,
 'HTTPHeaders': {'x-amzn-requestid': 'fad289ac-aa51-46a9-8b18-24e4e4063f4d',
```

```
 'content-type': 'text/xml', 'content-length': '212', 'date': 'Tue, 16 Aug 2022
 17:54:45 GMT'}, 'RetryAttempts': 0}}
[2022-08-16, 10:54:46 UTC] {{logging_mixin.py:109}} INFO - writing value
 14356287488.0 to metric virtual_memory_available
[2022-08-16, 10:54:46 UTC] {{logging_mixin.py:109}} INFO - {'ResponseMetadata':
 {'RequestId': '6ef60085-07ab-4865-8abf-dc94f90cab46', 'HTTPStatusCode': 200,
 'HTTPHeaders': {'x-amzn-requestid': '6ef60085-07ab-4865-8abf-dc94f90cab46',
 'content-type': 'text/xml', 'content-length': '212', 'date': 'Tue, 16 Aug 2022
 17:54:45 GMT'}, 'RetryAttempts': 0}}
[2022-08-16, 10:54:46 UTC] {{logging_mixin.py:109}} INFO - writing value
 1342296064.0 to metric virtual_memory_used
[2022-08-16, 10:54:46 UTC] {{logging_mixin.py:109}} INFO - {'ResponseMetadata':
 {'RequestId': 'd5331438-5d3c-4df2-bc42-52dcf8d60a00', 'HTTPStatusCode': 200,
 'HTTPHeaders': {'x-amzn-requestid': 'd5331438-5d3c-4df2-bc42-52dcf8d60a00',
 'content-type': 'text/xml', 'content-length': '212', 'date': 'Tue, 16 Aug 2022
 17:54:45 GMT'}, 'RetryAttempts': 0}}
...
[2022-08-16, 10:54:46 UTC] {{python.py:152}} INFO - Done. Returned value was: OK
[2022-08-16, 10:54:46 UTC] {{taskinstance.py:1280}} INFO - Marking task as SUCCESS.
 dag_id=dag-custom-metrics, task_id=memory_test, execution_date=20220816T175444,
 start_date=20220816T175445, end_date=20220816T175446
[2022-08-16, 10:54:46 UTC] {{local_task_job.py:154}} INFO - Task exited with return
 code 0
```

Amazon MWAA 環境でのAurora PostgreSQL データベースのクリーンアップ

Amazon Managed Workflows for Apache Airflow は、Apache Airflow メタデータデータベースとして Aurora PostgreSQL データベースを使用します。このデータベースでは、DAG が実行され、タスクインスタンスが保存されます。次のサンプルコードは、Amazon MWAA 環境の専用の Aurora PostgreSQL データベースから定期的にエントリを消去します。

トピック

- バージョン
- 前提条件
- 依存関係
- コードサンプル

バージョン

- このページのコード例は、「Python 3.10」で Apache Airflow v2 以上と共に使用可能です。

前提条件

このページのサンプルコードを使用するには、以下が必要です。

- 「Amazon MWAA 環境」。

依存関係

- このコード例を Apache Airflow v2 で使用する場合、追加の依存関係は必要ありません。このコードでは、ご使用の環境にある「Apache Airflow v2 のベースインストール」を使用します。

コードサンプル

次の DAG は、 で指定されたテーブルのメタデータデータベースをクリーンアップしますTABLES_TO_CLEAN。この例では、指定したテーブルから過去 7 日間のデータを削除します。エントリの削除範囲を調整するには、 MAX_AGE_IN_DAYSを別の値に設定します。

Apache Airflow v2

```
from airflow import settings
from airflow.utils.dates import days_ago
from airflow.models import DagTag, DagModel, DagRun, ImportError, Log, SlaMiss,
 RenderedTaskInstanceFields, TaskInstance, TaskReschedule, XCom
from airflow.decorators import dag, task
from airflow.utils.dates import days_ago
from time import sleep

from airflow.version import version
major_version, minor_version = int(version.split('.')[0]), int(version.split('.')
[1])
if major_version >= 2 and minor_version >= 6:
    from airflow.jobs.job import Job
else:
    # The BaseJob class was renamed as of Apache Airflow v2.6
    from airflow.jobs.base_job import BaseJob as Job
```

```
# Delete entries for the past seven days. Adjust MAX_AGE_IN_DAYS to set how far back
 this DAG cleans the database.
MAX_AGE_IN_DAYS = 7
MIN_AGE_IN_DAYS = 0
DECREMENT = -7

# This is a list of (table, time) tuples.
# table = the table to clean in the metadata database
# time  = the column in the table associated to the timestamp of an entry
#         or None if not applicable.
TABLES_TO_CLEAN = [[Job, Job.latest_heartbeat],
    [TaskInstance, TaskInstance.execution_date],
    [TaskReschedule, TaskReschedule.execution_date],
    [DagTag, None],
    [DagModel, DagModel.last_parsed_time],
    [DagRun, DagRun.execution_date],
    [ImportError, ImportError.timestamp],
    [Log, Log.dttm],
    [SlaMiss, SlaMiss.execution_date],
    [RenderedTaskInstanceFields, RenderedTaskInstanceFields.execution_date],
    [XCom, XCom.execution_date],
]

@task()
def cleanup_db_fn(x):
    session = settings.Session()

    if x[1]:
        for oldest_days_ago in range(MAX_AGE_IN_DAYS, MIN_AGE_IN_DAYS, DECREMENT):
            earliest_days_ago = max(oldest_days_ago + DECREMENT, MIN_AGE_IN_DAYS)
            print(f"deleting {str(x[0])} entries between {earliest_days_ago} and
 {oldest_days_ago} days old...")
            earliest_date = days_ago(earliest_days_ago)
            oldest_date = days_ago(oldest_days_ago)
            query = session.query(x[0]).filter(x[1] >= oldest_date).filter(x[1] <=
 earliest_date)
            query.delete(synchronize_session= False)
            session.commit()
            sleep(5)
    else:
        # No time column specified for the table. Delete all entries
        print("deleting", str(x[0]), "...")
        query = session.query(x[0])
```

```
        query.delete(synchronize_session= False)
        session.commit()

    session.close()


@dag(
    dag_id="cleanup_db",
    schedule_interval="@weekly",
    start_date=days_ago(7),
    catchup=False,
    is_paused_upon_creation=False
)

def clean_db_dag_fn():
    t_last=None
    for x in TABLES_TO_CLEAN:
        t=cleanup_db_fn(x)
        if t_last:
            t_last >> t
        t_last = t

clean_db_dag = clean_db_dag_fn()
```

Amazon S3 の CSV ファイルへの環境メタデータのエクスポート

次のコード例は、データベースに対して一連の DAG 実行情報を照会し、Amazon S3 に保存されている.csvファイルにデータを書き込む有向非循環グラフ (DAG) を作成する方法を示しています。

お使いの環境の Aurora PostgreSQL データベースから情報をエクスポートして、データをローカルで検査したり、オブジェクトストレージにアーカイブしたり、Amazon S3 to Amazon Redshift オペレータやデータベースクリーンアップなどのツールと組み合わせたりして Amazon MWAA メタデータを環境外に移動し、future 分析のために保存しておきたい場合があります。

Apache Airflow のモデルにリストされているオブジェクトのいずれかに対してデータベースをクエリできます。このコードサンプルでは、3つのモデル DagRun、TaskFail、および TaskInstance を使用しており、DAG実行に関連する情報を提供します。

トピック

- バージョン

バージョン

- このページのコード例は、「Python 3.10」で Apache Airflow v2 以上と共に使用可能です。

前提条件

このページのサンプルコードを使用するには、以下が必要です。

- 「Amazon MWAA 環境」。
- メタデータ情報をエクスポートする新しい Amazon S3 バケット。

許可

Amazon MWAA は、クエリされたメタデータ情報を Amazon S3に書き込むためのアクション`s3:PutObject` の許可が必要です。次のポリシーステートメントを、環境の実行ロールに追加します。

```
{
  "Effect": "Allow",
  "Action": "s3:PutObject*",
  "Resource": "arn:aws:s3:::your-new-export-bucket"
}
```

このポリシーは、書き込みアクセスを`############`のみに制限します。

要件

- このコード例を Apache Airflow v2 で使用する場合、追加の依存関係は必要ありません。このコードでは、ご使用の環境にある Apache Airflow v2 のベースインストールを使用します。

コードサンプル

次のステップでは、Aurora PostgreSQL にクエリを実行し、その結果を新しい Amazon S3 バケットに書き込む DAG を作成する方法について説明します。

1. ターミナルで、DAG コードが保存されているディレクトリに移動します。例:

```
cd dags
```

2. 次のコード例の内容をコピーし、metadata_to_csv.py としてローカルに保存します。DAG がメタデータデータベースからクエリする最古のレコードの年齢を制御するために、MAX_AGE_IN_DAYS に割り当てられた値を変更することができます。

```
from airflow.decorators import dag, task
from airflow import settings
import os
import boto3
from airflow.utils.dates import days_ago
from airflow.models import DagRun, TaskFail, TaskInstance
import csv, re
from io import StringIO

DAG_ID = os.path.basename(__file__).replace(".py", "")

MAX_AGE_IN_DAYS = 30
S3_BUCKET = '<your-export-bucket>'
S3_KEY = 'files/export/{0}.csv'

# You can add other objects to export from the metadatabase,
OBJECTS_TO_EXPORT = [
    [DagRun,DagRun.execution_date],
    [TaskFail,TaskFail.execution_date],
    [TaskInstance, TaskInstance.execution_date],
]

@task()
def export_db_task(**kwargs):
    session = settings.Session()
    print("session: ",str(session))

    oldest_date = days_ago(MAX_AGE_IN_DAYS)
    print("oldest_date: ",oldest_date)
```

```
    s3 = boto3.client('s3')

    for x in OBJECTS_TO_EXPORT:
        query = session.query(x[0]).filter(x[1] >= days_ago(MAX_AGE_IN_DAYS))
        print("type",type(query))
        allrows=query.all()
        name=re.sub("[<>']", "", str(x[0]))
        print(name,": ",str(allrows))

        if len(allrows) > 0:
            outfileStr=""
            f = StringIO(outfileStr)
            w = csv.DictWriter(f, vars(allrows[0]).keys())
            w.writeheader()
            for y in allrows:
                w.writerow(vars(y))
            outkey = S3_KEY.format(name[6:])
            s3.put_object(Bucket=S3_BUCKET, Key=outkey, Body=f.getvalue())

@dag(
    dag_id=DAG_ID,
    schedule_interval=None,
    start_date=days_ago(1),
    )
def export_db():
    t = export_db_task()

metadb_to_s3_test = export_db()
```

3. 以下の AWS CLI コマンドを実行して、DAG を環境のバケットにコピーし、次に Apache Airflow UI を使用して DAG をトリガーします。

```
$ aws s3 cp your-dag.py s3://your-environment-bucket/dags/
```

4. 成功した場合は、export_dbタスクのタスクログに以下のような出力が表示されます。

```
[2022-01-01, 12:00:00 PDT] {{logging_mixin.py:109}} INFO - type <class
 'sqlalchemy.orm.query.Query'>
[2022-01-01, 12:00:00 PDT] {{logging_mixin.py:109}} INFO - class
 airflow.models.dagrun.DagRun : [your-tasks]
[2022-01-01, 12:00:00 PDT] {{logging_mixin.py:109}} INFO - type <class
 'sqlalchemy.orm.query.Query'>
```

```
[2022-01-01, 12:00:00 PDT] {{logging_mixin.py:109}} INFO - class
 airflow.models.taskfail.TaskFail :  [your-tasks]
[2022-01-01, 12:00:00 PDT] {{logging_mixin.py:109}} INFO - type <class
 'sqlalchemy.orm.query.Query'>
[2022-01-01, 12:00:00 PDT] {{logging_mixin.py:109}} INFO - class
 airflow.models.taskinstance.TaskInstance :  [your-tasks]
[2022-01-01, 12:00:00 PDT] {{python.py:152}} INFO - Done. Returned value was: OK
[2022-01-01, 12:00:00 PDT] {{taskinstance.py:1280}} INFO - Marking task as
 SUCCESS. dag_id=metadb_to_s3, task_id=export_db, execution_date=20220101T000000,
 start_date=20220101T000000, end_date=20220101T000000
[2022-01-01, 12:00:00 PDT] {{local_task_job.py:154}} INFO - Task exited with return
 code 0
[2022-01-01, 12:00:00 PDT] {{local_task_job.py:264}} INFO - 0 downstream tasks
 scheduled from follow-on schedule check
```

今、/files/export/ の新しい Amazon S3 バケット内の.csv ファイルにアクセスしてダウンロードできます。

Apache Airflow 変数の AWS Secrets Manager におけるシークレットキーの使用

以下のサンプルは、Apache Airflow 用の Amazon マネージドワークフローの Apache Airflow 変数のシークレットキーを取得するために AWS Secrets Manager を呼び出します。AWS Secrets Manager シークレットを使用した Apache Airflow 接続の設定 でのステップを完了していることが前提となります。

トピック

バージョン

- このページのサンプルコードは「Python 3.7」の Apache Airflow v1 で使用できます。

- このページのコード例は、「Python 3.10」で Apache Airflow v2 以上と共に使用可能です。

前提条件

このページのサンプルコードを使用するには、以下が必要です。

- Secrets Manager バックエンドを Apache Airflow 構成オプションとして使用する方法は、AWS Secrets Manager シークレットを使用した Apache Airflow 接続の設定 で示されています。
- AWS Secrets Manager シークレットを使用した Apache Airflow 接続の設定 に示すように、Secrets Manager の Apache Airflow 変数文字列。

許可

- Secrets Manager の権限 (AWS Secrets Manager シークレットを使用した Apache Airflow 接続の設定 を参照)。

要件

- このコード例を Apache Airflow v1 で使用する場合、追加の依存関係は必要ありません。このコードでは、ご使用の環境にある「Apache Airflow v1 のベースインストール」を使用します。

- このコード例を Apache Airflow v2 で使用する場合、追加の依存関係は必要ありません。このコードでは、ご使用の環境にある「Apache Airflow v2 のベースインストール」を使用します。

コードサンプル

以下の手順では、Secrets Manager を呼び出してシークレットを取得する DAG コードを作成する方法について説明します。

1. コマンドプロンプトで、DAG コードが保存されているディレクトリに移動します。例:

```
cd dags
```

2. 以下のコードサンプルの内容をコピーし、ローカルに `secrets-manager-var.py` として保存します。

```
from airflow import DAG
from airflow.operators.python_operator import PythonOperator
from airflow.models import Variable
from airflow.utils.dates import days_ago
from datetime import timedelta
import os
DAG_ID = os.path.basename(__file__).replace(".py", "")
DEFAULT_ARGS = {
    'owner': 'airflow',
    'depends_on_past': False,
    'email': ['airflow@example.com'],
    'email_on_failure': False,
    'email_on_retry': False,
}
def get_variable_fn(**kwargs):
    my_variable_name = Variable.get("test-variable", default_var="undefined")
    print("my_variable_name: ", my_variable_name)
    return my_variable_name
with DAG(
    dag_id=DAG_ID,
    default_args=DEFAULT_ARGS,
    dagrun_timeout=timedelta(hours=2),
    start_date=days_ago(1),
    schedule_interval='@once',
    tags=['variable']
) as dag:
    get_variable = PythonOperator(
        task_id="get_variable",
        python_callable=get_variable_fn,
        provide_context=True
    )
```

次のステップ

- この例の DAG コードを Amazon S3 バケットの dags フォルダにアップロードする方法については、「DAG の追加と更新」を参照してください。

AWS Secrets Manager の Apache Airflow 接続でのシークレットキーの使用

以下のサンプルでは、Amazon Managed Workflows for Apache Airflow で Apache Airflow 接続のシークレットキーを取得するために AWS Secrets Manager を呼び出しています。AWS Secrets Manager シークレットを使用した Apache Airflow 接続の設定 でのステップを完了していることが前提となります。

トピック

バージョン

- このページのサンプルコードは「Python 3.7」の Apache Airflow v1 で使用できます。
- このページのコード例は、「Python 3.10」で Apache Airflow v2 以上と共に使用可能です。

前提条件

このページのサンプルコードを使用するには、以下が必要です。

- Secrets Manager バックエンドを Apache Airflow 構成オプションとして使用する方法は、AWS Secrets Manager シークレットを使用した Apache Airflow 接続の設定 で示されています。
- AWS Secrets Manager シークレットを使用した Apache Airflow 接続の設定 に示すように、Secrets Manager のApache Airflow 接続文字列。

許可

- Secrets Manager の権限 (AWS Secrets Manager シークレットを使用した Apache Airflow 接続の設定 を参照)。

要件

- このコード例を Apache Airflow v1 で使用する場合、追加の依存関係は必要ありません。このコードでは、ご使用の環境にある「Apache Airflow v1 のベースインストール」を使用します。

- このコード例を Apache Airflow v2 で使用する場合、追加の依存関係は必要ありません。このコードでは、ご使用の環境にある「Apache Airflow v2 のベースインストール」を使用します。

コードサンプル

以下の手順では、Secrets Manager を呼び出してシークレットを取得する DAG コードを作成する方法について説明します。

Apache Airflow v2

1. コマンドプロンプトで、DAG コードが保存されているディレクトリに移動します。例:

```
cd dags
```

2. 以下のコードサンプルの内容をコピーし、ローカルに `secrets-manager.py` として保存します。

```
"""
Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the "Software"), to deal in
the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of
the Software, and to permit persons to whom the Software is furnished to do so.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS
FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
```

```
COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER
IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
"""
from airflow import DAG, settings, secrets
from airflow.operators.python import PythonOperator
from airflow.utils.dates import days_ago
from airflow.providers.amazon.aws.hooks.base_aws import AwsBaseHook

from datetime import timedelta
import os

### The steps to create this secret key can be found at: https://
docs.aws.amazon.com/mwaa/latest/userguide/connections-secrets-manager.html
sm_secretId_name = 'airflow/connections/myconn'

default_args = {
    'owner': 'airflow',
    'start_date': days_ago(1),
    'depends_on_past': False
}


### Gets the secret myconn from Secrets Manager
def read_from_aws_sm_fn(**kwargs):
    ### set up Secrets Manager
    hook = AwsBaseHook(client_type='secretsmanager')
    client = hook.get_client_type('secretsmanager')
    response = client.get_secret_value(SecretId=sm_secretId_name)
    myConnSecretString = response["SecretString"]

    return myConnSecretString

### 'os.path.basename(__file__).replace(".py", "")' uses the file name secrets-
manager.py for a DAG ID of secrets-manager
with DAG(
        dag_id=os.path.basename(__file__).replace(".py", ""),
        default_args=default_args,
        dagrun_timeout=timedelta(hours=2),
        start_date=days_ago(1),
        schedule_interval=None
) as dag:
    write_all_to_aws_sm = PythonOperator(
        task_id="read_from_aws_sm",
```

```
        python_callable=read_from_aws_sm_fn,
        provide_context=True
    )
```

Apache Airflow v1

1. コマンドプロンプトで、DAG コードが保存されているディレクトリに移動します。例:

```
cd dags
```

2. 以下のコードサンプルの内容をコピーし、ローカルに secrets-manager.py として保存します。

```
from airflow import DAG, settings, secrets
from airflow.operators.python_operator import PythonOperator
from airflow.utils.dates import days_ago
from airflow.contrib.hooks.aws_hook import AwsHook

from datetime import timedelta
import os

### The steps to create this secret key can be found at: https://
docs.aws.amazon.com/mwaa/latest/userguide/connections-secrets-manager.html
sm_secretId_name = 'airflow/connections/myconn'

default_args = {
    'owner': 'airflow',
    'start_date': days_ago(1),
    'depends_on_past': False
}


### Gets the secret myconn from Secrets Manager
def read_from_aws_sm_fn(**kwargs):
    ### set up Secrets Manager
    hook = AwsHook()
    client = hook.get_client_type('secretsmanager')
    response = client.get_secret_value(SecretId=sm_secretId_name)
    myConnSecretString = response["SecretString"]

    return myConnSecretString
```

```
### 'os.path.basename(__file__).replace(".py", "")' uses the file name secrets-
manager.py for a DAG ID of secrets-manager
with DAG(
        dag_id=os.path.basename(__file__).replace(".py", ""),
        default_args=default_args,
        dagrun_timeout=timedelta(hours=2),
        start_date=days_ago(1),
        schedule_interval=None
) as dag:
    write_all_to_aws_sm = PythonOperator(
        task_id="read_from_aws_sm",
        python_callable=read_from_aws_sm_fn,
        provide_context=True
    )
```

次のステップ

- この例の DAG コードを Amazon S3 バケットの dags フォルダにアップロードする方法については、「DAG の追加と更新」を参照してください。

Oracle でのカスタムプラグインの作成

次のサンプルでは、Oracle for Amazon MWAA を使用してカスタムプラグインを作成する手順を詳しく説明しており、これを plugins.zip ファイル内の他のカスタムプラグインやバイナリと組み合わせることができます。

目次

- Airflow 設定オプション
- 次のステップ

バージョン

- このページのサンプルコードは「Python 3.7」の Apache Airflow v1 で使用できます。
- このページのコード例は、「Python 3.10」で Apache Airflow v2 以上と共に使用可能です。

前提条件

このページのサンプルコードを使用するには、以下が必要です。

- 「Amazon MWAA 環境」。
- ワーカーのログ記録が、環境のログレベルが CRITICAL 以上で有効になっています。Amazon MWAA ログタイプとロググループの管理方法の詳細については、the section called “Airflow ログの表示” を参照してください。

許可

- このページのコード例を使用する場合、追加のアクセス許可は必要ありません。

要件

このページのサンプルコードを使用するには、次の依存関係を requirements.txt に追加してください。詳細については、「Python 依存関係のインストール」を参照してください。

Apache Airflow v2

```
-c https://raw.githubusercontent.com/apache/airflow/constraints-2.0.2/
constraints-3.7.txt
cx_Oracle
apache-airflow-providers-oracle
```

Apache Airflow v1

```
cx_Oracle==8.1.0
apache-airflow[oracle]==1.10.12
```

コードサンプル

次のステップでは、カスタムプラグインをテストする DAG コードを作成する方法について説明します。

1. コマンドプロンプトで、DAG コードが保存されているディレクトリに移動します。例:

```
cd dags
```

2. 以下のコードサンプルの内容をコピーし、ローカルに oracle.py として保存します。

```
from airflow import DAG
from airflow.operators.python_operator import PythonOperator
from airflow.utils.dates import days_ago
import os
import cx_Oracle

DAG_ID = os.path.basename(__file__).replace(".py", "")

def testHook(**kwargs):
    cx_Oracle.init_oracle_client()
    version = cx_Oracle.clientversion()
    print("cx_Oracle.clientversion",version)
    return version

with DAG(dag_id=DAG_ID, schedule_interval=None, catchup=False,
 start_date=days_ago(1)) as dag:
    hook_test = PythonOperator(
        task_id="hook_test",
        python_callable=testHook,
        provide_context=True
    )
```

カスタムプラグインを作成する

このセクションでは、依存関係のダウンロード、カスタムプラグインと plugins.zip の作成方法について説明します。

依存関係のダウンロード

Amazon MWAA は plugins.zip のコンテンツを各 Amazon MWAA スケジューラーとワーカーコンテナの `/usr/local/airflow/plugins` に抽出します。これはバイナリを環境に追加するために使用されます。次のステップでは、カスタムプラグインに必要なファイルを組み立てる方法について説明します。

Amazon Linux コンテナイメージをプルする

1. コマンドプロンプトで Amazon Linux コンテナイメージを取得し、コンテナをローカルで実行します。例:

   ```
   docker pull amazonlinux
   docker run -it amazonlinux:latest /bin/bash
   ```

   コマンドプロンプトで bash コマンドラインを呼び出す必要があります。例:

   ```
   bash-4.2#
   ```

2. Linux ネイティブの非同期 I/O 機能 (libaio) をインストールします。

   ```
   yum -y install libaio
   ```

3. 以降の手順では、このウィンドウを開いたままにしておいてください。次のファイルをローカルにコピーします: `lib64/libaio.so.1`、`lib64/libaio.so.1.0.0`、`lib64/libaio.so.1.0.1`。

クライアントフォルダをダウンロード

1. unzip パッケージをローカルにインストールします。例:

   ```
   sudo yum install unzip
   ```

2. `oracle_plugin` ディレクトリの作成 例:

```
mkdir oracle_plugin
cd oracle_plugin
```

3. 次の curl コマンドを使用して、「Linux x86-64 (64 ビット) 用 Oracle インスタントクライアントダウンロード」から「instantclient-basic-linux.x64-18.5.0.0.0dbru.zip」をダウンロードします。

```
curl https://download.oracle.com/otn_software/linux/instantclient/185000/
instantclient-basic-linux.x64-18.5.0.0.0dbru.zip > client.zip
```

4. client.zip ファイルを解凍します。例:

```
unzip *.zip
```

Docker からファイルを抽出します。

1. 新しいコマンドプロンプトで、Docker コンテナ ID を表示して書き留めます。例:

```
docker container ls
```

コマンドプロンプトでは、すべてのコンテナとその ID が返されるはずです。例:

```
debc16fd6970
```

2. oracle_plugin ディレクトリで、lib64/libaio.so.1、lib64/libaio.so.1.0.0、lib64/libaio.so.1.0.1 ファイルをローカル instantclient_18_5 フォルダに抽出してください。例:

```
docker cp debc16fd6970:/lib64/libaio.so.1 instantclient_18_5/
docker cp debc16fd6970:/lib64/libaio.so.1.0.0 instantclient_18_5/
docker cp debc16fd6970:/lib64/libaio.so.1.0.1 instantclient_18_5/
```

カスタムプラグイン

Apache Airflow は、起動時にプラグインフォルダにある Python ファイルの内容を実行します。これは環境変数の設定と変更に使用されます。次のステップでは、カスタムプラグインのサンプルコードを説明します。

- 以下のコードサンプルの内容をコピーし、ローカルに env_var_plugin_oracle.py として保存します。

```
from airflow.plugins_manager import AirflowPlugin
import os

os.environ["LD_LIBRARY_PATH"]='/usr/local/airflow/plugins/instantclient_18_5'
os.environ["DPI_DEBUG_LEVEL"]="64"

class EnvVarPlugin(AirflowPlugin):
    name = 'env_var_plugin'
```

Plugins.zip

以下のステップは、plugins.zip を作成する方法を示しています。この例の内容は、他のプラグインやバイナリと組み合わせて 1 つの plugins.zip ファイルにすることができます。

プラグインディレクトリの中身を Zip 圧縮する

1. コマンドプロンプトで、oracle_plugin ディレクトリに移動します。例:

```
cd oracle_plugin
```

2. instantclient_18_5 ディレクトリを plugins.zip に圧縮します。例:

```
zip -r ../plugins.zip ./
```

3. コマンドプロンプトに次のように表示されるはずです。

```
oracle_plugin$ ls
client.zip  instantclient_18_5
```

4. client.zip ファイルを削除します。例:

```
rm client.zip
```

env_var_plugin_oracle.py ファイルを圧縮します。

1. plugins.zip のルートに env_var_plugin_oracle.py ファイルを追加します。例:

```
zip plugins.zip env_var_plugin_oracle.py
```

2. これで、plugins.zip に次のものが含まれている必要があります。

```
env_var_plugin_oracle.py
instantclient_18_5/
```

Airflow 設定オプション

Apache Airflow v2 を使用している場合、`core.lazy_load_plugins : False` を Apache Airflow の構成オプションとして追加してください。詳細については、「2 の設定オプションによるプラグインの読み込み」を参照してください。

次のステップ

- この例の `requirements.txt` ファイルを Amazon S3 バケットにアップロードする方法について詳しくは、「Python 依存関係のインストール」をご覧ください。
- この例の DAG コードを Amazon S3 バケットの `dags` フォルダにアップロードする方法については、「DAG の追加と更新」を参照してください。
- この例の `plugins.zip` ファイルを Amazon S3 バケットにアップロードする方法について詳しくは、「カスタムプラグインのインストール」をご覧ください。

ランタイム環境変数を生成するカスタムプラグインを作成します

次の例では、Amazon Managed Workflows for Apache Airflow 環境でランタイムに環境変数を生成するカスタムプラグインを作成する手順を説明します。

トピック

- バージョン
- 前提条件
- 許可
- 要件
- カスタムプラグイン
- Plugins.zip

- Airflow 設定オプション
- 次のステップ

バージョン

- このページのサンプルコードは「Python 3.7」の Apache Airflow v1 で使用できます。

前提条件

このページのサンプルコードを使用するには、以下が必要です。

- 「Amazon MWAA 環境」。

許可

- このページのコード例を使用する場合、追加のアクセス許可は必要ありません。

要件

- このコード例を Apache Airflow v1 で使用する場合、追加の依存関係は必要ありません。このコードでは、お使いの環境にある「Apache Airflow v1 のベースインストール」を使用します。

カスタムプラグイン

Apache Airflow は、起動時にプラグインフォルダにある Python ファイルの内容を実行します。これは環境変数の設定と変更に使用されます。次のステップでは、カスタムプラグインのサンプルコードを説明します。

1. コマンドプロンプトで、プラグインが保存されているディレクトリに移動します。例:

```
cd plugins
```

2. 以下のコードサンプルの内容をコピーし、ローカルに env_var_plugin.py として保存する。としてローカルに保存します。

```
from airflow.plugins_manager import AirflowPlugin
```

```
import os

os.environ["PATH"] = os.getenv("PATH") + ":/usr/local/airflow/.local/lib/python3.7/
site-packages"
os.environ["JAVA_HOME"]="/usr/lib/jvm/java-1.8.0-
openjdk-1.8.0.272.b10-1.amzn2.0.1.x86_64"

class EnvVarPlugin(AirflowPlugin):
     name = 'env_var_plugin'
```

Plugins.zip

以下のステップは、plugins.zip を作成する方法を示しています。この例の内容は、他のプラグインやバイナリと組み合わせて 1 つの plugins.zip のファイルにすることができます。

1. コマンドプロンプトで、前のステップの hive_plugin ディレクトリに移動します。例:

```
cd plugins
```

2. plugins フォルダ内のコンテンツを圧縮します。

```
zip -r ../plugins.zip ./
```

Airflow 設定オプション

Apache Airflow v2 を使用している場合、core.lazy_load_plugins : False を Apache Airflow の構成オプションとして追加してください。詳細については、「2 の設定オプションによるプラグインの読み込み」を参照してください。

次のステップ

- この例の requirements.txt ファイルを Amazon S3 バケットにアップロードする方法について詳しくは、「Python 依存関係のインストール」をご覧ください。
- この例の DAG コードを Amazon S3 バケットの dags フォルダにアップロードする方法については、「DAG の追加と更新」を参照してください。
- この例の plugins.zip ファイルを Amazon S3 バケットにアップロードする方法について詳しくは、「カスタムプラグインのインストール」をご覧ください。

Amazon MWAA での DAG のタイムゾーンの変更

Apache Airflow は、有向非巡回グラフ (DAG) をデフォルトで UTC+0 でスケジュールします。次のステップは、Amazon MWAA が「Pendulum」を使用して DAG を実行するタイムゾーンを変更する方法を示しています。このトピックでは、任意で、環境の Apache Airflow ログのタイムゾーンを変更するカスタムプラグインを作成する方法を示します。

トピック

バージョン

- このページのコード例は、「Python 3.10」で Apache Airflow v2 以上と共に使用可能です。

前提条件

このページのサンプルコードを使用するには、以下が必要です。

- 「Amazon MWAA 環境」。

許可

- このページのコード例を使用する場合、追加のアクセス許可は必要ありません。

Airflow ログのタイムゾーンを変更するプラグインを作成する

Apache Airflow は、起動時に `plugins` ディレクトリ内の Python ファイルを実行します。次のプラグインを使用すると、エグゼキューターのタイムゾーンをオーバーライドできます。これにより、Apache Airflow がログを書き込むタイムゾーンが変更されます。

1. カスタムプラグイン用に plugins という名前のディレクトリを作成し、そのディレクトリに移動します。例:

```
$ mkdir plugins
$ cd plugins
```

2. 以下のコードサンプルの内容をコピーし、ローカルに dag-timezone-plugin.py として plugins フォルダに保存します。

```
import time
import os

os.environ['TZ'] = 'America/Los_Angeles'
time.tzset()
```

3. plugins ディレクトリに、__init__.py という名前の空の Python ファイルを作成します。plugins ディレクトリは以下のようになっているはずです。

```
plugins/
|-- __init__.py
|-- dag-timezone-plugin.py
```

plugins.zip を作成する

以下のステップは、plugins.zip を作成する方法を示しています。この例の内容は、他のプラグインやバイナリと組み合わせて 1 つの plugins.zip ファイルにすることができます。

1. コマンドプロンプトで、前の手順で作成した plugins ディレクトリに移動してください。例:

```
cd plugins
```

2. plugins ディレクトリ内のコンテンツを圧縮します。

```
zip -r ../plugins.zip ./
```

3. plugins.zip を S3 バケットにアップロードします

```
$ aws s3 cp plugins.zip s3://your-mwaa-bucket/
```

コードサンプル

DAG が実行されるデフォルトのタイムゾーン (UTC+0) を変更するには、「Pendulum」というライブラリを使用します。これは、タイムゾーン対応の日時を処理するための Python ライブラリです。

1. コマンドプロンプトで、DAG が保存されているディレクトリに移動します。例:

```
$ cd dags
```

2. 以下の例の内容をコピーして、tz-aware-dag.py として保存してください。

```
from airflow import DAG
from airflow.operators.bash_operator import BashOperator
from datetime import datetime, timedelta
# Import the Pendulum library.
import pendulum

# Instantiate Pendulum and set your timezone.
local_tz = pendulum.timezone("America/Los_Angeles")

with DAG(
    dag_id = "tz_test",
    schedule_interval="0 12 * * *",
    catchup=False,
    start_date=datetime(2022, 1, 1, tzinfo=local_tz)
) as dag:
    bash_operator_task = BashOperator(
        task_id="tz_aware_task",
        dag=dag,
        bash_command="date"
    )
```

3. 以下の AWS CLI コマンドを実行して、DAG を環境のバケットにコピーし、次に Apache Airflow UI を使用して DAG をトリガーします。

```
$ aws s3 cp your-dag.py s3://your-environment-bucket/dags/
```

4. 成功した場合、tz_test DAG で tz_aware_task のタスクログに以下のような出力が表示されます。

```
[2022-08-01, 12:00:00 PDT] {{subprocess.py:74}} INFO - Running command: ['bash', '-
c', 'date']
[2022-08-01, 12:00:00 PDT] {{subprocess.py:85}} INFO - Output:
[2022-08-01, 12:00:00 PDT] {{subprocess.py:89}} INFO - Mon Aug  1 12:00:00 PDT 2022
[2022-08-01, 12:00:00 PDT] {{subprocess.py:93}} INFO - Command exited with return
 code 0
[2022-08-01, 12:00:00 PDT] {{taskinstance.py:1280}} INFO - Marking task as
 SUCCESS. dag_id=tz_test, task_id=tz_aware_task, execution_date=20220801T190033,
 start_date=20220801T190035, end_date=20220801T190035
[2022-08-01, 12:00:00 PDT] {{local_task_job.py:154}} INFO - Task exited with return
 code 0
[2022-08-01, 12:00:00 PDT] {{local_task_job.py:264}} INFO - 0 downstream tasks
 scheduled from follow-on schedule check
```

次のステップ

- この例の plugins.zip ファイルを Amazon S3 バケットにアップロードする方法について詳しくは、「カスタムプラグインのインストール」をご覧ください。

CodeArtifact トークンのリフレッシュ

CodeArtifact を使用して Python の依存関係をインストールする場合、Amazon MWAA にはアクティブなトークンが必要です。Amazon MWAA が実行時に CodeArtifact リポジトリにアクセスできるようにするには、「起動スクリプト」を使用して「PIP_EXTRA_INDEX_URL」をトークンで設定できます。

次のトピックでは、「get_authorization_token」CodeArtifact API オペレーションを使用して環境が起動または更新されるたびに新しいトークンを取得する起動スクリプトを作成する方法について説明します。

トピック

バージョン

- このページのコード例は、「Python 3.10」で Apache Airflow v2 以上と共に使用可能です。

前提条件

このページのサンプルコードを使用するには、以下が必要です。

- 「Amazon MWAA 環境」。
- 環境の依存関係を保存する「CodeArtifact リポジトリ」です。

許可

CodeArtifact トークンを更新して結果を Amazon S3 に書き込むには、Amazon MWAA が実行ロールに以下のアクセス権限を持っている必要があります。

- codeartifact:GetAuthorizationToken このアクションにより、Amazon MWAA は CodeArtifact から新しいトークンを取得することができます。次のポリシーは、作成したすべての CodeArtifact ドメインにアクセス許可を付与します。ステートメントのリソース値を変更し、環境からアクセスさせたいドメインのみを指定することで、ドメインへのアクセスをさらに制限できます。

```
{
  "Effect": "Allow",
  "Action": "codeartifact:GetAuthorizationToken",
  "Resource": "arn:aws:codeartifact:us-west-2:*:domain/*"
}
```

- sts:GetServiceBearerToken アクションは、CodeArtifact「GetAuthorizationToken」API オペレーションを呼び出すために必要です。この操作は、pip などのパッケージ・マネージャーを CodeArtifact とともに使用するときに使用しなければならないトークンを返します。CodeArtifact リポジトリでパッケージマネージャーを使用するには、環境の実行ロールロールが、以下のポリシーステートメントに示すように sts:GetServiceBearerToken を許可する必要があります。

```
{
  "Sid": "AllowServiceBearerToken",
  "Effect": "Allow",
```

```
  "Action": "sts:GetServiceBearerToken",
  "Resource": "*"
}
```

コードサンプル

以下の手順では、CodeArtifact トークンを更新する起動スクリプトを作成する方法について説明します。

1. 以下のコードサンプルの内容をコピーし、ローカルに code_artifact_startup_script.sh として保存します。

```
#!/bin/sh

# Startup script for MWAA, see https://docs.aws.amazon.com/mwaa/latest/userguide/
using-startup-script.html

set -eu

# setup code artifact endpoint and token
# https://pip.pypa.io/en/stable/cli/pip_install/#cmdoption-0
# https://docs.aws.amazon.com/mwaa/latest/userguide/samples-code-artifact.html
DOMAIN="amazon"
DOMAIN_OWNER="112233445566"
REGION="us-west-2"
REPO_NAME="MyRepo"
echo "Getting token for CodeArtifact with args: --domain $DOMAIN --region $REGION
 --domain-owner $DOMAIN_OWNER"
TOKEN=$(aws codeartifact get-authorization-token --domain $DOMAIN --region $REGION
 --domain-owner $DOMAIN_OWNER | jq -r '.authorizationToken')
echo "Setting Pip env var for '--index-url' to point to CodeArtifact"
export PIP_EXTRA_INDEX_URL="https://aws:$TOKEN@$DOMAIN-
$DOMAIN_OWNER.d.codeartifact.$REGION.amazonaws.com/pypi/$REPO_NAME/simple/"
echo "CodeArtifact startup setup complete"
```

2. スクリプトを保存したフォルダに移動します。新しいプロンプトウィンドウで cp を使用して、スクリプトをバケットにアップロードします。*your-s3-bucket* を情報に置き換えてください。

```
$ aws s3 cp code_artifact_startup_script.sh s3://your-s3-bucket/
code_artifact_startup_script.sh
```

成功すると、Amazon S3 はオブジェクトへの URL パスを出力します。

```
upload: ./code_artifact_startup_script.sh to s3://your-s3-bucket/
code_artifact_startup_script.sh
```

スクリプトをアップロードすると、環境が更新され、起動時にスクリプトが実行されます。

次のステップ

- スタートアップスクリプトを使用して環境をカスタマイズする方法については、the section called "スタートアップスクリプトの使用" を参照してください。
- この例の DAG コードを Amazon S3 バケットの dags フォルダにアップロードする方法については、「DAG の追加と更新」を参照してください。
- この例の plugins.zip ファイルを Amazon S3 バケットにアップロードする方法について詳しくは、「カスタムプラグインのインストール」をご覧ください。

Apache Hive と Hadoop を使ったカスタムプラグインの作成

Amazon MWAA は plugins.zip から /usr/local/airflow/plugins にコンテンツを抽出します。これを使用して、コンテナにバイナリを追加できます。さらに、Apache Airflow は起動時に plugins フォルダ内の Python ファイルの内容を実行するため、環境変数を設定および変更できます。以下のサンプルでは、Apache Airflow 用 Amazon マネージドワークフロー環境で Apache Hive と Hadoop を使用してカスタムプラグインを作成する手順を説明し、他のカスタムプラグインやバイナリと組み合わせることができます。

トピック

- バージョン
- 前提条件
- 許可
- 要件
- 依存関係のダウンロード

- カスタムプラグイン
- Plugins.zip
- コードサンプル
- Airflow 設定オプション
- 次のステップ

バージョン

- このページのサンプルコードは「Python 3.7」の Apache Airflow v1 で使用できます。
- このページのコード例は、「Python 3.10」で Apache Airflow v2 以上と共に使用可能です。

前提条件

このページのサンプルコードを使用するには、以下が必要です。

- 「Amazon MWAA 環境」。

許可

- このページのコード例を使用する場合、追加のアクセス許可は必要ありません。

要件

このページのサンプルコードを使用するには、次の依存関係を `requirements.txt` に追加してください。詳細については、「Python 依存関係のインストール」を参照してください。

Apache Airflow v2

```
-c https://raw.githubusercontent.com/apache/airflow/constraints-2.0.2/
constraints-3.7.txt
apache-airflow-providers-amazon[apache.hive]
```

Apache Airflow v1

```
apache-airflow[hive]==1.10.12
```

依存関係のダウンロード

Amazon MWAA は plugins.zip のコンテンツを各 Amazon MWAA スケジューラーとワーカーコンテナの /usr/local/airflow/plugins に抽出します。これはバイナリを環境に追加するために使用されます。次のステップでは、カスタムプラグインに必要なファイルを組み立てる方法について説明します。

1. コマンドプロンプトで、プラグインを作成したいディレクトリに移動します。例:

```
cd plugins
```

2. 「ミラー」から「Hadoop」をダウンロードします。例えば:

```
wget https://downloads.apache.org/hadoop/common/hadoop-3.3.0/hadoop-3.3.0.tar.gz
```

3. 「ミラー」から「Hive」をダウンロードします。例えば:

```
wget https://downloads.apache.org/hive/hive-3.1.2/apache-hive-3.1.2-bin.tar.gz
```

4. ディレクトリを作成します。例:

```
mkdir hive_plugin
```

5. Hadoop を抽出します。

```
tar -xvzf hadoop-3.3.0.tar.gz -C hive_plugin
```

6. Hive を抽出します。

```
tar -xvzf apache-hive-3.1.2-bin.tar.gz -C hive_plugin
```

カスタムプラグイン

Apache Airflow は、起動時にプラグインフォルダにある Python ファイルの内容を実行します。これは環境変数の設定と変更に使用されます。次のステップでは、カスタムプラグインのサンプルコードを説明します。

1. コマンドラインプロンプトで、hive_plugin ディレクトリに移動します。例:

```
cd hive_plugin
```

2. 次のコードサンプルの内容をコピーし、同じ hive_plugin ディレクトリで、hive_plugin.py と名前を付けてローカルに保存します。

```
from airflow.plugins_manager import AirflowPlugin
import os
os.environ["JAVA_HOME"]="/usr/lib/jvm/jre"
os.environ["HADOOP_HOME"]='/usr/local/airflow/plugins/hadoop-3.3.0'
os.environ["HADOOP_CONF_DIR"]='/usr/local/airflow/plugins/hadoop-3.3.0/etc/hadoop'
os.environ["HIVE_HOME"]='/usr/local/airflow/plugins/apache-hive-3.1.2-bin'
os.environ["PATH"] = os.getenv("PATH") + ":/usr/local/airflow/plugins/
hadoop-3.3.0:/usr/local/airflow/plugins/apache-hive-3.1.2-bin/bin:/usr/local/
airflow/plugins/apache-hive-3.1.2-bin/lib"
os.environ["CLASSPATH"] = os.getenv("CLASSPATH") + ":/usr/local/airflow/plugins/
apache-hive-3.1.2-bin/lib"
class EnvVarPlugin(AirflowPlugin):
    name = 'hive_plugin'
```

3. 次のテキストの内容をコピーし、hive_plugin ディレクトリに .airflowignore としてローカルに保存します。

```
hadoop-3.3.0
apache-hive-3.1.2-bin
```

Plugins.zip

以下のステップは、plugins.zip を作成する方法を示しています。この例の内容は、他のプラグインやバイナリと組み合わせて 1 つの plugins.zip のファイルにすることができます。

1. コマンドプロンプトで、前のステップの hive_plugin ディレクトリに移動します。例:

```
cd hive_plugin
```

2. plugins フォルダ内のコンテンツを圧縮します。

```
zip -r ../hive_plugin.zip ./
```

コードサンプル

次のステップでは、カスタムプラグインをテストする DAG コードを作成する方法について説明します。

1. コマンドプロンプトで、DAG コードが保存されているディレクトリに移動します。例:

```
cd dags
```

2. 以下のコードサンプルの内容をコピーし、ローカルに hive.py として保存します。

```
from airflow import DAG
from airflow.operators.bash_operator import BashOperator
from airflow.utils.dates import days_ago

with DAG(dag_id="hive_test_dag", schedule_interval=None, catchup=False,
 start_date=days_ago(1)) as dag:
    hive_test = BashOperator(
        task_id="hive_test",
        bash_command='hive --help'
    )
```

Airflow 設定オプション

Apache Airflow v2 を使用している場合、core.lazy_load_plugins : False を Apache Airflow の構成オプションとして追加してください。詳細については、「2 の設定オプションによるプラグインの読み込み」を参照してください。

次のステップ

- この例の requirements.txt ファイルを Amazon S3 バケットにアップロードする方法について詳しくは、「Python 依存関係のインストール」をご覧ください。

- この例の DAG コードを Amazon S3 バケットの dags フォルダにアップロードする方法については、「DAG の追加と更新」を参照してください。
- この例の plugins.zip ファイルを Amazon S3 バケットにアップロードする方法について詳しくは、「カスタムプラグインのインストール」をご覧ください。

Apache Airflow Python VirtualEnv オペレータ用のカスタムプラグインを作成する

次のサンプルは、Apache Airflow 用の Amazon マネージドワークフロー上のカスタムプラグインを使用して Apache Airflow PythonVirtualEnvOperator にパッチを適用する方法を示しています。

トピック

バージョン

- このページのサンプルコードは「Python 3.7」の Apache Airflow v1 で使用できます。
- このページのコード例は、「Python 3.10」で Apache Airflow v2 以上と共に使用可能です。

前提条件

このページのサンプルコードを使用するには、以下が必要です。

- 「Amazon MWAA 環境」。

許可

- このページのコード例を使用する場合、追加のアクセス許可は必要ありません。

要件

このページのサンプルコードを使用するには、次の依存関係を `requirements.txt` に追加してください。詳細については、「Python 依存関係のインストール」を参照してください。

```
virtualenv
```

カスタムプラグインのサンプルコード

Apache Airflow は、起動時にプラグインフォルダにある Python ファイルの内容を実行します。このプラグインは、そのスタートアッププロセス中の組み込み `PythonVirtualenvOperator` をパッチし、Amazon MWAA と互換性があるようにします。次のステップは、カスタムプラグインのサンプルコードが 示しています。

Apache Airflow v2

1. コマンドラインプロンプトで、`plugins` ディレクトリに移動します。例:

   ```
   cd plugins
   ```

2. 以下のコードサンプルの内容をコピーし、ローカルに `virtual_python_plugin.py` として保存します。

   ```
   """
   Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.

   Permission is hereby granted, free of charge, to any person obtaining a copy of
   this software and associated documentation files (the "Software"), to deal in
   the Software without restriction, including without limitation the rights to
   use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of
   the Software, and to permit persons to whom the Software is furnished to do so.

   THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
   IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS
   FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
   COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER
   ```

```
IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
"""
from airflow.plugins_manager import AirflowPlugin
import airflow.utils.python_virtualenv
from typing import List

def _generate_virtualenv_cmd(tmp_dir: str, python_bin: str,
 system_site_packages: bool) -> List[str]:
    cmd = ['python3','/usr/local/airflow/.local/lib/python3.7/site-packages/
virtualenv', tmp_dir]
    if system_site_packages:
        cmd.append('--system-site-packages')
    if python_bin is not None:
        cmd.append(f'--python={python_bin}')
    return cmd

airflow.utils.python_virtualenv._generate_virtualenv_cmd=_generate_virtualenv_cmd

class VirtualPythonPlugin(AirflowPlugin):
    name = 'virtual_python_plugin'
```

Apache Airflow v1

1. コマンドラインプロンプトで、plugins ディレクトリに移動します。例:

```
cd plugins
```

2. 以下のコードサンプルの内容をコピーし、ローカルに virtual_python_plugin.py として保存します。

```
from airflow.plugins_manager import AirflowPlugin
from airflow.operators.python_operator import PythonVirtualenvOperator

def _generate_virtualenv_cmd(self, tmp_dir):
    cmd = ['python3','/usr/local/airflow/.local/lib/python3.7/site-packages/
virtualenv', tmp_dir]
    if self.system_site_packages:
        cmd.append('--system-site-packages')
    if self.python_version is not None:
        cmd.append('--python=python{}'.format(self.python_version))
    return cmd
```

```
PythonVirtualenvOperator._generate_virtualenv_cmd=_generate_virtualenv_cmd

class EnvVarPlugin(AirflowPlugin):
    name = 'virtual_python_plugin'
```

Plugins.zip

以下のステップは、plugins.zip を作成する方法を示しています。

1. コマンドプロンプトで、上記の virtual_python_plugin.py が含まれるディレクトリに移動してください。例:

```
cd plugins
```

2. plugins フォルダ内のコンテンツを圧縮します。

```
zip plugins.zip virtual_python_plugin.py
```

コードサンプル

次の手順では、カスタムプラグインの DAG コードが 作成する方法について説明します。

Apache Airflow v2

1. コマンドプロンプトで、DAG コードが保存されているディレクトリに移動します。例:

```
cd dags
```

2. 以下のコードサンプルの内容をコピーし、ローカルに virtualenv_test.py として保存します。

```
"""
Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the "Software"), to deal in
the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of
the Software, and to permit persons to whom the Software is furnished to do so.
```

```
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS
FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER
IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
"""
from airflow import DAG
from airflow.operators.python import PythonVirtualenvOperator
from airflow.utils.dates import days_ago
import os

os.environ["PATH"] = os.getenv("PATH") + ":/usr/local/airflow/.local/bin"

def virtualenv_fn():
    import boto3
    print("boto3 version ",boto3.__version__)

with DAG(dag_id="virtualenv_test", schedule_interval=None, catchup=False,
 start_date=days_ago(1)) as dag:
    virtualenv_task = PythonVirtualenvOperator(
        task_id="virtualenv_task",
        python_callable=virtualenv_fn,
        requirements=["boto3>=1.17.43"],
        system_site_packages=False,
        dag=dag,
    )
```

Apache Airflow v1

1. コマンドプロンプトで、DAG コードが保存されているディレクトリに移動します。例:

```
cd dags
```

2. 以下のコードサンプルの内容をコピーし、ローカルに virtualenv_test.py として保存します。

```
"""
Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
```

```
Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the "Software"), to deal in
the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of
the Software, and to permit persons to whom the Software is furnished to do so.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS
FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER
IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
"""
from airflow import DAG
from airflow.operators.python_operator import PythonVirtualenvOperator
from airflow.utils.dates import days_ago
import os

os.environ["PATH"] = os.getenv("PATH") + ":/usr/local/airflow/.local/bin"

def virtualenv_fn():
    import boto3
    print("boto3 version ",boto3.__version__)

with DAG(dag_id="virtualenv_test", schedule_interval=None, catchup=False,
 start_date=days_ago(1)) as dag:
    virtualenv_task = PythonVirtualenvOperator(
        task_id="virtualenv_task",
        python_callable=virtualenv_fn,
        requirements=["boto3>=1.17.43"],
        system_site_packages=False,
        dag=dag,
    )
```

Airflow 設定オプション

Apache Airflow v2 を使用している場合、`core.lazy_load_plugins : False` を Apache Airflow の構成オプションとして追加してください。詳細については、「2 の設定オプションによるプラグインの読み込み」を参照してください。

次のステップ

- この例の `requirements.txt` ファイルを Amazon S3 バケットにアップロードする方法について詳しくは、「Python 依存関係のインストール」をご覧ください。
- この例の DAG コードを Amazon S3 バケットの `dags` フォルダにアップロードする方法については、「DAG の追加と更新」を参照してください。
- この例の `plugins.zip` ファイルを Amazon S3 バケットにアップロードする方法について詳しくは、「カスタムプラグインのインストール」をご覧ください。

Lambda 関数を使用して DAG を呼び出す

次のコード例では、「AWS Lambda」関数を使用して Apache Airflow CLI トークンを取得し、Amazon MWAA 環境で有向非巡回グラフ (DAG) を呼び出します。

トピック

Version

- このページのコード例は、「Python 3.10」で Apache Airflow v2 以上と共に使用可能です。

前提条件

コードサンプルを使用するには、以下が必要です。

- 「Amazon MWAA 環境」には「パブリックネットワークアクセスモード」を使用してください。
- 最新の Python ランタイムを使用する「Lambda 関数」を用意してください。

> **Note**
>
> Lambda 関数と Amazon MWAA 環境が同じ VPC にある場合は、このコードをプライベートネットワークで使用できます。この構成では、Lambda 関数の実行ロールに、Amazon Elastic Compute Cloud (Amazon EC2) CreateNetworkInterface API オペレーションを呼び出すアクセス許可が必要です。このアクセス許可は、AWSLambdaVPCAccessExecutionRole AWS 管理ポリシーを使用して付与できます。

アクセス許可

このページのコード例を使用するには、Amazon MWAA 環境の実行ロールが airflow:CreateCliToken アクションを実行するためのアクセス権が必要です。このアクセス許可は、 AmazonMWAAAirflowCliAccess AWS 管理ポリシーを使用して指定できます。

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

詳細については、「Apache Airflow CLI ポリシー: AmazonMWAAAirflowCli アクセス」を参照してください。

依存関係

- このコード例を Apache Airflow v2 で使用する場合、追加の依存関係は必要ありません。このコードでは、ご使用の環境にある「Apache Airflow v2 のベースインストール」を使用します。

コード例

1. https://console.aws.amazon.com/lambda/ で AWS Lambda コンソールを開きます。

2. 関数リストから Lambda 関数を選択します。
3. 関数ページで次のコードをコピーし、以下をリソース名に置き換えます。

   - YOUR_ENVIRONMENT_NAME – Amazon MWAA 環境の名前。
   - YOUR_DAG_NAME — 呼び出したい DAG の名前。

```
import boto3
import http.client
import base64
import ast
mwaa_env_name = 'YOUR_ENVIRONMENT_NAME'
dag_name = 'YOUR_DAG_NAME'
mwaa_cli_command = 'dags trigger'

client = boto3.client('mwaa')

def lambda_handler(event, context):
    # get web token
    mwaa_cli_token = client.create_cli_token(
        Name=mwaa_env_name
    )

    conn = http.client.HTTPSConnection(mwaa_cli_token['WebServerHostname'])
    payload = mwaa_cli_command + " " + dag_name
    headers = {
      'Authorization': 'Bearer ' + mwaa_cli_token['CliToken'],
      'Content-Type': 'text/plain'
    }
    conn.request("POST", "/aws_mwaa/cli", payload, headers)
    res = conn.getresponse()
    data = res.read()
    dict_str = data.decode("UTF-8")
    mydata = ast.literal_eval(dict_str)
    return base64.b64decode(mydata['stdout'])
```

4. [デプロイ] を選択します。
5. [テスト] を選択し、Lambda コンソールを使用して関数を呼び出します。
6. Lambda が DAG を正常に呼び出したことを確認するには、Amazon MWAA コンソールを使用して環境の Apache Airflow UI に移動し、次の操作を行います。

   a. [DAG] ページの DAG のリストから新しいターゲット DAG を見つけます。

b. [最終実行] で、最新の DAG 実行のタイムスタンプを確認します。このタイムスタンプは、他の環境における `invoke_dag` の最新のタイムスタンプとほぼ一致する必要があります。

c. [最近のタスク] で、前回の実行が成功したことを確認します。

さまざまな Amazon MWAA 環境での DAG の呼び出し

次のコード例では、Apache Airflow CLI トークンを作成します。次に、このコードは、ある Amazon MWAA 環境の有向非巡回グラフ (DAG) を使用して、別の Amazon MWAA 環境の DAG を呼び出します。

トピック

バージョン

- このページのコード例は、「Python 3.10」で Apache Airflow v2 以上と共に使用可能です。

前提条件

このページのコード例を使用するには、以下のものが必要である：

- [パブリックネットワーク] のウェブサーバーにアクセスできる 2 つの「Amazon MWAA 環境」(現在の環境を含む)。
- ターゲット環境の Amazon Simple Storage Service (Amazon S3) バケットに、サンプルの DAG。

許可

このページのコード例を使用するには、環境の実行ロールに Apache Airflow CLI トークンを作成する権限が必要です。この権限を付与するためには、AWS のマネージドポリシー `AmazonMWAAAirflowCliAccess` をアタッチすることができます。

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

詳細については、「Apache Airflow CLI ポリシー: AmazonMWAAAirflowCli アクセス」を参照してください。

依存関係

- このコード例を Apache Airflow v2 で使用する場合、追加の依存関係は必要ありません。このコードでは、ご使用の環境にある「Apache Airflow v2 のベースインストール」を使用します。

コード例

次のコード例は、現在の環境で DAG を使用して別の環境で DAG を呼び出していると想定しています。

1. ターミナルで、DAG コードが保存されているディレクトリに移動します。例:

```
cd dags
```

2. 次のコード例の内容をコピーし、invoke_dag.py という名前でローカルに保存します。以下の値をお客様の情報に置き換えます。

   - your-new-environment-name — DAG を起動する他の環境の名前。
   - your-target-dag-id — 起動する他の環境の DAG の ID。

```
from airflow.decorators import dag, task
import boto3
from datetime import datetime, timedelta
import os, requests
```

```
DAG_ID = os.path.basename(__file__).replace(".py", "")

@task()
def invoke_dag_task(**kwargs):
    client = boto3.client('mwaa')
    token = client.create_cli_token(Name='your-new-environment-name')
    url = f"https://{token['WebServerHostname']}/aws_mwaa/cli"
    body = 'dags trigger your-target-dag-id'
    headers = {
        'Authorization' : 'Bearer ' + token['CliToken'],
        'Content-Type': 'text/plain'
        }
    requests.post(url, data=body, headers=headers)

@dag(
    dag_id=DAG_ID,
    schedule_interval=None,
    start_date=datetime(2022, 1, 1),
    dagrun_timeout=timedelta(minutes=60),
    catchup=False
    )
def invoke_dag():
    t = invoke_dag_task()

invoke_dag_test = invoke_dag()
```

3. 以下の AWS CLI コマンドを実行して、DAG を環境のバケットにコピーし、次に Apache Airflow UI を使用して DAG をトリガーします。

```
$ aws s3 cp your-dag.py s3://your-environment-bucket/dags/
```

4. DAG が正常に実行されると、invoke_dag_task のタスクログに以下のような出力が表示されます。

```
[2022-01-01, 12:00:00 PDT] {{python.py:152}} INFO - Done. Returned value was: None
[2022-01-01, 12:00:00 PDT] {{taskinstance.py:1280}} INFO - Marking task as SUCCESS.
 dag_id=invoke_dag, task_id=invoke_dag_task, execution_date=20220101T120000,
 start_date=20220101T120000, end_date=20220101T120000
[2022-01-01, 12:00:00 PDT] {{local_task_job.py:154}} INFO - Task exited with return
 code 0
```

```
[2022-01-01, 12:00:00 PDT] {{local_task_job.py:264}} INFO - 0 downstream tasks
 scheduled from follow-on schedule check
```

DAG が正常に呼び出されたことを確認するには、新しい環境の Apache Airflow UI に移動し、次の操作を行います。

a. [DAG] ページの DAG のリストから新しいターゲット DAG を見つけます。

b. [最終実行] で、最新の DAG 実行のタイムスタンプを確認します。このタイムスタンプは、他の環境における `invoke_dag` の最新のタイムスタンプとほぼ一致する必要があります。

c. [最近のタスク] で、前回の実行が成功したことを確認します。

Amazon RDS for Microsoft SQL Server で Amazon MWAA を使用する

Apache Airflow 用 Amazon マネージドワークフローを使用して「RDS for SQL Server」に接続できます。次のサンプルコードでは、Apache Airflow 用 Amazon マネージドワークフロー環境の DAG を使用して、Amazon RDS for Microsoft SQL Server に接続してクエリを実行します。

トピック

バージョン

- このページのサンプルコードは「Python 3.7」の Apache Airflow v1 で使用できます。

- このページのコード例は、「Python 3.10」で Apache Airflow v2 以上と共に使用可能です。

前提条件

このページのサンプルコードを使用するには、以下が必要です。

- 「Amazon MWAA 環境」。
- Amazon MWAA と RDS for SQL Server は同じ Amazon MWAA/ で実行されています
- Amazon MWAA とサーバーの VPC セキュリティグループは以下の接続で構成されます。
  - Amazon MWAA のセキュリティグループにある Amazon MWAA 用にポート 1433 を開くためのインバウンドルール
  - または、Amazon MWAA から RDS へ 1433 のポートのオープンに関するアウトバウンドルール
- SQL サーバー用 RDS 用 Apache Airflow Connection には、前のプロセスで作成された Amazon MWAA SQL サーバーデータベースのホスト名、ポート、ユーザー名、パスワードが反映されます。

依存関係

このセクションのサンプルコードを使用するには、`requirements.txt` に次の依存関係を追加します。詳細については、「Python 依存関係のインストール」を参照してください。

Apache Airflow v2

```
apache-airflow-providers-microsoft-mssql==1.0.1
apache-airflow-providers-odbc==1.0.1
pymssql==2.2.1
```

Apache Airflow v1

```
apache-airflow[mssql]==1.10.12
```

Apache Airflow v2 接続

Apache Airflow v2 の接続を使用している場合は、Airflow 接続オブジェクトに次のキーと値のペアが含まれていることを確認してください。

1. 接続 ID: mssql_default
2. 接続タイプ: Amazon Web Services

3. ホスト: YOUR_DB_HOST
4. スキーマ :
5. ログイン:管理者
6. パスワード:
7. ポート: 1433
8. エキストラ:

コードサンプル

1. コマンドプロンプトで、DAG コードが保存されているディレクトリに移動します。例:

```
cd dags
```

2. 以下のコードサンプルの内容をコピーし、ローカルに sql-server.py として保存します。

```
"""
Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the "Software"), to deal in
the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of
the Software, and to permit persons to whom the Software is furnished to do so.
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS
FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER
IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
"""
import pymssql
import logging
import sys
from airflow import DAG
from datetime import datetime
from airflow.operators.mssql_operator import MsSqlOperator
from airflow.operators.python_operator import PythonOperator

default_args = {
    'owner': 'aws',
    'depends_on_past': False,
```

```
    'start_date': datetime(2019, 2, 20),
    'provide_context': True
}

dag = DAG(
    'mssql_conn_example', default_args=default_args, schedule_interval=None)

drop_db = MsSqlOperator(
   task_id="drop_db",
   sql="DROP DATABASE IF EXISTS testdb;",
   mssql_conn_id="mssql_default",
   autocommit=True,
   dag=dag
)

create_db = MsSqlOperator(
   task_id="create_db",
   sql="create database testdb;",
   mssql_conn_id="mssql_default",
   autocommit=True,
   dag=dag
)

create_table = MsSqlOperator(
   task_id="create_table",
   sql="CREATE TABLE testdb.dbo.pet (name VARCHAR(20), owner VARCHAR(20));",
   mssql_conn_id="mssql_default",
   autocommit=True,
   dag=dag
)

insert_into_table = MsSqlOperator(
   task_id="insert_into_table",
   sql="INSERT INTO testdb.dbo.pet VALUES ('Olaf', 'Disney');",
   mssql_conn_id="mssql_default",
   autocommit=True,
   dag=dag
)

def select_pet(**kwargs):
   try:
        conn = pymssql.connect(
            server='sampledb.<xxxxxx>.<region>.rds.amazonaws.com',
            user='admin',
```

```
            password='<yoursupersecretpassword>',
            database='testdb'
        )

        # Create a cursor from the connection
        cursor = conn.cursor()
        cursor.execute("SELECT * from testdb.dbo.pet")
        row = cursor.fetchone()

        if row:
            print(row)
   except:
      logging.error("Error when creating pymssql database connection: %s",
 sys.exc_info()[0])

select_query = PythonOperator(
    task_id='select_query',
    python_callable=select_pet,
    dag=dag,
)

drop_db >> create_db >> create_table >> insert_into_table >> select_query
```

次のステップ

- この例の `requirements.txt` ファイルを Amazon S3 バケットにアップロードする方法について詳しくは、「Python 依存関係のインストール」をご覧ください。
- この例の DAG コードを Amazon S3 バケットの `dags` フォルダにアップロードする方法については、「DAG の追加と更新」を参照してください。
- サンプルスクリプトやその他の pymssql モジュールの例を参照してください。
- mssql_operator を使用した特定の Microsoft SQL データベースでの SQL コード実行については、「Apache Airflow リファレンスガイド」を参照してください。

Amazon MWAA と Amazon EMR を併用する

以下のコード・サンプルは、Amazon EMR と Amazon Managed Workflows for Apache Airflow を使用して統合を有効にする方法を示しています。

トピック

バージョン

- このページのサンプルコードは「Python 3.7」の Apache Airflow v1 で使用できます。

コードサンプル

```
    """
    Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.

    Permission is hereby granted, free of charge, to any person obtaining a copy of
    this software and associated documentation files (the "Software"), to deal in
    the Software without restriction, including without limitation the rights to
    use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of
    the Software, and to permit persons to whom the Software is furnished to do so.

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS
    FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
    COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER
    IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
    CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
    """
    from airflow import DAG

    from airflow.contrib.operators.emr_add_steps_operator import EmrAddStepsOperator
    from airflow.contrib.operators.emr_create_job_flow_operator import
EmrCreateJobFlowOperator
    from airflow.contrib.sensors.emr_step_sensor import EmrStepSensor

    from airflow.utils.dates import days_ago
    from datetime import timedelta
    import os

    DAG_ID = os.path.basename(__file__).replace(".py", "")

    DEFAULT_ARGS = {
        'owner': 'airflow',
        'depends_on_past': False,
```

```
        'email': ['airflow@example.com'],
        'email_on_failure': False,
        'email_on_retry': False,
    }

    SPARK_STEPS = [
        {
            'Name': 'calculate_pi',
            'ActionOnFailure': 'CONTINUE',
            'HadoopJarStep': {
                'Jar': 'command-runner.jar',
                'Args': ['/usr/lib/spark/bin/run-example', 'SparkPi', '10'],
            },
        }
    ]

    JOB_FLOW_OVERRIDES = {
        'Name': 'my-demo-cluster',
        'ReleaseLabel': 'emr-5.30.1',
        'Applications': [
            {
                'Name': 'Spark'
            },
        ],
        'Instances': {
            'InstanceGroups': [
                {
                    'Name': "Master nodes",
                    'Market': 'ON_DEMAND',
                    'InstanceRole': 'MASTER',
                    'InstanceType': 'm5.xlarge',
                    'InstanceCount': 1,
                },
                {
                    'Name': "Slave nodes",
                    'Market': 'ON_DEMAND',
                    'InstanceRole': 'CORE',
                    'InstanceType': 'm5.xlarge',
                    'InstanceCount': 2,
                }
            ],
            'KeepJobFlowAliveWhenNoSteps': False,
            'TerminationProtected': False,
            'Ec2KeyName': 'mykeypair',
```

```
        },
        'VisibleToAllUsers': True,
        'JobFlowRole': 'EMR_EC2_DefaultRole',
        'ServiceRole': 'EMR_DefaultRole'
    }

    with DAG(
        dag_id=DAG_ID,
        default_args=DEFAULT_ARGS,
        dagrun_timeout=timedelta(hours=2),
        start_date=days_ago(1),
        schedule_interval='@once',
        tags=['emr'],
    ) as dag:

        cluster_creator = EmrCreateJobFlowOperator(
            task_id='create_job_flow',
            job_flow_overrides=JOB_FLOW_OVERRIDES
        )

        step_adder = EmrAddStepsOperator(
            task_id='add_steps',
            job_flow_id="{{ task_instance.xcom_pull(task_ids='create_job_flow',
key='return_value') }}",
            aws_conn_id='aws_default',
            steps=SPARK_STEPS,
        )

        step_checker = EmrStepSensor(
            task_id='watch_step',
            job_flow_id="{{ task_instance.xcom_pull('create_job_flow',
key='return_value') }}",
            step_id="{{ task_instance.xcom_pull(task_ids='add_steps',
key='return_value')[0] }}",
            aws_conn_id='aws_default',
        )

        cluster_creator >> step_adder >> step_checker
```

Amazon EKS での Amazon MWAA の使用

以下のサンプルは、Amazon EKS で Apache Airflow 用の Amazon マネージドワークフローを使用する方法を示しています。

トピック

バージョン

- このページのサンプルコードは「Python 3.7」の Apache Airflow v1 で使用できます。
- このページのコード例は、「Python 3.10」で Apache Airflow v2 以上と共に使用可能です。

前提条件

このトピックの例を使用するには、以下が必要です。

- 「Amazon MWAA 環境」。
- eksctl。詳細については、「eksctl のインストール」 を参照してください。
- kubectl。詳細については、「kubectl のインストールとセットアップ」 を参照してください。eksctl と共にインストールされる場合もあります。
- Amazon MWAA 環境を作成したリージョンの EC2 key pair。詳細については、「key pair 作成またはインポート」 を参照してください。

> **Note**
>
> eksctl コマンドを使用するときに、--profile を含めたデフォルト以外のプロファイルを指定できます。

Amazon EC2 用のパブリックキーを作成します

以下のコマンドを使用して、プライベートキーからパブリックキーを作成します。

```
ssh-keygen -y -f myprivatekey.pem > mypublickey.pub
```

詳しくは、「キーペアのパブリックキーを取得する」を参照してください。

クラスターを作成します

次のコマンドを使用してクラスターを作成します。クラスターにカスタム名を付けたい場合や、別のリージョンで作成したい場合は、名前とリージョンの値を置き換えてください。 クラスターは、Amazon MWAA 環境を作成したのと同じリージョン内で作成する必要があります。 サブネットの値を、Amazon MWAA に使用している Amazon VPC ネットワークのサブネットと一致する ように置き換えます。ssh-public-key の値は、使用するキーと一致するように置き換えてください。同じリージョン内にある Amazon EC2 の既存のキーを使用するか、Amazon MWAA 環境を作成したのと同じリージョンで新しいキーを作成できます。

```
eksctl create cluster \
--name mwaa-eks \
--region us-west-2 \
--version 1.18 \
--nodegroup-name linux-nodes \
--nodes 3 \
--nodes-min 1 \
--nodes-max 4 \
--with-oidc \
--ssh-access \
--ssh-public-key MyPublicKey \
--managed \
--vpc-public-subnets "subnet-1111111111111111, subnet-2222222222222222" \
--vpc-private-subnets "subnet-33333333333333333, subnet-44444444444444444"
```

クラスターの作成が完了するまで。少し時間がかかります。完了後、以下のコマンドを使用して、クラスターが正常に作成され、IAM OIDC プロバイダーが設定されていることを確認できます。

```
eksctl utils associate-iam-oidc-provider \
--region us-west-2 \
--cluster mwaa-eks \
--approve
```

mwaa 名前空間を作成します。

クラスターが正常に作成されたことを確認できたあと、以下のコマンドを使用してポッドの名前空間を作成します。

```
kubectl create namespace mwaa
```

mwaa 名前空間のロールを作成します。

名前空間を作成した後、MWAA 名前空間でポッドを実行できる EKS 上の Amazon MWAA ユーザー用のロールとロールバインディングを作成します。名前空間に別の名前を使用した場合は、-n *mwaa* の mwaa を使用した名前に置き換えます。

```
cat << EOF | kubectl apply -f - -n mwaa
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: mwaa-role
rules:
  - apiGroups:
      - ""
      - "apps"
      - "batch"
      - "extensions"
    resources:
      - "jobs"
      - "pods"
      - "pods/attach"
      - "pods/exec"
      - "pods/log"
      - "pods/portforward"
      - "secrets"
```

```
      - "services"
    verbs:
      - "create"
      - "delete"
      - "describe"
      - "get"
      - "list"
      - "patch"
      - "update"
---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: mwaa-role-binding
subjects:
- kind: User
  name: mwaa-service
roleRef:
  kind: Role
  name: mwaa-role
  apiGroup: rbac.authorization.k8s.io
EOF
```

次のコマンドを実行して、新しいロールが Amazon EKS クラスターにアクセスできることを確認します。 *mwaa* を使用していない場合は、必ず正しい名前を使用してください。

```
kubectl get pods -n mwaa --as mwaa-service
```

以下のメッセージが返ってくるはずだ：

```
No resources found in mwaa namespace.
```

Amazon EKS クラスタの IAM ロールを作成してアタッチする

IAM による認証に使用できるように、IAM ロールを作成して Amazon EKS (k8s) クラスターにバインドする必要があります。 ロールはクラスターへのログインにのみ使用され、コンソールや API コールの権限はありません。

「Amazon MWAA 実行ロール」 のステップを使用して Amazon MWAA 環境用の新しいロールを作成します。ただし、このトピックで説明されているポリシーを作成して追加せずに、次のポリシーを追加します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "airflow:PublishMetrics",
            "Resource": "arn:aws:airflow:${MWAA_REGION}:${ACCOUNT_NUMBER}:environment/
${MWAA_ENV_NAME}"
        },
        {
            "Effect": "Deny",
            "Action": "s3:ListAllMyBuckets",
            "Resource": [
                "arn:aws:s3:::{MWAA_S3_BUCKET}",
                "arn:aws:s3:::{MWAA_S3_BUCKET}/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject*",
                "s3:GetBucket*",
                "s3:List*"
            ],
            "Resource": [
                "arn:aws:s3:::{MWAA_S3_BUCKET}",
                "arn:aws:s3:::{MWAA_S3_BUCKET}/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:CreateLogGroup",
                "logs:PutLogEvents",
                "logs:GetLogEvents",
                "logs:GetLogRecord",
                "logs:GetLogGroupFields",
                "logs:GetQueryResults",
                "logs:DescribeLogGroups"
            ],
            "Resource": [
```

```
                "arn:aws:logs:${MWAA_REGION}:${ACCOUNT_NUMBER}:log-group:airflow-
${MWAA_ENV_NAME}-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "cloudwatch:PutMetricData",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "sqs:ChangeMessageVisibility",
                "sqs:DeleteMessage",
                "sqs:GetQueueAttributes",
                "sqs:GetQueueUrl",
                "sqs:ReceiveMessage",
                "sqs:SendMessage"
            ],
            "Resource": "arn:aws:sqs:${MWAA_REGION}:*:airflow-celery-*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey",
                "kms:GenerateDataKey*",
                "kms:Encrypt"
            ],
            "NotResource": "arn:aws:kms:*:${ACCOUNT_NUMBER}:key/*",
            "Condition": {
                "StringLike": {
                    "kms:ViaService": [
                        "sqs.${MWAA_REGION}.amazonaws.com"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "eks:DescribeCluster"
            ],
```

```
            "Resource": "arn:aws:eks:${MWAA_REGION}:${ACCOUNT_NUMBER}:cluster/
${EKS_CLUSTER_NAME}"
        }
    ]
}
```

ロールを作成し、環境の実行ロールとして使用するように Amazon MWAA 環境を編集します。ロールを変更するには、使用する環境を編集します。[権限] で実行ロールを選択します。

既知の問題：

- Amazon EKS での認証に失敗するサブパスを持つロール ARN には既知の問題があります。この問題の回避策は、Amazon MWAA 自体が作成したサービスロールを使用するのではなく、手動でサービスロールを作成することです。　詳しくは、「aws-auth configmapのARNにパスが含まれている場合、パスを持つロールは機能しない」 を参照してください。
- Amazon MWAA サービスの一覧が IAM で利用できない場合は、Amazon EC2 などの代替サービスポリシーを選択し、ロールの信頼ポリシーを次のように更新する必要があります。

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

詳細については、「IAM ロールで信頼ポリシーを使用する方法」 を参照してください。

要件.txt ファイルを作成します

このセクションのサンプルコードを使用するには、以下のデータベースオプションのいずれかを `requirements.txt` に追加していることを確認してください。詳細については、「Python 依存関係のインストール」を参照してください。

Apache Airflow v2

```
kubernetes
apache-airflow[cncf.kubernetes]==3.0.0
```

Apache Airflow v1

```
awscli
kubernetes==12.0.1
```

Amazon EKS 用のアイデンティティマッピングを作成します。

次のコマンドで作成したロールの ARN を使用して、Amazon EKS の ID マッピングを作成します。リージョン *your-region* を、環境を作成したリージョンに変更します。ロールの ARN を置き換え、最後に *mwaa-execution-role* ご使用の環境の実行ロールに置き換えます。

```
eksctl create iamidentitymapping \
--region your-region \
--cluster mwaa-eks \
--arn arn:aws:iam::111222333444:role/mwaa-execution-role \
--username mwaa-service
```

kubeconfig の作成

kubeconfig を作成するには以下のコマンドを使用する：

```
aws eks update-kubeconfig \
--region us-west-2 \
--kubeconfig ./kube_config.yaml \
--name mwaa-eks \
--alias aws
```

実行時に特定のプロファイルを使用した場合は、update-kubeconfig kube_config.yaml env: ファイルに追加されたセクションを削除して Amazon MWAA で正しく動作させる必要があります。そのためには、以下をファイルから削除して保存します。

```
env:
- name: AWS_PROFILE
  value: profile_name
```

DAG を作成する

次のコード例を使用して、DAG の mwaa_pod_example.py などの Python ファイルを作成します。

Apache Airflow v2

```
"""
Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the "Software"), to deal in
the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of
the Software, and to permit persons to whom the Software is furnished to do so.
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS
FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER
IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
"""
from airflow import DAG
from datetime import datetime
from airflow.providers.cncf.kubernetes.operators.kubernetes_pod import
 KubernetesPodOperator

default_args = {
   'owner': 'aws',
   'depends_on_past': False,
   'start_date': datetime(2019, 2, 20),
   'provide_context': True
}

dag = DAG(
```

```
   'kubernetes_pod_example', default_args=default_args, schedule_interval=None)

#use a kube_config stored in s3 dags folder for now
kube_config_path = '/usr/local/airflow/dags/kube_config.yaml'

podRun = KubernetesPodOperator(
                        namespace="mwaa",
                        image="ubuntu:18.04",
                        cmds=["bash"],
                        arguments=["-c", "ls"],
                        labels={"foo": "bar"},
                        name="mwaa-pod-test",
                        task_id="pod-task",
                        get_logs=True,
                        dag=dag,
                        is_delete_operator_pod=False,
                        config_file=kube_config_path,
                        in_cluster=False,
                        cluster_context='aws'
                        )
```

Apache Airflow v1

```
"""
Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the "Software"), to deal in
the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of
the Software, and to permit persons to whom the Software is furnished to do so.
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS
FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER
IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
"""
from airflow import DAG
from datetime import datetime
from airflow.contrib.operators.kubernetes_pod_operator import KubernetesPodOperator

default_args = {
   'owner': 'aws',
```

```
    'depends_on_past': False,
    'start_date': datetime(2019, 2, 20),
    'provide_context': True
}

dag = DAG(
    'kubernetes_pod_example', default_args=default_args, schedule_interval=None)

#use a kube_config stored in s3 dags folder for now
kube_config_path = '/usr/local/airflow/dags/kube_config.yaml'

podRun = KubernetesPodOperator(
                        namespace="mwaa",
                        image="ubuntu:18.04",
                        cmds=["bash"],
                        arguments=["-c", "ls"],
                        labels={"foo": "bar"},
                        name="mwaa-pod-test",
                        task_id="pod-task",
                        get_logs=True,
                        dag=dag,
                        is_delete_operator_pod=False,
                        config_file=kube_config_path,
                        in_cluster=False,
                        cluster_context='aws'
                        )
```

DAG と **kube_config.yaml** を Amazon S3 バケットに追加します

作成した DAG と kube_config.yaml ファイルを Amazon MWAA 環境の Amazon S3 バケットに配置します。Amazon S3 コンソールまたは AWS Command Line Interface を使用して、ファイルをバケットに入れることができます。

サンプルを有効にして、トリガーしてください。

Apache Airflow で、サンプルを有効にしてからトリガーします。

実行して正常に完了したら、次のコマンドを使用して、ポッドを確認します。

```
kubectl get pods -n mwaa
```

次のような出力が表示されます。

```
NAME READY STATUS RESTARTS AGE
mwaa-pod-test-aa11bb22cc3344445555666677778888 0/1 Completed 0 2m23s
```

次に、以下のコマンドを使用して、ポッドの出力を確認できます。名前の値を、前のコマンドで返された値に置き換えます。

```
kubectl logs -n mwaa mwaa-pod-test-aa11bb22cc3344445555666677778888
```

ECSOperator を使用して Amazon ECS に接続します。

このトピックでは、ECSOperator を使用して、Amazon MWAA から Amazon Elastic Container Service (Amazon ECS) コンテナに接続する方法について説明します。次のステップでは、必要なアクセス権限を環境の実行ロールに追加し、AWS CloudFormation テンプレートを使用して Amazon ECS Fargate クラスターを作成し、最後に新しいクラスターに接続する DAG を作成してアップロードします。

トピック

バージョン

- このページのコード例は、「Python 3.10」で Apache Airflow v2 以上と共に使用可能です。

前提条件

このページのサンプルコードを使用するには、以下が必要です。

- 「Amazon MWAA 環境」。

許可

- 環境の実行ロールには、Amazon ECS でタスクを実行する権限が必要です。「AmazonECS_FullAccess」 AWS 管理ポリシーを実行ロールにアタッチするか、次のポリシーを作成して、実行ロールにアタッチできます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "ecs:RunTask",
                "ecs:DescribeTasks"
            ],
            "Resource": "*"
        },
        {
            "Action": "iam:PassRole",
            "Effect": "Allow",
            "Resource": [
                "*"
            ],
            "Condition": {
                "StringLike": {
                    "iam:PassedToService": "ecs-tasks.amazonaws.com"
                }
            }
        }
    ]
}
```

- Amazon ECS でタスクを実行するために必要なプレミッションを追加することに加えて、Amazon MWAA 実行ロールの CloudWatch Logs ポリシー・ステートメントを変更して、Amazon ECS タスク・ログ・グループへのアクセスを許可する必要があります。Amazon ECS ロググループは、AWS CloudFormation テンプレートによって「the section called “Amazon ECS クラスターを作成する”」 で作成されます。

```
{
    "Effect": "Allow",
    "Action": [
```

```
            "logs:CreateLogStream",
            "logs:CreateLogGroup",
            "logs:PutLogEvents",
            "logs:GetLogEvents",
            "logs:GetLogRecord",
            "logs:GetLogGroupFields",
            "logs:GetQueryResults"
        ],
        "Resource": [
            "arn:aws:logs:region:account-id:log-group:airflow-environment-name-*",
            "arn:aws:logs:*:*:log-group:ecs-mwaa-group:*"
        ]
}
```

Amazon MWAA 実行ロールとポリシーをアタッチする方法の詳細については、「実行ロール」 を参照してください。

Amazon ECS クラスターを作成する

以下の AWS CloudFormation テンプレートを使用して、Amazon ECS Fargate クラスターを構築して Amazon MWAA ワークフローで使用します。詳細については、Amazon Elastic Container Service デベロッパーガイドの「タスク定義の作成」 を参照してください。

1. 以下のコードで JSON ファイルを作成し、ecs-mwaa-cfn.json として保存します。

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Description": "This template deploys an ECS Fargate cluster with an Amazon
 Linux image as a test for MWAA.",
    "Parameters": {
        "VpcId": {
            "Type": "AWS::EC2::VPC::Id",
            "Description": "Select a VPC that allows instances access to ECR, as
 used with MWAA."
        },
        "SubnetIds": {
            "Type": "List<AWS::EC2::Subnet::Id>",
            "Description": "Select at two private subnets in your selected VPC, as
 used with MWAA."
        },
        "SecurityGroups": {
```

```
            "Type": "List<AWS::EC2::SecurityGroup::Id>",
            "Description": "Select at least one security group in your selected
 VPC, as used with MWAA."
        }
    },
    "Resources": {
        "Cluster": {
            "Type": "AWS::ECS::Cluster",
            "Properties": {
                "ClusterName": {
                    "Fn::Sub": "${AWS::StackName}-cluster"
                }
            }
        },
        "LogGroup": {
            "Type": "AWS::Logs::LogGroup",
            "Properties": {
                "LogGroupName": {
                    "Ref": "AWS::StackName"
                },
                "RetentionInDays": 30
            }
        },
        "ExecutionRole": {
            "Type": "AWS::IAM::Role",
            "Properties": {
                "AssumeRolePolicyDocument": {
                    "Statement": [
                        {
                            "Effect": "Allow",
                            "Principal": {
                                "Service": "ecs-tasks.amazonaws.com"
                            },
                            "Action": "sts:AssumeRole"
                        }
                    ]
                },
                "ManagedPolicyArns": [
                    "arn:aws:iam::aws:policy/service-role/
AmazonECSTaskExecutionRolePolicy"
                ]
            }
        },
        "TaskDefinition": {
```

```
            "Type": "AWS::ECS::TaskDefinition",
            "Properties": {
                "Family": {
                    "Fn::Sub": "${AWS::StackName}-task"
                },
                "Cpu": 2048,
                "Memory": 4096,
                "NetworkMode": "awsvpc",
                "ExecutionRoleArn": {
                    "Ref": "ExecutionRole"
                },
                "ContainerDefinitions": [
                    {
                        "Name": {
                            "Fn::Sub": "${AWS::StackName}-container"
                        },
                        "Image": "137112412989.dkr.ecr.us-east-1.amazonaws.com/
amazonlinux:latest",
                        "PortMappings": [
                            {
                                "Protocol": "tcp",
                                "ContainerPort": 8080,
                                "HostPort": 8080
                            }
                        ],
                        "LogConfiguration": {
                            "LogDriver": "awslogs",
                            "Options": {
                                "awslogs-region": {
                                    "Ref": "AWS::Region"
                                },
                                "awslogs-group": {
                                    "Ref": "LogGroup"
                                },
                                "awslogs-stream-prefix": "ecs"
                            }
                        }
                    }
                ],
                "RequiresCompatibilities": [
                    "FARGATE"
                ]
            }
        },
```

```
        "Service": {
            "Type": "AWS::ECS::Service",
            "Properties": {
                "ServiceName": {
                    "Fn::Sub": "${AWS::StackName}-service"
                },
                "Cluster": {
                    "Ref": "Cluster"
                },
                "TaskDefinition": {
                    "Ref": "TaskDefinition"
                },
                "DesiredCount": 1,
                "LaunchType": "FARGATE",
                "PlatformVersion": "1.3.0",
                "NetworkConfiguration": {
                    "AwsvpcConfiguration": {
                        "AssignPublicIp": "ENABLED",
                        "Subnets": {
                            "Ref": "SubnetIds"
                        },
                        "SecurityGroups": {
                            "Ref": "SecurityGroups"
                        }
                    }
                }
            }
        }
    }
}
```

2. コマンドプロンプトで、以下の AWS CLI コマンドを使用して新しいスタックを作成します。SecurityGroups と SubnetIds の値を Amazon MWAA 環境のセキュリティグループとサブネットの値に置き換える必要があります。

```
$ aws cloudformation create-stack \
--stack-name my-ecs-stack --template-body file://ecs-mwaa-cfn.json \
--parameters ParameterKey=SecurityGroups,ParameterValue=your-mwaa-security-group \
ParameterKey=SubnetIds,ParameterValue=your-mwaa-subnet-1\\,your-mwaa-subnet-1 \
--capabilities CAPABILITY_IAM
```

あるいは、以下のシェルスクリプトを使用できます。このスクリプトは、get-environment AWS CLI コマンドを使用して環境のセキュリティグループとサブネットに必要な値を取得し、それに応じてスタックを作成します。スクリプトを実行するには、以下のようにします。

a. スクリプトをコピーして、AWS CloudFormation テンプレートと同じディレクトリに ecs-stack-helper.sh として保存してください。

```
#!/bin/bash

joinByString() {
  local separator="$1"
  shift
  local first="$1"
  shift
  printf "%s" "$first" "${@/#/$separator}"
}

response=$(aws mwaa get-environment --name $1)

securityGroupId=$(echo "$response" | jq -r
 '.Environment.NetworkConfiguration.SecurityGroupIds[]')
subnetIds=$(joinByString '\,' $(echo "$response" | jq -r
 '.Environment.NetworkConfiguration.SubnetIds[]'))

aws cloudformation create-stack --stack-name $2 --template-body file://ecs-
cfn.json \
--parameters ParameterKey=SecurityGroups,ParameterValue=$securityGroupId \
ParameterKey=SubnetIds,ParameterValue=$subnetIds \
--capabilities CAPABILITY_IAM
```

b. 以下のコマンドを使ってスクリプトを実行します。environment-name と stack-name をあなたの情報に置き換えます。

```
$ chmod +x ecs-stack-helper.sh
$ ./ecs-stack-helper.bash environment-name stack-name
```

成功した場合は、新しい AWS CloudFormation スタック ID を示す次の出力が表示されます。

```
{
```

```
    "StackId": "arn:aws:cloudformation:us-west-2:123456789012:stack/my-ecs-
stack/123456e7-8ab9-01cd-b2fb-36cce63786c9"
}
```

AWS CloudFormation スタックが完了し、AWS がAmazon ECSリソースをプロビジョニングしたら、DAGを作成してアップロードする準備が整います。

コードサンプル

1. コマンドプロンプトを開き、DAG コードが保存されているディレクトリに移動します。例:

```
cd dags
```

2. 以下のコードサンプルの内容をコピーして、mwaa-ecs-operator.py としてローカルに保存し、新しいDAGをAmazon S3にアップロードしてください。

```
from http import client
from airflow import DAG
from airflow.providers.amazon.aws.operators.ecs import ECSOperator
from airflow.utils.dates import days_ago
import boto3

CLUSTER_NAME="mwaa-ecs-test-cluster" #Replace value for CLUSTER_NAME with your
 information.
CONTAINER_NAME="mwaa-ecs-test-container" #Replace value for CONTAINER_NAME with
 your information.
LAUNCH_TYPE="FARGATE"

with DAG(
    dag_id = "ecs_fargate_dag",
    schedule_interval=None,
    catchup=False,
    start_date=days_ago(1)
) as dag:
    client=boto3.client('ecs')
    services=client.list_services(cluster=CLUSTER_NAME,launchType=LAUNCH_TYPE)

 service=client.describe_services(cluster=CLUSTER_NAME,services=services['serviceArns'])

    ecs_operator_task = ECSOperator(
        task_id = "ecs_operator_task",
```

```
        dag=dag,
        cluster=CLUSTER_NAME,
        task_definition=service['services'][0]['taskDefinition'],
        launch_type=LAUNCH_TYPE,
        overrides={
            "containerOverrides":[
                {
                    "name":CONTAINER_NAME,
                    "command":["ls", "-l", "/"],
                },
            ],
        },

        network_configuration=service['services'][0]['networkConfiguration'],
        awslogs_group="mwaa-ecs-zero",
        awslogs_stream_prefix=f"ecs/{CONTAINER_NAME}",
    )
```

> **Note**
>
> サンプルの DAG では、awslogs_group について、Amazon ECSタスクロググループの名前に合わせてロググループを変更する必要があるかもしれません。この例では、mwaa-ecs-zero という名前のロググループを想定しています。awslogs_stream_prefix には Amazon ECS タスクログストリームのプレフィックスを使用してください。この例では、ログストリームのプレフィックスが ecs であることを前提としています。

3. 以下の AWS CLI コマンドを実行して、DAG を環境のバケットにコピーし、次に Apache Airflow UI を使用して DAG をトリガーします。

```
$ aws s3 cp your-dag.py s3://your-environment-bucket/dags/
```

4. 成功した場合、ecs_operator_task DAG 内の ecs_fargate_dag タスクログで次のような出力が表示されます。

```
[2022-01-01, 12:00:00 UTC] {{ecs.py:300}} INFO - Running ECS Task -
Task definition: arn:aws:ecs:us-west-2:123456789012:task-definition/mwaa-ecs-test-
task:1 - on cluster mwaa-ecs-test-cluster
[2022-01-01, 12:00:00 UTC] {{ecs-operator-test.py:302}} INFO - ECSOperator
 overrides:
```

```
{'containerOverrides': [{'name': 'mwaa-ecs-test-container', 'command': ['ls', '-l',
 '/']}]}
.
.
.
[2022-01-01, 12:00:00 UTC] {{ecs.py:379}} INFO - ECS task ID is:
 e012340b5e1b43c6a757cf012c635935
[2022-01-01, 12:00:00 UTC] {{ecs.py:313}} INFO - Starting ECS Task Log Fetcher
[2022-01-01, 12:00:00 UTC] {{ecs.py:119}} INFO - [2022-07-19, 17:54:03 UTC] total
 52
[2022-01-01, 12:00:00 UTC] {{ecs.py:119}} INFO - [2022-07-19, 17:54:03 UTC]
 lrwxrwxrwx   1 root root    7 Jun 13 18:51 bin -> usr/bin
[2022-01-01, 12:00:00 UTC] {{ecs.py:119}} INFO - [2022-07-19, 17:54:03 UTC] dr-xr-
xr-x   2 root root 4096 Apr  9  2019 boot
[2022-01-01, 12:00:00 UTC] {{ecs.py:119}} INFO - [2022-07-19, 17:54:03 UTC] drwxr-
xr-x   5 root root  340 Jul 19 17:54 dev
[2022-01-01, 12:00:00 UTC] {{ecs.py:119}} INFO - [2022-07-19, 17:54:03 UTC] drwxr-
xr-x   1 root root 4096 Jul 19 17:54 etc
[2022-01-01, 12:00:00 UTC] {{ecs.py:119}} INFO - [2022-07-19, 17:54:03 UTC] drwxr-
xr-x   2 root root 4096 Apr  9  2019 home
[2022-01-01, 12:00:00 UTC] {{ecs.py:119}} INFO - [2022-07-19, 17:54:03 UTC]
 lrwxrwxrwx   1 root root    7 Jun 13 18:51 lib -> usr/lib
[2022-01-01, 12:00:00 UTC] {{ecs.py:119}} INFO - [2022-07-19, 17:54:03 UTC]
 lrwxrwxrwx   1 root root    9 Jun 13 18:51 lib64 -> usr/lib64
[2022-01-01, 12:00:00 UTC] {{ecs.py:119}} INFO - [2022-07-19, 17:54:03 UTC] drwxr-
xr-x   2 root root 4096 Jun 13 18:51 local
[2022-01-01, 12:00:00 UTC] {{ecs.py:119}} INFO - [2022-07-19, 17:54:03 UTC] drwxr-
xr-x   2 root root 4096 Apr  9  2019 media
[2022-01-01, 12:00:00 UTC] {{ecs.py:119}} INFO - [2022-07-19, 17:54:03 UTC] drwxr-
xr-x   2 root root 4096 Apr  9  2019 mnt
[2022-01-01, 12:00:00 UTC] {{ecs.py:119}} INFO - [2022-07-19, 17:54:03 UTC] drwxr-
xr-x   2 root root 4096 Apr  9  2019 opt
[2022-01-01, 12:00:00 UTC] {{ecs.py:119}} INFO - [2022-07-19, 17:54:03 UTC] dr-xr-
xr-x 103 root root    0 Jul 19 17:54 proc
[2022-01-01, 12:00:00 UTC] {{ecs.py:119}} INFO - [2022-07-19, 17:54:03 UTC] dr-xr-
x-\-\-   2 root root 4096 Apr  9  2019 root
[2022-01-01, 12:00:00 UTC] {{ecs.py:119}} INFO - [2022-07-19, 17:54:03 UTC] drwxr-
xr-x   2 root root 4096 Jun 13 18:52 run
[2022-01-01, 12:00:00 UTC] {{ecs.py:119}} INFO - [2022-07-19, 17:54:03 UTC]
 lrwxrwxrwx   1 root root    8 Jun 13 18:51 sbin -> usr/sbin
[2022-01-01, 12:00:00 UTC] {{ecs.py:119}} INFO - [2022-07-19, 17:54:03 UTC] drwxr-
xr-x   2 root root 4096 Apr  9  2019 srv
[2022-01-01, 12:00:00 UTC] {{ecs.py:119}} INFO - [2022-07-19, 17:54:03 UTC] dr-xr-
xr-x  13 root root    0 Jul 19 17:54 sys
```

```
[2022-01-01, 12:00:00 UTC] {{ecs.py:119}} INFO - [2022-07-19, 17:54:03 UTC]
 drwxrwxrwt   2 root root 4096 Jun 13 18:51 tmp
[2022-01-01, 12:00:00 UTC] {{ecs.py:119}} INFO - [2022-07-19, 17:54:03 UTC] drwxr-
xr-x  13 root root 4096 Jun 13 18:51 usr
[2022-01-01, 12:00:00 UTC] {{ecs.py:119}} INFO - [2022-07-19, 17:54:03 UTC] drwxr-
xr-x  18 root root 4096 Jun 13 18:52 var
.
.
.
[2022-01-01, 12:00:00 UTC] {{ecs.py:328}} INFO - ECS Task has been successfully
 executed
```

Amazon MWAA での dbt の使用

このトピックでは、Amazon MWAA で dbt と Postgres を使用する方法を示します。次のステップでは、必要な依存関係を `requirements.txt` に追加し、サンプルの dbt プロジェクトを環境の Amazon S3 バケットにアップロードします。次に、サンプル DAG を使用して Amazon MWAA が依存関係をインストールしたことを確認し、最後に `BashOperator` を使用して dbt プロジェクトを実行します。

トピック

Version

- このページのコード例は、「Python 3.10」で Apache Airflow v2 以上と共に使用可能です。

前提条件

次の手順を完了するには、以下のものが必要です。

- Apache Airflow v2.2.2 を使用する Amazon MWAA 環境。このサンプルは v2.2.2 で作成され、テストされています。他の Apache Airflow バージョンで使用するためには、サンプルを変更する必要がある場合があります。
- dbt プロジェクトのサンプル。Amazon MWAA で dbt の使用を開始するには、フォークを作成し、dbt-labs リポジトリから dbt スタータープロジェクトのクローンを作成できます。 GitHub

依存関係

dbt で Amazon MWAA を使用するには、次のスタートアップスクリプトを環境に追加します。詳細については、「Amazon MWAA でのスタートアップスクリプトの使用」を参照してください。

```
#!/bin/bash

if [[ "${MWAA_AIRFLOW_COMPONENT}" != "worker" ]]
then
    exit 0
fi

echo "------------------------------"
echo "Installing virtual Python env"
echo "------------------------------"

pip3 install --upgrade pip

echo "Current Python version:"
python3 --version
echo "..."

sudo pip3 install --user virtualenv
sudo mkdir python3-virtualenv
cd python3-virtualenv
sudo python3 -m venv dbt-env
sudo chmod -R 777 *

echo "------------------------------"
echo "Activating venv in"
$DBT_ENV_PATH
echo "------------------------------"

source dbt-env/bin/activate
pip3 list
```

```
echo "------------------------------"
echo "Installing libraries..."
echo "------------------------------"

# do not use sudo, as it will install outside the venv
pip3 install dbt-redshift==1.6.1 dbt-postgres==1.6.1

echo "------------------------------"
echo "Venv libraries..."
echo "------------------------------"

pip3 list
dbt --version

echo "------------------------------"
echo "Deactivating venv..."
echo "------------------------------"

deactivate
```

以下のセクションでは、dbt プロジェクトディレクトリを Amazon S3 にアップロードし、Amazon MWAA が必要な dbt 依存関係を正常にインストールしたかどうかを検証する DAG を実行します。

DBT プロジェクトを Amazon S3 にアップロードする

Amazon MWAA 環境で dbt プロジェクトを使用できるようにするには、プロジェクトディレクトリ全体を環境の`dags`フォルダにアップロードできます。環境が更新されると、Amazon MWAA は dbt ディレクトリをローカル`usr/local/airflow/dags/`フォルダにダウンロードします。

DBT プロジェクトを Amazon S3 にアップロードするには

1. dbt スタータープロジェクトをクローンしたディレクトリに移動します。
2. 次の Amazon S3 AWS CLI コマンドを実行して、 `--recursive`パラメータを使用してプロジェクトのコンテンツを環境の `dags`フォルダに再帰的にコピーします。このコマンドは、`dbt`と呼ばれるサブディレクトリを作成し、これをすべての`dbt`プロジェクトに使用できます。サブディレクトリが既に存在する場合、プロジェクトファイルは既存のディレクトリにコピーされ、新しいディレクトリは作成されません。このコマンドは、この特定のスターターのために dbt ディレクトリ内にサブディレクトリも作成します。

```
$ aws s3 cp dbt-starter-project s3://mwaa-bucket/dags/dbt/dbt-starter-project --
recursive
```

プロジェクトのサブディレクトリに異なる名前を使用して、親 dbt ディレクトリ内の複数の dbt プロジェクトを整理できます。

DAG を使用して dbt 依存関係のインストールを検証します。

次の DAGは、BashOperator を使用し、requirements.txt で指定されたdbtの依存関係を正常にインストールしたかどうかを Amazon MWAA が確認します。

```
from airflow import DAG
from airflow.operators.bash_operator import BashOperator
from airflow.utils.dates import days_ago

with DAG(dag_id="dbt-installation-test", schedule_interval=None, catchup=False,
 start_date=days_ago(1)) as dag:
    cli_command = BashOperator(
        task_id="bash_command",
        bash_command="/usr/local/airflow/.local/bin/dbt --version"
    )
```

以下を実行してタスクログを表示し、dbt とその依存関係がインストールされていることを確認します。

1. Amazon MWAA コンソールに移動し、使用可能な環境のリストから [Airflow UI を開く] を選択します。
2. Apache Airflow UI のリストから dbt-installation-test DAG を探し、Last Run列の下にある日付を選択して、最後に成功したタスクを開きます。
3. グラフビューを使用してタスクを選択し、bash_commandタスクインスタンスの詳細を開きます。
4. [Log] を選択してタスクログを開き、次に、requirements.txt で指定されたdbtのバージョンをログが正常にリストしていることを確認してください。

DAG を使用して dbt プロジェクトを実行します。

次の DAG は、BashOperatorを使用して Amazon S3 にアップロードした dbt プロジェクトをローカル usr/local/airflow/dags/ ディレクトリから書き込み可能な/tmpディレクトリにコピーし、dbt プロジェクトを実行します。bash コマンドは、dbt-starter-projectというタイトルのスターター dbt プロジェクトを想定しています。プロジェクトディレクトリの名前に従ってディレクトリ名を変更します。

```
from airflow import DAG
from airflow.operators.bash_operator import BashOperator
from airflow.utils.dates import days_ago

import os

DAG_ID = os.path.basename(__file__).replace(".py", "")

# assumes all files are in a subfolder of DAGs called dbt

with DAG(dag_id=DAG_ID, schedule_interval=None, catchup=False, start_date=days_ago(1))
 as dag:
    cli_command = BashOperator(
        task_id="bash_command",
        bash_command="source /usr/local/airflow/python3-virtualenv/dbt-env/bin/
activate;\
cp -R /usr/local/airflow/dags/dbt /tmp;\
echo 'listing project files:';\
ls -R /tmp;\
cd /tmp/dbt/mwaa_dbt_test_project;\
/usr/local/airflow/python3-virtualenv/dbt-env/bin/dbt run --project-dir /tmp/dbt/
mwaa_dbt_test_project --profiles-dir ..;\
cat /tmp/dbt_logs/dbt.log;\
rm -rf /tmp/dbt/mwaa_dbt_test_project"
    )
```

AWS ブログとチュートリアル

- Apache Airflow v2.x 向けの Amazon EKS と Amazon MWAA との連携

Amazon Managed Workflows for Apache Airflow のベストプラクティス

このガイドでは、Apache Airflow 用 Amazon マネージドワークフローを使用する場合に推奨されるベストプラクティスについて説明します。

トピック

Amazon MWAA での Apache Airflow のパフォーマンス調整

このページでは、「Amazon MWAA での Apache Airflow 構成オプションの使用」を使用して Apache Airflow 環境向けの Amazon マネージドワークフローのパフォーマンスを調整するために推奨するベストプラクティスについて説明します。

目次

Apache Airflow 構成オプションの追加

以下の手順では、Airflow 構成オプションを環境に追加するステップを説明します。

1. Amazon MWAA コンソールで「環境ページ」を開きます。

2. 環境を選択します。
3. [編集] を選択します。
4. [次へ] をクリックします。
5. Airflow 設定オプションペインで [カスタム設定を追加] を選択します。
6. ドロップダウンリストから構成を選択して値を入力するか、カスタム構成を入力して値を入力します。
7. 追加する設定ごとに [カスタム設定を追加] を選択します。
8. [保存] を選択します。

詳細については、「Amazon MWAA での Apache Airflow 構成オプションの使用」を参照してください。

Apache Airflow スケジューラー

Apache Airflow スケジューラーは Apache Airflow のコアコンポーネントです。スケジューラーに問題があると、DAG の解析やタスクのスケジュール設定ができなくなる可能性があります。Apache Airflow スケジューラーの調整の詳細については、Apache Airflow ドキュメンテーションウェブサイトの「スケジューラーのパフォーマンスの微調整」を参照してください。

パラメータ

このセクションでは、Apache Airflow スケジューラーで使用できる構成オプションとそのユースケースについて説明します。

Apache Airflow v2

Version	設定オプション	デフォルト	[Description] (説明)	ユースケース
v2	「celery.sync_parallelism」	1	Celery エグゼキューターがタスクの状態を同期するために使用するプロセスの数です。	このオプションを使用すると、Celery エグゼキューターが使用するプロセスを制限することでキューの

Version	設定オプション	デフォルト	[Description] (説明)	ユースケース
				競合を防ぐことができます。デフォルトでは、タスクログを CloudWatch ログ1に配信する際のエラーを防ぐために、 値がに設定されています。この値を `0` に設定すると最大数のプロセスを使用することになりますが、タスクログの配信時にエラーが発生する可能性があります。

Version	設定オプション	デフォルト	[Description] (説明)	ユースケース
v2	「scheduler.processor_poll_interval」	1	スケジューラーの「ループ」で DAG ファイルが連続して処理されるまでに待機する秒数。	このオプションを使用すると、DAG 解析結果の取得、タスクの検索とキューへの追加、エグゼキューターでのキュー内のタスクの実行が完了した後に、スケジューラーがスリープする時間を長くすることで、スケジューラーの CPU 使用率を解放できます。この値を増やすと、Apache Airflow v2 の場合は `scheduler.parsing_processes` で、Apache Airflow v1 の場合は `scheduler.max_threads` で、環境で実行されるスケジューラー

Version	設定オプション	デフォルト	[Description] (説明)	ユースケース
				のスレッド数が消費されます。これにより、スケジューラーが DAG を解析する容量が減り、DAG がウェブサーバーに表示されるまでにかかる時間が長くなる可能性があります。
v2	「scheduler.max_dagruns_to_create_per_loop」	10	スケジューラDAGs の最大数。 DagRuns	このオプションを使用すると、スケジューラの「ループ」の最大数を減らすことで、タスクをスケジュールDagRunsするためのリソースを解放できます。

Version	設定オプション	デフォルト	[Description] (説明)	ユースケース
v2	「scheduler.parsing_processes」	2	スケジューラーが DAG をスケジュールするために並列して実行できるスレッドの数。	このオプションを使用すると、スケジューラーが DAG を解析するために並列して実行するプロセスの数を減らすことで、リソースを解放できます。DAG の解析がタスクスケジューリングに影響している場合は、この数を低く抑えることをお勧めします。環境の vCPU 数よりも小さい値を指定する必要があります。詳細については、「制限」を参照してください。

制限

このセクションでは、スケジューラーのデフォルトパラメータを調整する際に考慮すべき制限について説明します。

scheduler.parsing_processes、scheduler.max_threads

1 つの環境クラスの vCPU ごとに 2 つのスレッドを使用できます。環境クラスのスケジューラー用に少なくとも 1 つのスレッドを予約する必要があります。タスクのスケジュールが遅

れていることに気付いた場合は、「環境クラス」を増やす必要があるかもしれません。たとえば、大規模な環境では、スケジューラ用に 4 vCPU の Fargate コンテナインスタンスがあります。つまり、他のプロセスに使用できるスレッドの 7 合計は最大数になります。つまり、2 つのスレッドに 4 つの vCPUs を乗算した値から、スケジューラー自体の 1 を引いた値です。`scheduler.max_threads` および `scheduler.parsing_processes` で指定する値は、環境クラスで使用できるスレッド数 (以下を参照) を超えてはなりません。

- mw1.small — 他のプロセスの 1 スレッド数を超えてはいけません。残りのスレッドはスケジューラー用に予約されています。
- mw1.medium — 他のプロセスの 3 スレッド数を超えてはいけません。残りのスレッドはスケジューラー用に予約されています。
- mw1.large — 他のプロセスの 7 スレッド数を超えてはいけません。残りのスレッドはスケジューラー用に予約されています。

DAG フォルダー

Apache Airflow スケジューラーは、環境内の DAG フォルダーを継続的にスキャンします。含まれている `plugins.zip` ファイル、または「airflow」インポートステートメントを含む Python (`.py`) ファイル。その後、結果の Python DAG オブジェクトは に配置され、DagBagスケジューラがそのファイルを処理して、スケジュールする必要があるタスクがあるかどうかを判断します。DAG ファイルの解析は、そのファイルに実行可能な DAG オブジェクトが含まれているかどうかに関係なく行われます。

パラメータ

このセクションでは、DAGs フォルダーで使用できる構成オプションとそのユースケースについて説明します。

Apache Airflow v2

Version	設定オプション	デフォルト	[Description] (説明)	ユースケース
v2	「scheduler.dag_dir_list_interval」	300 秒	DAGs フォルダーをスキャンして新しいファイルを探す必要のある秒数。	このオプションを使用すると、DAGs フォルダーを解析する秒数を増や

Version	設定オプション	デフォルト	[Description] (説明)	ユースケース
				すことでリソースを解放できます。DAGs フォルダーに大量のファイルがあることが原因で、`total_parse_time` metrics で解析時間が長くなる場合は、この値を増やすことをお勧めします。

Version	設定オプション	デフォルト	[Description] (説明)	ユースケース
v2	「scheduler.min_file_process_interval」	30 秒	スケジューラーが DAG を解析して DAG への更新が反映されるまでの秒数。	このオプションを使用して、DAG を解析する前にスケジューラーが待機する秒数を増やすことで、リソースを解放できます。たとえば、30 の値を指定すると、DAG ファイルは 30 秒ごとに解析されます。お使いの環境の CPU 使用率を下げるには、この数値を高くしておくことをお勧めします。

DAG ファイル

Apache Airflow スケジューラーループの一部として、個々の DAG ファイルが解析されて DAG Python オブジェクトが抽出されます。Apache Airflow v2 以降では、スケジューラーは同時に最大数の「解析プロセス」を解析します。同じファイルが再び解析される前に、`scheduler.min_file_process_interval` で指定した秒数が経過する必要があります。

パラメータ

このセクションでは、Apache Airflow DAG ファイルで使用できる構成オプションとそのユースケースについて説明します。

Apache Airflow v2

Version	設定オプション	デフォルト	[Description] (説明)	ユースケース
v2	「core.dag_file_processor_timeout」	50 秒	DagFileプロセッサが DAG ファイルの処理をタイムアウトするまでの秒数。	このオプションを使用すると、DagFileプロセッサがタイムアウトするまでにかかる時間を長くすることで、リソースを解放できます。DAG 処理ログにタイムアウトが発生し、実行可能な DAG が読み込まれなくなる場合は、この値を増やすことをお勧めします。
v2	「core.dagbag_import_timeout」	30 秒	Python ファイルをインポートするまでの秒数がタイムアウトします。	このオプションを使用すると、Python ファイルをインポートして DAG オブジェクトを抽出する際にスケジューラーがタイムアウトになるまでにかかる時間を増やすことで、リソースを解放で

Version	設定オプション	デフォルト	[Description] (説明)	ユースケース
				きます。このオプションはスケジューラーの「ループ」の一部として処理され、`core.dag_file_processor_timeout` で指定されている値よりも小さい値が含まれている必要があります。

Version	設定オプション	デフォルト	[Description] (説明)	ユースケース
v2	「core.min_serialized_dag_update_interval」	30	データベース内のシリアル化された DAG が更新されるまでの最小秒数。	このオプションを使用すると、データベース内のシリアル化された DAG が更新されるまでの秒数を増やすことで、リソースを解放できます。DAG の数が多い場合、または複雑な DAG がある場合は、この値を増やすことをおすすめします。この値を増やすと、DAG がシリアル化されるときのスケジューラーとデータベースへの負荷が軽減されます。

Version	設定オプション	デフォルト	[Description] (説明)	ユースケース
v2	「core.min_serialized_dag_fetch_interval」	10	に既にロードされているときに、シリアル化された DAG がデータベースから再取得される秒数 DagBag。	このオプションを使用すると、シリアル化された DAG を再フェッチする秒数を増やすことでリソースを解放できます。データベースの「書き込み」速度を下げるには、この値を `core.min_serialized_dag_update_interval` で指定した値より大きくする必要があります。この値を増やすと、DAG がシリアル化される際のウェブサーバーとデータベースの負荷が軽減されます。

タスク

Apache Airflow スケジューラーとワーカーはどちらもタスクのキューイングとデキューに関与します。スケジューラーは、解析済みのスケジュール設定が完了したタスクを [なし] ステータスから [スケジュール済み] ステータスに移行します。同じく Fargate のスケジューラーコンテナーで実行され

ている工グゼキューターは、それらのタスクをキューに入れ、そのステータスを [キューで待機中] に設定します。ワーカーにキャパシティがあると、キューからタスクを取り出してステータスを [実行中] に設定し、その後、タスクが成功したか失敗したかに基づいてステータスを [成功] または [失敗] に変更します。

パラメータ

このセクションでは、Apache Airflow タスクで使用できる構成オプションとそのユースケースについて説明します。

Amazon MWAA がオーバーライドするデフォルトの構成オプションは#でマークされています。

Apache Airflow v2

Version	設定オプション	デフォルト	[Description] (説明)	ユースケース
v2	「コア・パラレリズム」	10000	「実行中」のステータスを持つことができるタスクインスタンスの最大数。	このオプションを使用すると、同時に実行できるタスクインスタンスの数を増やすことでリソースを解放できます。指定する値は、使用可能なワーカーの数にワーカーのタスク密度を「掛けた」ものであるべきです。この値を変更するのは、多数のタスクが「実行中」または「キューで待機中」の状態で停止している場

Version	設定オプション	デフォルト	[Description] (説明)	ユースケース
				合のみにすることをおすすめします。
v2	「core.dag_concurrency」	10000	各 DAG で同時に実行できるタスクインスタンスの数。	このオプションを使用すると、同時に実行できるタスクインスタンスの数を増やすことでリソースを解放できます。たとえば、10 個の並列タスクを含む 100 個の DAG があり、すべての DAG を同時に実行したい場合、利用可能なワーカーの数に `celery.worker_concurrency` でのワーカータスクの密度を「掛け」、それを DAG の数 (100 など) で割って最大並列処理を計算できます。

Version	設定オプション	デフォルト	[Description] (説明)	ユースケース
v2	「core.execute_tasks_new_python_interpreter」	`True`	Apache Airflow が親プロセスをフォークするか、新しい Python プロセスを作成することによってタスクを実行するかどうかを決定します。	`True` に設定すると、Apache Airflow はプラグインに加えた変更を、タスクを実行するために作成された新しい Python プロセスとして認識します。
v2	「celery.worker_concurrency」	該当なし	Amazon MWAA は、このオプションの Airflow ベースインストールをオーバーライドして、自動スケーリングコンポーネントの一部としてワーカーをスケーリングします。	########## ########## ###

Version	設定オプション	デフォルト	[Description] (説明)	ユースケース
v2	「celery.worker_autoscale」	mw1.small - 5,0 mw1.medium - 10,0 mw1.large - 20,0 mw1.xlarge - 40,0 mw1.2xlarge - 80,0	ワーカー向けのタスク同時実行性。	このオプションを使用すると、ワーカーの `minimum`、`maximum` のタスク同時実行を減らすことでリソースを解放できます。ワーカーは、そのための十分なリソースがあるかどうかに関係なく、構成された `maximum` までの同時タスク受け入れます。十分なリソースがない状態でタスクをスケジュールすると、タスクはすぐに失敗します。リソースを大量に消費するタスクの場合は、タスクあたりの容量を増やすために値をデフォルトより小さくしてこの値を変更することをお勧めします。

requirements.txt での Python 依存関係の管理

このページでは、Apache Airflow 用 Amazon マネージドワークフロー環境の `requirements.txt` ファイルに Python の依存関係をインストールして管理するために推奨するベストプラクティスについて説明します。

目次

Amazon MWAA CLI ユーティリティを使用した DAG のテスト

- コマンドラインインターフェイス (CLI) ユーティリティは、Amazon Managed Workflows for Apache Airflow (Amazon MWAA) 環境をローカルに複製します。
- CLI は、Amazon MWAA のプロダクションイメージに似た Docker コンテナイメージをローカルでビルドします。これにより、Amazon MWAA にデプロイする前に、ローカルの Apache Airflow 環境を実行して DAG、カスタムプラグイン、依存関係を開発およびテストできます。
- CLI を実行するには、 の aws-mwaa-local-runner を参照してください GitHub。

PyPi.org 要件ファイル形式を使用した Python 依存関係のインストール

次のセクションでは、 PyPi.org 要件ファイル形式 に従って Python 依存関係をインストールするさまざまな方法について説明します。

オプション 1: Python Package インデックスからの Python 依存関係

次のセクションでは、`requirements.txt` ファイルの「Python Package インデックス」から Python 依存関係を指定する方法について説明します。

Apache Airflow v2

1. ローカルでテストします。`requirements.txt` ファイルを作成する前に、ライブラリを繰り返し追加してパッケージとバージョンの適切な組み合わせを見つけてください。Amazon MWAA CLI ユーティリティを実行するには、「」の「aws-mwaa-local-runner」を参照してください GitHub。
2. Apache Airflow パッケージのエクストラを確認してください。Amazon MWAA で Apache Airflow v2 にインストールされているパッケージのリストを表示するには、 GitHub ウェブサイトの「Amazon MWAA local runner`requirements.txt`」を参照してください。
3. 制約ステートメントを追加します。Apache Airflow v2 環境用の制約ファイルを `requirements.txt` ファイルの先頭に追加します。Apache Airflow の制約ファイルには、Apache Airflow のリリース時点で利用可能なプロバイダーのバージョンが指定されています。

   Apache Airflow v2.7.2 から、要件ファイルには `--constraint` ステートメントを含める必要があります。制約を指定しない場合、要件に記載されているパッケージが使用している Apache Airflow のバージョンと互換性があることを確認するため、Amazon MWAA はお客様に代わって制約を指定します。

   次の例では、*`{environment-version}`* を環境のバージョン番号に置き換え、*`{Python-version}`* を環境と互換性のある Python のバージョンに置き換えます。

   Apache Airflow 環境と互換性のある Python のバージョンについては、「Apache Airflow バージョン」を参照してください。

   ```
   --constraint "https://raw.githubusercontent.com/apache/airflow/
   constraints-{Airflow-version}/constraints-{Python-version}.txt"
   ```

制約ファイルが xyz==1.0 パッケージが環境内の他のパッケージと互換性がないと判断した場合、pip3 install は環境に互換性のないライブラリがインストールされるのを防ぐために失敗しま す。パッケージのインストールに失敗した場合、各 Apache Airflow コンポーネント (スケジューラ、ワーカー、ウェブサーバー) のエラーログを、対応するログストリームで CloudWatch Logs に表示できます。ログタイプの詳細については、「the section called "Airflow ログの表示"」を参照してください。

4. Apache Airflow パッケージ。パッケージエクストラとバージョン (==)を追加します。これにより、同じ名前で異なるバージョンのパッケージが環境にインストールされるのを防ぐことができます。

```
apache-airflow[package-extra]==2.5.1
```

5. Python ライブラリ パッケージ名とバージョン (==) を requirements.txt ファイルに追加します。これにより、PyPi.org からの将来の重大な更新が自動的に適用されないようにできます。

```
library == version
```

Example Boto3 と psycopg2-binary

この例は、デモンストレーションのみを目的としています。boto と psycopg2 のバイナリライブラリは Apache Airflow v2 のベースインストールに含まれており、requirements.txt ファイルで指定する必要はありません。

```
boto3==1.17.54
boto==2.49.0
botocore==1.20.54
psycopg2-binary==2.8.6
```

バージョンなしでパッケージが指定されている場合、Amazon MWAA は PyPi.org から最新バージョンのパッケージをインストールします。このバージョンは、お客様の requirements.txt 内の他のパッケージと競合する可能性があります。

Apache Airflow v1

1. ローカルでテストします。requirements.txt ファイルを作成する前に、ライブラリを繰り返し追加してパッケージとバージョンの適切な組み合わせを見つけてください。Amazon

MWAA CLI ユーティリティを実行するには、「」の「aws-mwaa-local-runner」を参照してください GitHub。

2. Airflow パッケージのエクストラを確認してください。https://raw.githubusercontent.com/apache/airflow/constraints-1.10.12/constraints-3.7.txt で Apache Airflow v1.10.12 で利用できるパッケージのリストが確認してください。
3. 制約ファイルを追加します。Apache Airflow v1.10.12 の制約ファイルを `requirements.txt` ファイルの先頭に追加します。制約ファイルが `xyz==1.0` パッケージが環境内の他のパッケージと互換性がないと判断した場合、`pip3 install` は環境に互換性のないライブラリがインストールされるのを防ぐために失敗しま す。

```
--constraint "https://raw.githubusercontent.com/apache/airflow/
constraints-1.10.12/constraints-3.7.txt"
```

4. アパッチ・エアフロー v1.10.12 パッケージ。「Airflow パッケージエクストラ」と Apache Airflow v1.10.12 バージョン (==) を追加します。これにより、同じ名前で異なるバージョンのパッケージが環境にインストールされるのを防ぐことができます。

```
apache-airflow[package]==1.10.12
```

Example Secure Shell (SSH)

次の例では、 `requirements.txt` ファイルは、Apache Airflow v1.10.12 用 SSH をインストールします。

```
apache-airflow[ssh]==1.10.12
```

5. Python ライブラリ パッケージ名とバージョン (==) を `requirements.txt` ファイルに追加します。これにより、PyPi.org からの将来の重大な更新が自動的に適用されないようにできます。

```
library == version
```

Example Boto3

次の例では、`requirements.txt` ファイルは、Apache Airflow v1.10.12 用の Boto3 ライブラリをインストールします。

```
boto3 == 1.17.4
```

バージョンなしでパッケージが指定されている場合、Amazon MWAA は PyPi.org から最新バージョンのパッケージをインストールします。このバージョンは、お客様の `requirements.txt` 内の他のパッケージと競合する可能性があります。

オプション 2: Python wheel (.whl)

Python wheel は、コンパイルされたアーティファクトをライブラリに同梱するために設計されたパッケージ形式です。Amazon MWAA に依存関係をインストールする方法として、ホイールパッケージにはいくつかの利点があります。

- より速いインストール — WHL ファイルは 1 つの ZIP としてコンテナにコピーされ、ローカルにインストールされます。各ファイルをダウンロードする必要はありません。
- より少ないコンフリクト — パッケージのバージョン互換性を事前に判断できます。その結果、`pip` が互換性のあるバージョンを再帰的に調べる必要がなくなります。
- 耐障害性の向上 — 外部でホストされているライブラリでは、ダウンストリームの要件が変更され、Amazon MWAA 環境上のコンテナ間でバージョン互換性がなくなる可能性があります。依存関係を外部ソースに依存しないことで、各コンテナがいつインスタンス化されたかに関係なく、上のすべてのコンテナに同じライブラリが割り当てられます。

`requirements.txt` の Python wheel アーカイブ (`.whl`) から Python 依存関係をインストールするには、次の方法をお勧めします。

方法

- Amazon S3 バケット上の plugins.zip ファイルを使用する
- URL にホストされている WHL ファイルを使用する
- DAG から WHL ファイルを作成しています。

Amazon S3 バケット上の **`plugins.zip`** ファイルを使用する

Apache Airflow スケジューラ、ワーカー、およびウェブサーバー (Apache Airflow v2.2.2 以降用) は、 の環境の AWSマネージド Fargate コンテナで起動時にカスタムプラグインを探します`/usr/local/airflow/plugins/*`。このプロセスは、Python 依存関係および Apache Airflow サービスの起動のための Amazon MWAA の `pip3 install -r requirements.txt` 前に始まります。`plugins.zip` ファイルは、環境実行中に継続的に変更されたくないファイル、またはDAGを

作成するユーザーにアクセス権を与えたくないファイルに使用できます。たとえば、Python ライブラリのホイールファイル、証明書 PEM ファイル、構成 YAML ファイルなどです。

次のセクションでは、plugins.zip ファイルにあるホイールを Amazon S3 バケットにインストールする方法について説明します。

1. 必要な WHL ファイルをダウンロードします。Amazon MWAA「ローカルランナー」または別の「Amazon Linux 2」コンテナにある既存の requirements.txt と「pip download」を使用して、必要な Python wheel ファイルを解決してダウンロードできます。

   ```
   $ pip3 download -r "$AIRFLOW_HOME/dags/requirements.txt" -d "$AIRFLOW_HOME/plugins"
   $ cd "$AIRFLOW_HOME/plugins"
   $ zip "$AIRFLOW_HOME/plugins.zip" *
   ```

2. **requirements.txt** でパスを指定します。以下に示すように、requirements.txt の先頭に「--find-links」を使用してプラグインディレクトリを指定し、pip に他のソースからインストールしないように「--no-index」を使用して指示します。

   ```
   --find-links /usr/local/airflow/plugins
   --no-index
   ```

   Example requirements.txt 内のホイール

   次の例では、Amazon S3 バケットのルートにある plugins.zip ファイルにホイールをアップロードしたことを前提としています。例:

   ```
   --find-links /usr/local/airflow/plugins
   --no-index

   numpy
   ```

   Amazon MWAA は plugins フォルダから numpy-1.20.1-cp37-cp37m-manylinux1_x86_64.whl ホイールを取得し、環境にインストールします。

URL にホストされている WHL ファイルを使用する

次のセクションでは、URL でホストされるホイールをインストールする方法について説明します。URL は、パブリックにアクセス可能であるか、Amazon MWAA 環境用に指定したカスタム Amazon MWAA 内からアクセスできる必要があります。

- URL を指定してください。`requirements.txt` 内のホイールの URL を指定します。

  Example 公開 URL でのホイールアーカイブ

  次の例では、公開サイトからホイールをダウンロードします。

  ```
  --find-links https://files.pythonhosted.org/packages/
  --no-index
  ```

  Amazon MWAA は、指定した URL からホイールを取得し、お使いの環境にインストールします。

  > **Note**
  >
  > Amazon MWAA v2.2.2 以降では、要件をインストールするプライベートウェブサーバーから URL にアクセスすることはできません。

DAG から WHL ファイルを作成しています。

Apache Airflow v2.2.2 以降を使用するプライベートウェブサーバーがあり、環境が外部リポジトリにアクセスできないために要件をインストールできない場合は、次の DAG を使用して既存の Amazon MWAA 要件を取得し、Amazon S3 にパッケージ化できます。

```
from airflow import DAG
from airflow.operators.bash_operator import BashOperator
from airflow.utils.dates import days_ago

S3_BUCKET = 'my-s3-bucket'
S3_KEY = 'backup/plugins_whl.zip'

with DAG(dag_id="create_whl_file", schedule_interval=None, catchup=False,
 start_date=days_ago(1)) as dag:
    cli_command = BashOperator(
        task_id="bash_command",
        bash_command=f"mkdir /tmp/whls;pip3 download -r /usr/local/airflow/
requirements/requirements.txt -d /tmp/whls;zip -j /tmp/plugins.zip /tmp/whls/*;aws s3
 cp /tmp/plugins.zip s3://{S3_BUCKET}/{S3_KEY}"
    )
```

DAG を実行したら、この新しいファイルを Amazon MWAA plugins.zip として使用します。オプションで、他のプラグインと一緒にパッケージ化することもできます。次に、 を追加--no-indexせずにrequirements.txt、 の前に --find-links /usr/local/airflow/pluginsとを更新します--constraint。

この方法では、同じライブラリをオフラインで使用できます。

オプション 3: プライベート PyPi/PEP-503 準拠リポジトリでホストされる Python 依存関係

次のセクションでは、認証付きのプライベート URL でホストされている Apache Airflow エクストラをインストールする方法について説明します。

1. ユーザー名とパスワードを「Apache Airflow 構成オプション」として追加します。例:
   - foo.user : *YOUR_USER_NAME*
   - foo.pass : *YOUR_PASSWORD*
2. requirements.txt ファイルを作成します。次の例のプレースホルダーは、プライベート URL と「Apache Airflow 構成オプション」として追加したユーザー名とパスワードに置き換えてください。例:

```
--index-url https://${AIRFLOW__FOO__USER}:${AIRFLOW__FOO__PASS}@my.privatepypi.com
```

3. その他のライブラリを requirements.txt ファイルに追加します。例:

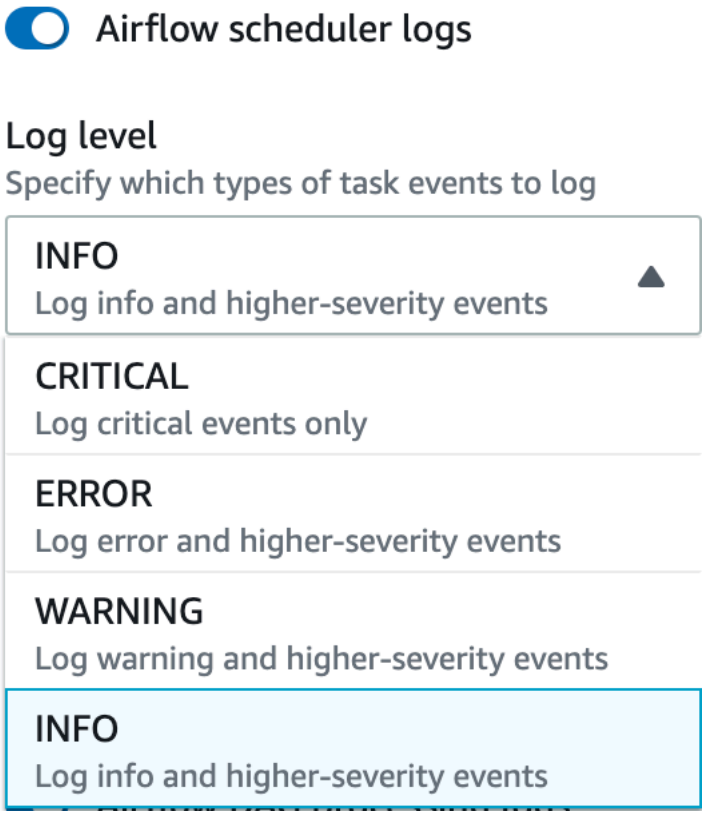
```
--index-url https://${AIRFLOW__FOO__USER}:${AIRFLOW__FOO__PASS}@my.privatepypi.com
my-private-package==1.2.3
```

Amazon MWAA コンソールでログを有効にします。

Amazon MWAA 環境の実行ロールには、ログを CloudWatch Logs に送信するためのアクセス許可が必要です。実行ロールのアクセス権限を更新するには、「Amazon MWAA 実行ロール」を参照してください。

Apache Airflow ログは INFO、WARNING、ERROR または CRITICAL レベルで有効にできます。ログレベルを選択すると、Amazon MWAA はそのレベルとそれ以上の重要度レベルのすべてのログを送信します。例えば、 INFOレベルでログを有効にすると、Amazon MWAA はINFOログと WARNING、ERROR、 CRITICALログレベルを CloudWatch Logs に送信しま

す。requirements.txt で受信したログをスケジューラーに表示できるように、INFO レベルで Apache Airflow ログを有効にすることをお勧めします。



Logs コンソールでの CloudWatch ログの表示

ワークフローのスケジュール設定と dags フォルダーの解析を行うスケジューラーの Apache Airflow ログを表示できます。次の手順では、Amazon MWAA コンソールでスケジューラのロググループを開き、Logs コンソールで Apache Airflow CloudWatch ログを表示する方法について説明します。

requirements.txt のログを表示するには

1. Amazon MWAA コンソールで「環境ページ」を開きます。
2. 環境を選択します。
3. モニタリングペインで Airflow スケジューラーロググループを選択します。
4. [ログストリーム] requirements_install_ipのログを選択します。
5. /usr/local/airflow/.local/bin で環境にインストールされたパッケージのリストが表示されるはずです。例:

```
Collecting appdirs==1.4.4 (from -r /usr/local/airflow/.local/bin (line 1))
```

```
Downloading https://files.pythonhosted.org/
packages/3b/00/2344469e2084fb28kjdsfiuyweb47389789vxbmnbjhsdgf5463acd6cf5e3db69324/
appdirs-1.4.4-py2.py3-none-any.whl
Collecting astroid==2.4.2 (from -r /usr/local/airflow/.local/bin (line 2))
```

6. パッケージのリストを確認し、インストール中にエラーが発生したパッケージがないか確認してください。何か問題が発生した場合、以下のようなエラーが表示されることがあります。

```
2021-03-05T14:34:42.731-07:00
No matching distribution found for LibraryName==1.0.0 (from -r /usr/local/
airflow/.local/bin (line 4))
No matching distribution found for LibraryName==1.0.0 (from -r /usr/local/
airflow/.local/bin (line 4))
```

Apache Airflow UI でエラーを表示する

また、Apache Airflow UI をチェックして、エラーが別の問題に関連しているかどうかを確認することもできます。Amazon MWAA の Apache Airflow で発生する可能性のある最も一般的なエラーは次のとおりです。

```
Broken DAG: No module named x
```

Apache Airflow UI にこのエラーが表示される場合は、`requirements.txt` のファイルに必要な依存関係が欠けている可能性があります。

Apache Airflow へのログイン

Apache Airflow UI を表示するには、 AWS Identity and Access Management (IAM) AWS のアカウントのApache Airflow UI アクセスポリシー: AmazonMWAAWebServer アクセスアクセス許可が必要です。

Apache Airflow UI にアクセスするには

1. Amazon MWAA コンソールで「環境ページ」を開きます。
2. 環境を選択します。
3. [Airflow UI を開く] を選択します。

requirements.txt シナリオ例

requirements.txt では異なるフォーマットを組み合わせることができます。次の例では、さまざまな方法を組み合わせてエクストラをインストールしています。

Example PyPi.org とパブリック URL の追加

カスタム PEP 503 準拠のリポジトリ URL などのパブリック URL 上のパッケージに加えて、PyPi.org からパッケージを指定する場合は、 --index-urlオプションを使用する必要がありますURLs 。

```
aws-batch == 0.6
phoenix-letter >= 0.3

--index-url http://dist.repoze.org/zope2/2.10/simple
    zopelib
```

Apache Airflow 用 Amazon マネージドワークフローのモニタリングとメトリクス

モニタリングは、Amazon Managed Workflows for Apache Airflow と AWS ソリューションの信頼性、可用性、パフォーマンスを維持する上で重要な部分です。マルチポイント障害が発生した場合は、その障害をより簡単にデバッグできるように、 AWS ソリューションのすべての部分からモニタリングデータを収集することをお勧めします。このトピックでは AWS 、Amazon MWAA 環境をモニタリングし、潜在的なイベントに対応するための リソースについて説明します。

> **Note**
>
> Apache Airflow のメトリクスと ログ記録には、標準の Amazon CloudWatch 料金が適用されます。

Apache Airflow のモニタリングの詳細については、Apache Airflow ドキュメンテーションウェブサイトの「ログとモニタリング」を参照してください。

セクション

Amazon MWAA でのモニタリングの概要

このページでは、Amazon Managed Workflows for Apache Airflow 環境のモニタリングに使用される AWS サービスについて説明します。

目次

Amazon CloudWatch の概要

CloudWatch は、 AWS サービスによって発行されたメトリクスとディメンションに基づいて統計を取得できる サービスのメトリクスリポジトリです。これらのメトリクスを使用してアラームを設定し、統計を計算し、Amazon CloudWatch コンソールで環境の状態を評価するのに役立つダッシュボードにデータを表示できます。

Apache Airflow は、Amazon Managed Workflows for Apache Airflow StatsD メトリクスを Amazon に送信するように既に設定されています CloudWatch。

詳細については、「Amazon とは CloudWatch」を参照してください。

AWS CloudTrail 概要

CloudTrail は、Amazon MWAA 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監査 AWS サービスです。によって収集された情報を使用して CloudTrail、Amazon MWAA に対して行われたリクエスト、リクエスト元の IP アドレス、リクエスト者、リクエスト日時、および監査ログで利用できる追加の詳細を確認できます。

詳細については、「 AWS CloudTrailとは」を参照してください。

での監査ログの表示 AWS CloudTrail

AWS CloudTrail は、IAM エンティティまたは Amazon Managed Workflows for Apache Airflow などの AWS サービスによって実行されたアクティビティ CloudTrail を CloudTrail イベントとして記録するときに、 AWS アカウントで有効になります。 CloudTrail コンソールで過去 90 日間のイベント履歴を表示、検索、ダウンロードできます。 は、Amazon MWAA コンソールのすべてのイベントと、Amazon MWAA API へのすべての呼び出しを CloudTrail キャプチャします。 APIs `GetEnvironment` などの読み取り専用アクションや `PublishMetrics` アクションはキャプチャされません。このページでは、 CloudTrail を使用して Amazon MWAA のイベントをモニタリングする方法について説明します。

目次

での証跡の作成 CloudTrail

Amazon MWAA のイベントなど、 AWS アカウント内のイベントの継続的な記録を表示するには、証跡を作成する必要があります。証跡により、 はログファイル CloudTrail を Amazon S3 バケットに配信できます。証跡を作成しない場合でも、 CloudTrail コンソールで利用可能なイベント履歴を表示できます。例えば、 で収集された情報を使用して CloudTrail、Amazon MWAA に対するリクエスト、リクエスト元の IP アドレス、リクエスト者、リクエスト日時などの詳細を確認できます。詳細については、AWS 「アカウントの証跡の作成」を参照してください。

イベント履歴を使用した CloudTrail イベントの表示

イベント履歴を表示することで、過去 90 日間の運用インシデントとセキュリティインシデントを CloudTrail コンソールでトラブルシューティングできます。例えば、アカウント内のリソース (IAM ユーザーやその他の AWS リソースなど) の作成、変更、削除に関連するイベント AWS をリージョンごとに表示できます。詳細については、「イベント履歴を使用した CloudTrail イベントの表示」を参照してください。

1. CloudTrail コンソールを開きます。
2. [イベント履歴] を選択します。
3. 表示したいイベントを選択し、[イベントの詳細を比較] を選択します。

`CreateEnvironment` のトレイルの例

「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 ログファイルには、1 つ以上のログエントリが含まれます。イベントは任意のソースからの単一のリクエストを表し、アクションの日時やリクエストパラメータなど、リクエストされたアクションに関する情報が含まれます。 CloudTrail ログファイルはパブリック API コールの順序付けられたスタックトレースではなく、特定の順序で表示されません。次の例では、アクセス許可がないために拒否された `CreateEnvironment` アクションのログエントリを示します。`AirflowConfigurationOptions` 内の値は、プライバシー保護のために編集されています。

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "00123456ABC7DEF8HIJK",
```

```
        "arn": "arn:aws:sts::012345678901:assumed-role/root/myuser",
        "accountId": "012345678901",
        "accessKeyId": "",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "00123456ABC7DEF8HIJK",
                "arn": "arn:aws:iam::012345678901:role/user",
                "accountId": "012345678901",
                "userName": "user"
            },
            "webIdFederationData": {},
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2020-10-07T15:51:52Z"
            }
        }
    },
    "eventTime": "2020-10-07T15:52:58Z",
    "eventSource": "airflow.amazonaws.com",
    "eventName": "CreateEnvironment",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "205.251.233.178",
    "userAgent": "PostmanRuntime/7.26.5",
    "errorCode": "AccessDenied",
    "requestParameters": {
        "SourceBucketArn": "arn:aws:s3:::my-bucket",
        "ExecutionRoleArn": "arn:aws:iam::012345678901:role/AirflowTaskRole",
        "AirflowConfigurationOptions": "***",
        "DagS3Path": "sample_dag.py",
        "NetworkConfiguration": {
            "SecurityGroupIds": [
                "sg-01234567890123456"
            ],
            "SubnetIds": [
                "subnet-01234567890123456",
                "subnet-65432112345665431"
            ]
        },
        "Name": "test-cloudtrail"
    },
    "responseElements": {
        "message": "Access denied."
    },
```

```
    "requestID": "RequestID",
    "eventID": "EventID",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "recipientAccountId": "012345678901"
}
```

次のステップ

- サポートされている AWS サービスと統合 の CloudTrail ログで収集されたイベントデータ用に他の サービスを設定する方法について説明します。 CloudTrail
- が新しいログファイルを Amazon S3 バケットに CloudTrail 発行するときに通知を受け取る方法については、「 の Amazon Amazon SNS 通知の設定」を参照してください CloudTrail。

Amazon での Airflow ログの表示 CloudWatch

Amazon MWAA は Apache Airflow ログを Amazon に送信できます CloudWatch。追加のサードパーティツールを必要とせずに、Apache Airflow のタスクの遅延やワークフローエラーを簡単に特定するために、単一の場所から複数の環境のログを表示できます。Apache Airflow DAG 処理、タスク、ウェブサーバー、ワーカーログを で表示するには、Apache Airflow コンソールの Amazon Managed Workflows で Apache Airflow ログを有効にする必要があります CloudWatch。

目次

料金

- Standard CloudWatch Logs の料金が適用されます。詳細については、「 のCloudWatch 料金」を参照してください。

開始する前に

- でログを表示できるロールが必要です CloudWatch。詳細については、「Amazon MWAA 環境へのアクセス」を参照してください。

ログタイプ

Amazon MWAA は、有効にした Airflow ログ記録オプションごとにロググループを作成し、環境に関連付けられた CloudWatch ロググループにログをプッシュします。ロググループの名前は以下の形式に従います: `YourEnvironmentName-`**`LogType`** たとえば、お使いの環境に `Airflow-v202-Public` という名前が付けられている場合、Apache Airflow タスクログは `Airflow-v202-Public-`**`Task`** に送信されます。

ログタイプ	説明
`YourEnvironmentName-` **`DAGProcessing`**	DAG プロセッサマネージャー (DAG ファイルを処理するスケジューラーの一部) のログ。
`YourEnvironmentName-` **`Scheduler`**	Airflow スケジューラーが生成するログ。
`YourEnvironmentName-` **`Task`**	DAG が生成するタスクログ。
`YourEnvironmentName-` **`WebServer`**	Airflow ウェブインターフェイスが生成するログ。
`YourEnvironmentName-` **`Worker`**	ワークフローと DAG 実行の一部として生成されたログ。

Apache Airflow ログを有効にする

Apache Airflow ログは `INFO`、`WARNING`、`ERROR` または `CRITICAL` レベルで有効にできます。ログレベルを選択すると、Amazon MWAA はそのレベルとそれ以上の重要度レベルのすべてのログを送信します。例えば、 `INFO`レベルでログを有効にすると、Amazon MWAA は`INFO`ログと `WARNING`、`ERROR`、 `CRITICAL`ログレベルを CloudWatch Logs に送信します。

1. Amazon MWAA コンソールで「環境ページ」を開きます。
2. 環境を選択します。

3. [編集] を選択します。
4. [次へ] をクリックします。
5. 以下のロギングオプションのうち 1 つ以上を選択します。
    a. [モニタリング] ペインで [Airflow スケジューラーロググループ] を選択します。
    b. [監視] ペインで [Airflow ウェブサーバーのロググループ] を選択します。
    c. [監視] ペインで [Airflow ワーカーロググループ] を選択します。
    d. [監視] ペインで [Airflow DAG 処理ロググループ] を選択します。
    e. [監視] ペインで [Airflow タスクロググループ] を選択します。
    f. [ログレベル] でログレベルを選択します。
6. [次へ] をクリックします。
7. [保存] を選択します。

Apache Airflow ログを表示する

次のセクションでは、コンソールで Apache Airflow ログを表示する方法について説明します CloudWatch 。

1. Amazon MWAA コンソールで「環境ページ」を開きます。
2. 環境を選択します。
3. [モニタリング] ペインでロググループを選択します。
4. ログストリームを選択します。

スケジューラーログの例

ワークフローのスケジュール設定と dags フォルダーの解析を行うスケジューラーの Apache Airflow ログを表示できます。次の手順では、Amazon MWAA コンソールでスケジューラのロググループを開き、Logs コンソールで Apache Airflow CloudWatch ログを表示する方法について説明します。

`requirements.txt` のログを表示するには

1. Amazon MWAA コンソールで「環境ページ」を開きます。
2. 環境を選択します。
3. モニタリングペインで Airflow スケジューラーロググループを選択します。

4. [ログストリーム] requirements_install_ipのログを選択します。
5. /usr/local/airflow/.local/bin で環境にインストールされたパッケージのリストが表示されるはずです。例:

```
Collecting appdirs==1.4.4 (from -r /usr/local/airflow/.local/bin (line 1))
Downloading https://files.pythonhosted.org/
packages/3b/00/2344469e2084fb28kjdsfiuyweb47389789vxbmnbjhsdgf5463acd6cf5e3db69324/
appdirs-1.4.4-py2.py3-none-any.whl
Collecting astroid==2.4.2 (from -r /usr/local/airflow/.local/bin (line 2))
```

6. パッケージのリストを確認し、インストール中にエラーが発生したパッケージがないか確認してください。何か問題が発生した場合、以下のようなエラーが表示されることがあります。

```
2021-03-05T14:34:42.731-07:00
No matching distribution found for LibraryName==1.0.0 (from -r /usr/local/
airflow/.local/bin (line 4))
No matching distribution found for LibraryName==1.0.0 (from -r /usr/local/
airflow/.local/bin (line 4))
```

次のステップ

- CloudWatch アラームの設定方法については、「Amazon アラームの使用 CloudWatch 」を参照してください。
- ダッシュボードの作成方法については、 CloudWatch 「 ダッシュボードの使用 CloudWatch 」を参照してください。

Amazon MWAA のモニタリングダッシュボードとアラーム

Amazon でカスタムダッシュボードを作成し CloudWatch 、特定のメトリクスのアラームを追加して、Amazon Managed Workflows for Apache Airflow 環境のヘルスステータスをモニタリングできます。ダッシュボードにある場合、ALARM 状態になると、アラームは赤に変わるため、Amazon MWAA の環境の健全を事前にモニタリングしやすくなります。

Apache Airflow は、DAG プロセスの数、DAG バッグサイズ、現在実行中のタスク、タスクの失敗、成功など、さまざまなプロセスのメトリクスを公開します。環境を作成すると、Airflow は Amazon MWAA 環境のメトリクスを に自動的に送信するように設定されます CloudWatch。このページで

は、Amazon MWAA 環境の で Airflow メトリクス CloudWatch のヘルスステータスダッシュボードを作成する方法について説明します。

目次

メトリクス

お使いの Apache Airflow バージョンで使用できるすべてのメトリクスについて、カスタムダッシュボードとアラームを作成できます。各メトリクスはApache Airflow キーパフォーマンスインジケーター (KPI) に対応しています。メトリクスのリストを表示するには、以下を参照してください。

- の Apache Airflow v2 環境メトリクス CloudWatch

アラーム状態の概要

メトリクスアラームには次の状態があります。

- OK – メトリクスや式は、定義されているしきい値の範囲内です。
- ALARM – メトリクスまたは式が、定義されているしきい値を超えています。
- INSUFFICIENT_DATA – アラームが開始直後であるか、メトリクスが利用できないか、メトリクス用のデータが不足しているため、アラームの状態を判定できません。

カスタムダッシュボードとアラームの例

Amazon MWAA 環境の選択したメトリクスのグラフを表示するカスタムモニタリングダッシュボードを構築できます。

これらのメトリクスについて

以下のリストでは、このセクションのチュートリアルとテンプレート定義によってカスタムダッシュボードに作成された各メトリクスについて説明します。

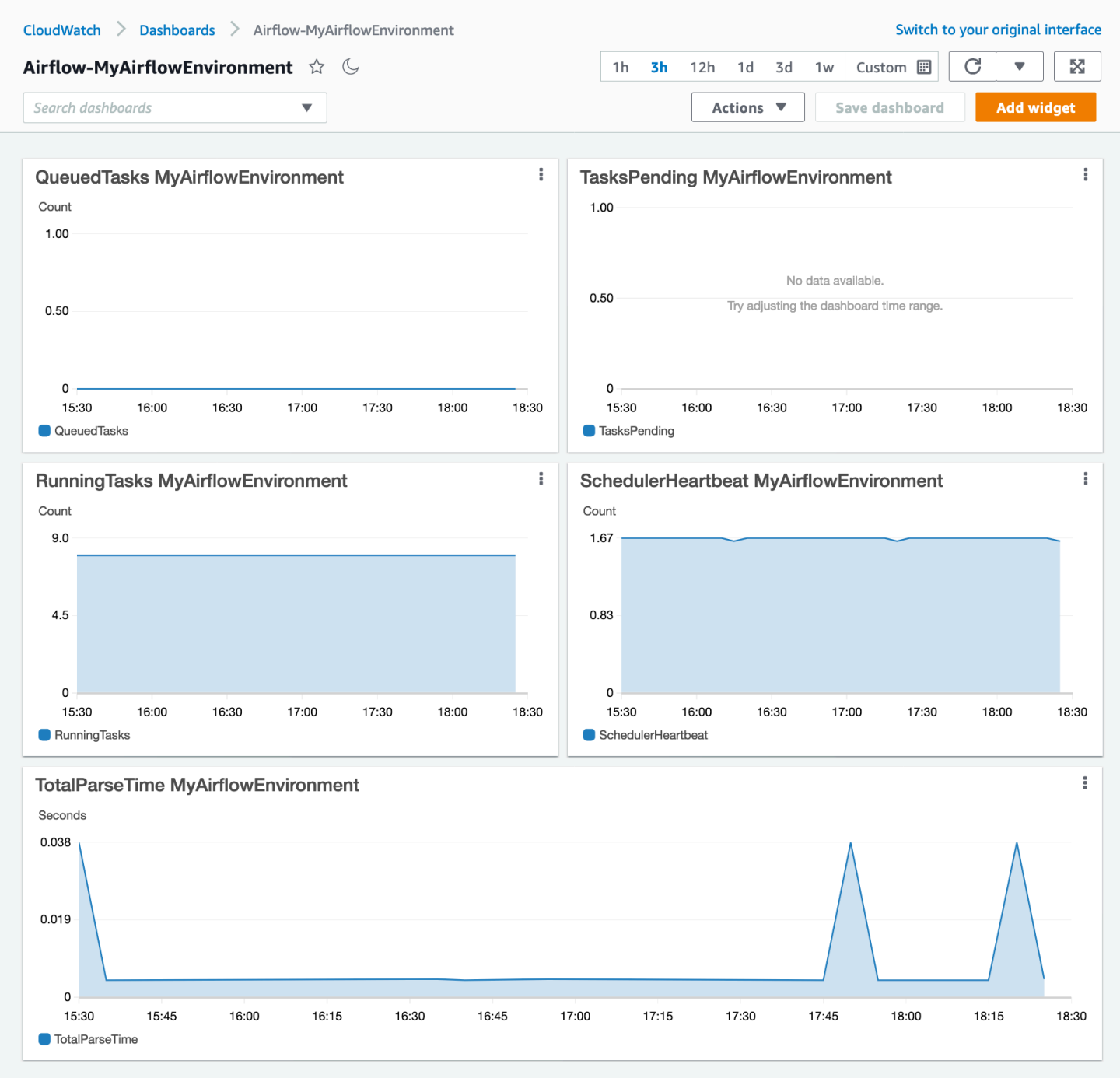
- QueuedTasks - キューに入っている状態のタスクの数。`executor.queued_tasks` Apache Airflow メトリクスに対応します。
- TasksPending - エグゼキューターで保留中のタスクの数。`scheduler.tasks.pending` Apache Airflow メトリクスに対応します。

> **Note**
>
> Apache Airflow v2.2 以上には適用されません。

- RunningTasks - エグゼキューターで実行されているタスクの数。`executor.running_tasks` Apache Airflow メトリクスに対応します。
- SchedulerHeartbeat - スケジューラジョブで Apache Airflow が実行するチェックインの数。`scheduler_heartbeat` Apache Airflow メトリクスに対応します。
- TotalParse時間 - すべての DAG ファイルを 1 回スキャンしてインポートするのにかかった秒数。`dag_processing.total_parse_time` Apache Airflow メトリクスに対応します。

ダッシュボードについて

以下の画像は、このセクションのチュートリアルとテンプレート定義によって作成された監視ダッシュボードを示しています。



AWS チュートリアルの使用

次の AWS チュートリアルを使用して、現在デプロイされている Amazon MWAA 環境のヘルスステータスダッシュボードを自動的に作成できます。また、すべての Amazon MWAA 環境で異常なワーカーやスケジューラのハートビート障害に対する CloudWatch アラームも作成します。

- CloudWatch Amazon MWAA のダッシュボード自動化

の使用 AWS CloudFormation

このセクションの AWS CloudFormation テンプレート定義を使用して、 でモニタリングダッシュボードを作成し CloudWatch、メトリクスが特定のしきい値を超えたときに通知を受け取るように CloudWatch コンソールにアラームを追加できます。このテンプレート定義を使用してスタックを作成するには、「コンソールでのスタックの作成 AWS CloudFormation」を参照してください。ダッシュボードにアラームを追加するには、「アラームの使用」を参照してください。

```
AWSTemplateFormatVersion: "2010-09-09"
Description: Creates MWAA Cloudwatch Dashboard
Parameters:
  DashboardName:
    Description: Enter the name of the CloudWatch Dashboard
    Type: String
  EnvironmentName:
    Description: Enter the name of the MWAA Environment
    Type: String
Resources:
  BasicDashboard:
    Type: AWS::CloudWatch::Dashboard
    Properties:
      DashboardName: !Ref DashboardName
      DashboardBody:
        Fn::Sub: '{
              "widgets": [
                  {
                      "type": "metric",
                      "x": 0,
                      "y": 0,
                      "width": 12,
                      "height": 6,
                      "properties": {
                          "view": "timeSeries",
                          "stacked": true,
                          "metrics": [
                              [
                                  "AmazonMWAA",
                                  "QueuedTasks",
                                  "Function",
                                  "Executor",
                                  "Environment",
                                  "${EnvironmentName}"
                              ]
```

```
                ],
                "region": "${AWS::Region}",
                "title": "QueuedTasks ${EnvironmentName}",
                "period": 300
            }
        },
        {
            "type": "metric",
            "x": 0,
            "y": 6,
            "width": 12,
            "height": 6,
            "properties": {
                "view": "timeSeries",
                "stacked": true,
                "metrics": [
                    [
                        "AmazonMWAA",
                        "RunningTasks",
                        "Function",
                        "Executor",
                        "Environment",
                        "${EnvironmentName}"
                    ]
                ],
                "region": "${AWS::Region}",
                "title": "RunningTasks ${EnvironmentName}",
                "period": 300
            }
        },
        {
            "type": "metric",
            "x": 12,
            "y": 6,
            "width": 12,
            "height": 6,
            "properties": {
                "view": "timeSeries",
                "stacked": true,
                "metrics": [
                    [
                        "AmazonMWAA",
                        "SchedulerHeartbeat",
                        "Function",
```

```
                        "Scheduler",
                        "Environment",
                        "${EnvironmentName}"
                    ]
                ],
                "region": "${AWS::Region}",
                "title": "SchedulerHeartbeat ${EnvironmentName}",
                "period": 300
            }
        },
        {
            "type": "metric",
            "x": 12,
            "y": 0,
            "width": 12,
            "height": 6,
            "properties": {
                "view": "timeSeries",
                "stacked": true,
                "metrics": [
                    [
                        "AmazonMWAA",
                        "TasksPending",
                        "Function",
                        "Scheduler",
                        "Environment",
                        "${EnvironmentName}"
                    ]
                ],
                "region": "${AWS::Region}",
                "title": "TasksPending ${EnvironmentName}",
                "period": 300
            }
        },
        {
            "type": "metric",
            "x": 0,
            "y": 12,
            "width": 24,
            "height": 6,
            "properties": {
                "view": "timeSeries",
                "stacked": true,
                "region": "${AWS::Region}",
```

```
                    "metrics": [
                        [
                            "AmazonMWAA",
                            "TotalParseTime",
                            "Function",
                            "DAG Processing",
                            "Environment",
                            "${EnvironmentName}"
                        ]
                    ],
                    "title": "TotalParseTime  ${EnvironmentName}",
                    "period": 300
                }
            }
        ]
    }'
```

メトリクスとダッシュボードの削除

Amazon MWAA 環境を削除すると、対応するダッシュボードも削除されます。 CloudWatch メトリクスは 15 (15) か月間保存され、削除できません。 CloudWatch コンソールでは、メトリクスの検索が最後に取り込まれた 2 週間後に制限され、Amazon MWAA 環境で最新のインスタンスが表示されるようになります。詳細については、「Amazon CloudWatch FAQs」を参照してください。

次のステップ

- 環境の Amazon Aurora PostgreSQL メタデータデータベースをクエリし、カスタムメトリクスを CloudWatch の に発行する DAG を作成する方法について説明しますDAG を使用して CloudWatch にカスタムメトリクスを書き込む。

の Apache Airflow v2 環境メトリクス CloudWatch

Apache Airflow v2 は、Amazon Managed Workflows for Apache Airflow 環境の StatsD メトリクスを収集して Amazon に送信するように既に設定されています CloudWatch。Apache Airflow が送信するメトリクスの全リストは、「Apache Airflow リファレンスガイド」のメトリクスページにあります。このページでは、 で利用可能な Apache Airflow メトリクスと CloudWatch、 CloudWatch コンソールでメトリクスにアクセスする方法について説明します。

目次

用語

名前空間

名前空間は、 AWS サービスの CloudWatch メトリクスのコンテナです。Amazon MWAA の場合、名前空間は AmazonMWAA です。

CloudWatch メトリクス

CloudWatch メトリクスは、 に固有の時系列のデータポイントのセットを表します CloudWatch。

Apache Airflow メトリクス

Apache Airflow 固有の「メトリクス」。

ディメンション

ディメンションは、メトリクスのアイデンティティの一部である名前と値のペアです。

単位

1 つの統計には、測定単位があります。Amazon MWAA の単位には、カウント、秒、ミリ秒が含まれます。Amazon MWAA の場合、単位は元の Airflow メトリクスの単位に基づいて設定されます。

ディメンション

このセクションでは、 CloudWatch の Apache Airflow メトリクスのディメンショングループについて説明します CloudWatch。

ディメンション	説明
DAG	特定の Apache Airflow DAG 名を示します。
DAG ファイル名	特定の Apache Airflow DAG ファイル名を示します。
機能	このディメンションは、でのメトリクスのグループ化を改善するために使用されます CloudWatch。
ジョブ	スケジューラーによって実行される Apache Airflow ジョブを示します。常にジョブという価値がある。
演算子	特定の Apache Airflow オペレーターを示します。
プール	特定の Apache Airflow ワーカープールを示します。
タスク	特定の Apache Airflow タスクを示します。
HostName	実行中の特定の Apache Airflow プロセスのホスト名を示します。

CloudWatch コンソールでのメトリクスへのアクセス

このセクションでは、特定の DAG CloudWatch の でパフォーマンスメトリクスにアクセスする方法について説明します。

ディメンションのパフォーマンスメトリクスを表示する方法

1. CloudWatch コンソールでメトリクスページを開きます。
2. AWS リージョンセレクタを使用して、リージョンを選択します。
3. [AmazonMWAA] の名前空間を選択します。
4. [すべてのメトリクス] タブでディメンションを選択します。たとえば、DAG、環境などです。
5. ディメンションの CloudWatch メトリクスを選択します。例えば、TaskInstance成功またはTaskInstance期間 などです。[すべての検索結果をグラフ化] を選択します。
6. 「グラフ化されたメトリクス」 タブを選択すると、DAG、環境、タスクなどの Apache Airflow メトリクスのパフォーマンス統計が表示されます。

で利用可能な Apache Airflow メトリクス CloudWatch

このセクションでは、 に送信される Apache Airflow メトリクスとディメンションについて説明します CloudWatch。

Apache Airflow カウンター

このセクションの Apache Airflow メトリクスには、「Apache Airflow カウンター」に関するデータが含まれています。

CloudWatch メトリクス	Apache Airflow メトリクス	単位	ディメンション	
SLAMissed **Note** Apache Airflow v2.4.3 以上で利用可能です。	sla_missed	カウント	関数、スケジュール	

CloudWatch メトリクス	Apache Airflow メトリクス	単位	ディメンション	
FailedSLACallback Note Apache Airflow v2.4.3 以上で利用可能です。	sla_callback_notification_failure	カウント	関数、スケジュール	
更新 Note Apache Airflow v2.6.3 以上で利用可能です。	dataset.updates	カウント	関数、スケジュール	
Orphaned Note Apache Airflow v2.6.3 以上で利用可能です。	dataset.orphaned	カウント	関数、スケジュール	
FailedCeleryTaskExecution Note Apache Airflow v2.4.3 以上で利用可能です。	celery.execute_command.failure	カウント	関数、Celery	

CloudWatch メトリクス	Apache Airflow メトリクス	単位	ディメンション	
FilePathQueueUpdateカウント Note Apache Airflow v2.6.3 以上で利用可能です。	dag_processing.file_path_queue_update_count	カウント	関数、スケジュール	
CriticalSectionビジー	scheduler.critical_section_busy	カウント	関数、スケジュール	
DagBagサイズ	dagbag_size	カウント	関数、DAG処理	
DagCallback例外	dag.callback_exceptions	カウント	DAG、すべて	
FailedSLAEmailAttempts	sla_email_notification_failure	カウント	関数、スケジュール	
TaskInstance終了	ti.finish.{dag_id}.{task_id}.{state}	カウント	DAG, {dag_id} Task, {task_id} State, {state}	
JobEnd	{job_name}_end	カウント	Job, {job_name}	

CloudWatch メトリクス	Apache Airflow メトリクス	単位	ディメンション
JobHeartbeat失敗	{job_name}_heartbeat_failure	カウント	Job, {job_name}
JobStart	{job_name}_start	カウント	Job, {job_name}
ManagerStalls	dag_processing.manager_stalls	カウント	関数、DAG処理
OperatorFailures	operator_failures_{operator_name}	カウント	Operator, {operator_name}
OperatorSuccesses	operator_successes_{operator_name}	カウント	Operator, {operator_name}
OtherCallbackカウント Note Apache Airflow v2.6.3 以上で利用可能です。	dag_processing.other_callback_count	カウント	関数、スケジュール
プロセス	dag_processing.processes	カウント	関数、DAG処理
SchedulerHeartbeat	scheduler_heartbeat	カウント	関数、スケジュール

CloudWatch メトリクス	Apache Airflow メトリクス	単位	ディメンション	
StartedTaskインスタンス	ti.start.{dag_id}.{task_id}	カウント	DAG、すべて タスク、すべて	
SlaCallbackカウント	dag_processing.sla_callback_count Note Apache Airflow v2.6.3 以上で利用可能です。	カウント	関数、スケジュール	
TasksKilled外部	scheduler.tasks.killed_externally	カウント	関数、スケジュール	
TaskTimeoutエラー	celery.task_timeout_error	カウント	関数、Celery	

CloudWatch メトリクス	Apache Airflow メトリクス	単位	ディメンション
TaskInstanceCreatedUsing演算子	task_instance_created-{operator_name}	カウント	Operator, {operator_name}
TaskInstancePreviouslySucceeded	previously_succeeded	カウント	DAG、すべて タスク、すべて
TaskInstance失敗	ti_failure	カウント	DAG、すべて タスク、すべて
TaskInstance成功	ti_success	カウント	DAG、すべて タスク、すべて
TaskRemovedFromDAG	task_removed_from_dag.{dag_id}	カウント	DAG, {dag_id}
TaskRestoredToDAG	task_restored_to_dag.{dag_id}	カウント	DAG, {dag_id}
TriggersSucceeded **Note** Apache Airflow v2.7.2 以上で利用可能です。	triggers.succeeded	カウント	関数、トリガー

CloudWatch メトリクス	Apache Airflow メトリクス	単位	ディメンション	
TriggersFailed Note Apache Airflow v2.7.2 以上で利用可能です。	triggers.failed	カウント	関数、トリガー	
TriggersBlockedMainThread Note Apache Airflow v2.7.2 以上で利用可能です。	triggers.blocked_main_thread	カウント	関数、トリガー	
TriggerHeartbeat Note Apache Airflow v2.8.1 以降で使用できます。	triggerer_heartbeat	カウント	関数、トリガー	

CloudWatch メトリクス	Apache Airflow メトリクス	単位	ディメンション	
TaskInstanceCreatedUsing演算子	airflow.task_instance_created_{operator_name} Note Apache Airflow v2.7.2 以上で利用可能です。	カウント	演算子、{operator_name}	
ZombiesKilled	zombies_killed	カウント	DAG、すべて タスク、すべて	

Apache Airflow Gauges

このセクションの Apache Airflow メトリクスには、「Apache Airflow ゲージ」に関するデータが含まれています。

CloudWatch メトリクス	Apache Airflow メトリクス	単位	ディメンション	
DAG FileRefresh エラー	dag_file_refresh_error	カウント	関数、DAG 処理	

CloudWatch メトリクス	Apache Airflow メトリクス	単位	ディメンション	
ImportErrors	dag_processing.import_errors	カウント	関数、DAG 処理	
Exception Failures	smart_sensor_operator.exception_failures	カウント	関数、スマートセンサーオペレーター	
ExecutedTasks	smart_sensor_operator.executed_tasks	カウント	関数、スマートセンサーオペレーター	
InfraFailures	smart_sensor_operator.infra_failures	カウント	関数、スマートセンサーオペレーター	
LoadedTasks	smart_sensor_operator.loaded_tasks	カウント	関数、スマートセンサーオペレーター	
TotalParse時間	dag_processing.total_parse_time	[秒]	関数、DAG 処理	

CloudWatch メトリクス	Apache Airflow メトリクス	単位	ディメンション	
TriggeredDag実行 Note Apache Airflow v2.6.3 以上で利用可能です。	dataset.triggered_dagruns	カウント	関数、スケジュール	
TriggersRunning Note Apache Airflow v2.7.2 以上で利用可能です。	triggers.running.*{hostname}*	カウント	関数、トリガー HostName、*{hostname}*	

CloudWatch メトリクス	Apache Airflow メトリクス	単位	ディメンション	
PoolDeferredスロット Note Apache Airflow v2.7.2 以上で利用可能です。	pool.deferred_slots.{pool_name}	カウント	Pool, {pool_name}	
DAGFileProcessingLastRunSecondsAgo	dag_processing.last_run.seconds_ago.{dag_filename}	[秒]	DAG Filename, {dag_filename}	
OpenSlots	executor.open_slots	カウント	関数、エグゼキューター	
OrphanedTasks採用済み	scheduler.orphaned_tasks.adopted	カウント	関数、スケジュール	
OrphanedTasksクリア済み	scheduler.orphaned_tasks.cleared	カウント	関数、スケジュール	
PokedExceptions	smart_sensor_operator.poked_exception	カウント	関数、スマートセンサーオペレーター	

CloudWatch メトリクス	Apache Airflow メトリクス	単位	ディメンション	
PokedSuccess	smart_sensor_operator.poked_success	カウント	関数、スマートセンサーオペレーター	
PokedTasks	smart_sensor_operator.poked_tasks	カウント	関数、スマートセンサーオペレーター	
PoolFailures	pool.open_slots.{pool_name}	カウント	Pool, {pool_name}	
PoolStarvingタスク	pool.starving_tasks.{pool_name}	カウント	Pool, {pool_name}	
PoolOpenスロット	pool.open_slots.{pool_name}	カウント	Pool, {pool_name}	
PoolQueuedスロット	pool.queued_slots.{pool_name}	カウント	Pool, {pool_name}	
PoolRunningスロット	pool.running_slots.{pool_name}	カウント	Pool, {pool_name}	
Processor Timeouts	dag_processing.processor_timeouts	カウント	関数、DAG 処理	
QueuedTasks	executor.queued_tasks	カウント	関数、エグゼキューター	
RunningTasks	executor.running_tasks	カウント	関数、エグゼキューター	

CloudWatch メトリクス	Apache Airflow メトリクス	単位	ディメンション	
TasksExecutable	scheduler.tasks.executable	カウント	関数、スケジュール	
TasksPending **Note** Apache Airflow v2.2 以上には適用されません。	scheduler.tasks.pending	カウント	関数、スケジュール	
TasksRunning	scheduler.tasks.running	カウント	関数、スケジュール	
TasksStarving	scheduler.tasks.starving	カウント	関数、スケジュール	
TasksWithoutDagRun	scheduler.tasks.without_dagrun	カウント	関数、スケジュール	

Apache Airflow Timers

このセクションの Apache Airflow メトリクスには、「Apache Airflow Timers」に関するデータが含まれています。

CloudWatch メトリクス	Apache Airflow メトリクス	単位	ディメンション	
CollectDBDags	collect_db_dags	ミリ秒	関数、DAG 処理	

CloudWatch メトリクス	Apache Airflow メトリクス	単位	ディメンション	
CriticalSection期間	scheduler.critical_section_duration	ミリ秒	関数、スケジュール	
CriticalSectionQueryDuration **Note** Apache Airflow v2.5.1 以上で利用可能です。	scheduler.critical_section_query_duration	ミリ秒	関数、スケジュール	
DAGDependencyCheck	dagrun.dependency-check.{dag_id}	ミリ秒	DAG, {dag_id}	
DAGDurationFailed	dagrun.duration.failed.{dag_id}	ミリ秒	DAG, {dag_id}	
DAGDurationSuccess	dagrun.duration.success.{dag_id}	ミリ秒	DAG, {dag_id}	
DAGFileProcessingLastDuration	dag_processing.last_duration.{dag_filename}	[秒]	DAG Filename, {dag_filename}	

CloudWatch メトリクス	Apache Airflow メトリクス	単位	ディメンション
DAGScheduleDelay	dagrun.schedule_delay.{dag_id}	ミリ秒	DAG, {dag_id}
FirstTaskSchedulingDelay	dagrun.{dag_id}.first_task_scheduling_delay	ミリ秒	DAG, {dag_id}
SchedulerLoop期間 **Note** Apache Airflow v2.5.1 以上で利用可能です。	scheduler.scheduler_loop_duration	ミリ秒	関数、スケジュール
TaskInstance期間	dag.{dag_id}.{task_id}.duration	ミリ秒	DAG, {dag_id} Task, {task_id}

CloudWatch メトリクス	Apache Airflow メトリクス	単位	ディメンション	
TaskInstanceQueuedDuration	dag.{dag_id}.{t .queued_duration Note Apache Airflow v2.7.2 以上で利用可能です。	ミリ秒	DAG, {dag_id} Task, {task_id}	
TaskInstanceScheduledDuration Note Apache Airflow v2.7.2 以上で利用可能です。	dag.{dag_id}.{t .scheduled_duration	ミリ秒	DAG, {dag_id} Task, {task_id}	

どのメトリクスを報告するかを選択する

次の Amazon MWAA 設定オプションを使用して CloudWatch、Apache Airflow に出力される、または Apache Airflow によってブロックされる Apache Airflow メトリクスを選択できます。

- **`metrics.metrics_allow_list`** — 環境 CloudWatch によって出力されるメトリクスを選択するために使用できるカンマ区切りプレフィックスのリスト。Apache Airflow で利用可能なメトリク

スをすべて送信させずに、要素のサブセットを選択させたい場合は、このオプションを使用してください。例えば scheduler,executor,dagrun です。

- **metrics.metrics_block_list** — リストの要素で始まるメトリクスを除外するための、カンマで区切られたプレフィックスのリスト。例えば scheduler,executor,dagrun です。

metrics.metrics_allow_list と metrics.metrics_block_list を両方構成した場合、Apache Airflow は metrics.metrics_block_list を無視します。metrics.metrics_block_list を構成して、metrics.metrics_allow_list を構成しなかった場合、Apache Airflow は metrics.metrics_block_list で指定した要素を除外します。

> **Note**
>
> metrics.metrics_allow_list および metrics.metrics_block_list設定オプションは、Apache Airflow v2.6.3 以降にのみ適用されます。Apache Airflow の以前のバージョンでは、metrics.statsd_block_list代わりに metrics.statsd_allow_listと を使用します。

次のステップ

- で環境ヘルスメトリクスを公開するために使用される Amazon MWAA API オペレーションについて説明しますPublishMetrics。

Amazon MWAA のコンテナ、キュー、およびデータベースメトリクス

Apache Airflow メトリクスに加えて、 を使用して Amazon Managed Workflows for Apache Airflow 環境の基盤となるコンポーネントをモニタリングできます。これにより CloudWatch、raw データを収集し、データを読み取り可能なほぼリアルタイムのメトリクスに加工できます。これらの環境メトリクスを使用すると、主要業績評価指標の可視性が高まり、環境の適切な規模を決定し、ワークフローの問題をデバッグするのに役立ちます。これらのメトリクスは、Amazon MWAA でサポートされているすべての Apache Airflow バージョンに適用されます。

Amazon MWAA は、各 Amazon Elastic Container Service (Amazon ECS) コンテナと Amazon Aurora PostgreSQL インスタンスの CPU とメモリの使用率、メッセージの数と最も古いメッセー

ジの経過時間に関する Amazon Simple Queue Service (Amazon SQS) メトリクス、データベース接続、ディスクキューの深さ、書き込みオペレーション、レイテンシー、スループットに関する Amazon Relational Database Service (Amazon RDS)のメトリクス、および Amazon RDS Proxy メトリクスを提供します。これらのメトリクスには、ベースワーカー、追加ワーカー、スケジューラー、およびウェブサーバーの数も含まれます。

これらの統計は 15 か月間保持されるため、履歴情報にアクセスしてスケジュールが失敗した理由をより的確に把握し、根本的な問題をトラブルシューティングできます。また、特定のしきい値をモニタリングするアラームを設定し、しきい値に達したときに通知を送信したりアクションを実行したりできます。詳細については、「Amazon ユーザーガイド CloudWatch 」を参照してください。

トピック

用語

名前空間

名前空間は、 AWS サービスの CloudWatch メトリクスのコンテナです。Amazon MWAA の名前空間は AWS/MWAA です。

CloudWatch メトリクス

CloudWatch メトリクスは、 に固有の時系列のデータポイントのセットを表します CloudWatch。

ディメンション

ディメンションは、メトリクスのアイデンティティの一部である名前と値のペアです。

単位

1 つの統計には、測定単位があります。Amazon MWAA の場合、単位にはカウントが含まれます。

ディメンション

このセクションでは、 の Amazon MWAA メトリクスの CloudWatch ディメンショングループについて説明します CloudWatch。

ディメンション	説明
クラスター	Amazon MWAA 環境が Apache Airflow コンポーネントの実行に使用する、少なくとも 3 つの Amazon MWAA コンテナのメトリクス (スケジューラー、ワーカー、ウェブサーバー)。
キュー	スケジューラーをワーカーから切り離す Amazon SQS キューのメトリクス。ワーカーがメッセージを読むと、そのメッセージは処理中と見なされ、他のワーカーは使用できません。12 時間の可視性タイムアウトまでにメッセージを削除しないと、他のワーカーがメッセージを読むことができるようになります。
データベース	Amazon MWAA が使用するAurora クラスターのメトリクス。これには、プライマリデータベースインスタンスと読み取りオペレーションをサポートするリードレプリカのメトリクスが含まれます。Amazon MWAA は READER インスタンスと WRITER インスタンスの両方のデータベースメトリクスを公開します。

CloudWatch コンソールでのメトリクスへのアクセス

このセクションでは、 で Amazon MWAA メトリクスにアクセスする方法について説明します CloudWatch。

ディメンションのパフォーマンスメトリクスを表示する方法

1. CloudWatch コンソールでメトリクスページを開きます。
2. AWS リージョンセレクタを使用して、リージョンを選択します。

3. [/MWAA] の名前空間を選択します。
4. すべてのメトリクス タブで、ディメンションを選択します。たとえば、[クラスター]。
5. ディメンションの CloudWatch メトリクスを選択します。例えば、 NumSchedulersや CPUUtilizationなどです。次に、[すべての検索結果をグラフ化] を選択します。
6. [グラフ化されたメトリクス] タブを選択すると、パフォーマンスメトリクスが表示されます。

メトリクスの一覧

次のテーブルは、Amazon MWAA のクラスター、キュー、データベースサービスのメトリクスを一覧表示されます。Amazon ECS、Amazon SQS、または Amazon RDS から直接生成されたメトリクスの説明を表示するには、それぞれのドキュメントリンクを選択してください。

トピック

- クラスターメトリクス
- データベースメトリクス
- キューメトリクス
- Application Load Balancer のメトリクス

クラスターメトリクス

以下のメトリクスは、各スケジューラー、ベースワーカー、追加ワーカー、ウェブサーバーに適用されます。各クラスターメトリクスの詳細と説明については、Amazon ECS ディベロッパーガイドの「利用可能なメトリクスとディメンション」を参照してください。

名前空間	メトリクス	Unit
AWS/MWAA	CPUUtilization	割合 (%)
AWS/MWAA	MemoryUtilization	割合 (%)

追加のワーカーコンテナとウェブサーバーコンテナの数の評価

次の手順で説明するように、クラスターディメンションで提供されるコンポーネントメトリクスを使用して、環境が特定の時点で使用している追加のワーカーまたはウェブサーバーの数を評価できます。これを行うには、CPUUtilization または MemoryUtilizationメトリクスのいずれかをグラフ化

し、統計タイプをサンプル数 に設定します。結果の値は、AdditionalWorker コンポーネントの RUNNING タスク総数です。環境で使用される追加のワーカーインスタンスの数を理解することは、環境のスケーリング方法を判断し、追加のワーカーの数を最適化するのに役立ちます。

Workers

を使用して追加のワーカーの数を評価するには AWS Management Console

1. [AWS/MWAA] の名前空間を選択します。
2. [すべてのメトリクス] タブで、[クラスター] ディメンションを選択します。
3. クラスターディメンションで、 に CPUUtilization または MemoryUtilizationメトリクスAdditionalWorkerを選択します。
4. [グラフ化したメトリクス] タブで、[期間] を [1 分] に、[統計] を [サンプル数] に設定します。

Web servers

を使用して追加のウェブサーバーの数を評価するには AWS Management Console

1. [AWS/MWAA] の名前空間を選択します。
2. [すべてのメトリクス] タブで、[クラスター] ディメンションを選択します。
3. クラスターディメンションで、 に CPUUtilization または MemoryUtilizationメトリクスAdditionalWebserversを選択します。
4. [グラフ化したメトリクス] タブで、[期間] を [1 分] に、[統計] を [サンプル数] に設定します。

詳細については、「Amazon Elastic Container Service デベロッパーガイド」の「サービスの RUNNING タスク数」を参照してください。

データベースメトリクス

次のメトリクスは、Amazon MWAA 環境に関連付けられている各データベースインスタンスに適用されます。

名前空間	メトリクス	Unit
AWS/MWAA	CPUUtilization	割合 (%)

名前空間	メトリクス	Unit
AWS/MWAA	DatabaseConnections	カウント
AWS/MWAA	DiskQueueDepth	カウント
AWS/MWAA	FreeableMemory	バイト
AWS/MWAA	VolumeWriteIOPS	5 分あたりのカウント
AWS/MWAA	WriteIOPS	1 秒あたりのカウント数
AWS/MWAA	WriteLatency	[秒]
AWS/MWAA	WriteThroughput	1 秒あたりのバイト数

キューメトリクス

以下のキューメトリクスの単位と説明の詳細については、Amazon SQS で使用できる CloudWatch メトリクス」を参照してください。

名前空間	メトリクス	Unit
AWS/MWAA	ApproximateAgeOfOldestTask	[秒]
AWS/MWAA	RunningTasks	カウント
AWS/MWAA	QueuedTasks	カウント

Application Load Balancer のメトリクス

Application Load Balancer メトリクスは、環境で実行されているウェブサーバーに適用されます。Amazon MWAA は、これらのメトリクスを使用して、トラフィック量に基づいてウェブサーバーをスケーリングします。以下のロードバランサーメトリクスの単位と説明の詳細について

は、CloudWatch Application Load Balancer」の「Application Load Balancer のメトリクス」を参照してください。

名前空間	メトリクス	Unit
AWS/MWAA	ActiveConnectionCount	カウント

Amazon Managed Workflows for Apache Airflow におけるセキュリティー

のクラウドセキュリティが最優先事項 AWS です。お客様は AWS 、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 (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、およびクラウド内で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、 では、安全に使用できるサービスも提供しています。コンプライアンスAWS 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 MWAA 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。
- クラウドのセキュリティ — お客様の責任は、使用する AWS サービスによって決まります。また、お客様は、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 Managed Workflows for Apache Airflow 使用時における責任共有モデルの適用法を理解するのに役立ちます。ここでは、セキュリティやコンプライアンスに関する目標を達成できるようにAmazon MWAAを構成する方法について説明します。また、Amazon MWAA リソースのモニタリングや保護に役立つ他の AWS のサービスの使用方法についても説明します。

このセクションの内容:

- Amazon Managed Workflows for Apache Airflow におけるデータ保護
- AWS Identity and Access Management
- Apache Airflow 用 Amazon マネージドワークフローのコンプライアンス検証
- Amazon Managed Workflows for Apache Airflow におけるレジリエンス
- Amazon MWAA のインフラストラクチャセキュリティ
- Amazon MWAA での構成と脆弱性の分析
- Amazon MWAA のセキュリティのベストプラクティス

Amazon Managed Workflows for Apache Airflow におけるデータ保護

責任 AWS 共有モデル、Amazon Managed Workflows for Apache Airflow のデータ保護に適用されます。このモデルで説明されているように、 AWS はすべての を実行するグローバルインフラストラクチャを保護する責任があります AWS クラウド。お客様は、このインフラストラクチャでホストされているコンテンツに対する管理を維持する責任があります。このコンテンツには、使用する AWS のサービス のセキュリティ構成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に投稿された記事「AWS 責任共有モデルおよび GDPR」を参照してください。

データ保護の目的で、 (AWS Identity and Access Management IAM) を使用して AWS アカウント 認証情報を保護し、個々のユーザー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、 サービス内のすべての AWS デフォルトのセキュリティコントロールを使用します。
- Amazon Macie などのアドバンストマネージドセキュリティサービスを使用します。これは、Amazon S3 に保存されている個人データの検出と保護を支援します。

顧客の E メールアドレスなどの機密情報やセンシティブ情報は、タグや [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、または AWS CLI SDK を使用して Amazon MWAA 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 タグまたは名前に使用する自由記入欄に入力したデータは、課金や診断ログに使用される場合があります。外部サーバーへ URL を供給する場合は、そのサーバーへのリクエストを検証するために、認証情報を URL に含めないことを強くお勧めします。

Amazon MWAA での暗号化

以下のトピックでは、Amazon MWAA が保管中および転送中のデータを保護する方法について説明します。この情報を使用して、Amazon MWAA が と統合 AWS KMS して保管中のデータを暗号化す

る方法と、転送中の Transport Layer Security (TLS) プロトコルを使用してデータを暗号化する方法について説明します。

トピック

保管中の暗号化

Amazon MWAA では、保管中のデータとは、サービスが永続メディアに保存するデータです。

保管中のデータの暗号化には「AWS 所有キー」を使用することも、環境の作成時にオプションで「カスタマーマネージドキー」を提供して追加の暗号化を行うこともできます。カスタマーマネージド KMS キーを使用する場合は、環境で使用する他の AWS リソースやサービスと同じアカウントに存在する必要があります。

カスタマーマネージド KMS キーを使用するには、キーポリシー CloudWatch にアクセスするために必要なポリシーステートメントをアタッチする必要があります。お客様の環境でカスタマーマネージド KMS キーを使用する場合、Amazon MWAA はお客様に代わって 4 つの「許可」をアタッチします。Amazon MWAA がカスタマーマネージド KMS キーに付与する許可の詳細については、「データ暗号化用のカスタマーマネージドキー」を参照してください。

カスタマー管理の KMS キーを指定しない場合、Amazon MWAA はデフォルトで の AWS 所有の KMS キーを使用してデータを暗号化および復号します。Amazon MWAA でのデータ暗号化を管理するには、 AWS 所有の KMS キーを使用することをお勧めします。

> **Note**
>
> Amazon MWAA での AWS 所有またはカスタマーマネージド KMS キーのストレージと使用に対して料金が発生します。詳細については、「AWS KMS の料金」を参照してください。

暗号化アーティファクト

Amazon MWAA 環境を作成するときに、「AWS 所有キー」または「カスタマーマネージドキー」を指定して、保管時の暗号化に使用する暗号化アーティファクトを指定します。Amazon MWAA は、指定されたキーに必要な「許可」を追加します。

[Amazon S3] — Amazon S3 データは、サーバー側の暗号化 (SSE) を使用してオブジェクトレベルで暗号化されます。Amazon S3 の暗号化と復号化は、DAG コードとサポートファイルが保存される Amazon S3 バケットで行われます。オブジェクトは Amazon S3 にアップロードされるときに暗号化され、Amazon MWAA 環境にダウンロードされるときに復号化されます。デフォルトでは、カスタマー管理の KMS キーを使用している場合、Amazon MWAA はそのキーを使用して Amazon S3 バケットのデータを読み取り、復号化します。

CloudWatch ログ – AWS 所有の KMS キーを使用している場合、 CloudWatch ログに送信される Apache Airflow ログは、ログ AWS が所有する KMS キーを持つサーバー側の暗号化 (SSE) CloudWatch を使用して暗号化されます。カスタマーマネージド KMS キーを使用している場合は、KMS キーにキーポリシーを追加して、 CloudWatch ログがキーを使用できるようにする必要があります。

[Amazon SQS] — Amazon MWAA は、お使いの環境用に 1 つの Amazon SQS キューを作成します。Amazon MWAA は、サーバー側の暗号化 (SSE) を使用して、 AWS 所有の KMS キー、または指定したカスタマー管理の KMS キーを使用して、キューとの間で送受信されるデータの暗号化を処理します。 AWS 所有の KMS キーとカスタマー管理の KMS キーのどちらを使用しているかにかかわらず、Amazon SQS アクセス許可を実行ロールに追加する必要があります。

[Aurora PostgreSQL] — Amazon MWAA は、お使いの環境に合わせて 1 つの PostgreSQL クラスターを作成します。Aurora PostgreSQL は、サーバー側の暗号化 (SSE) を使用して、 AWS 所有またはカスタマー管理の KMS キーでコンテンツを暗号化します。カスタマーマネージドの KMS キーを使用している場合、Amazon RDS はキーに少なくとも 2 つの権限を追加します。1 つはクラスター用、もう 1 つはデータベースインスタンス用です。カスタマーマネージド KMS キーを複数の環境で使用することを選択した場合、Amazon RDS は追加の権限を作成することがあります。詳細については、Amazon RDSRDS におけるデータ保護を参照してください。

転送中の暗号化

転送中のデータとは、ネットワークを移動する際に傍受される可能性があるデータと呼ばれます。

Transport Layer Security (TLS) は、環境の Apache Airflow コンポーネントと、Amazon S3 などの Amazon MWAA と統合する他の AWS サービスの間で転送中の Amazon MWAA オブジェクトを暗号化します。Amazon S3 暗号化の使用の詳細については、「暗号化でデータを保護する」を参照してください。

暗号化のためのカスタマーマネージドキーの使用

オプションで、環境内のデータ暗号化するための「カスタマーマネージドキー」を提供できます。カスタマーマネージド KMS キーは、Amazon MWAA 環境インスタンスと、ワークフローのリソース

を保存する Amazon S3 バケットと同じリージョンに作成する必要があります。指定するカスタマーマネージド KMS キーが、クラスターの構成に使用するアカウントとは異なるアカウントにある場合は、その ARN を使用してキーをクロスアカウントアクセスに指定する必要があります。キーの作成の詳細については、「AWS Key Management Service デベロッパーガイド」の「キーの作成」を参照してください。

サポート対象

AWS KMS 機能	サポート対象	
「AWS KMS キー ID または ARN」。	はい	
「AWS KMS キーエイリアス」。	いいえ	
「AWS KMS マルチリージョンキー」。	いいえ	

暗号化のための許可の使用。

このトピックでは、データの暗号化および復号化のために、Amazon MWAA がお客様に代わってカスタマーマネージド KMS キーに付与する許可について説明します。

仕組み

カスタマーマネージド KMS キー AWS KMS には、 でサポートされるリソースベースのアクセスコントロールメカニズムとして、キーポリシーとグラントの 2 つがあります。

キーポリシーは、権限がほとんど静的で、同期サービスモードで使用される場合に使用されます。許可は、サービスが自身や他のアカウントに異なるアクセス権限を定義する必要がある場合など、より動的で詳細な権限が必要な場合に使用されます。

Amazon MWAA では、4 つの付与ポリシーを使用してカスタマーマネージド KMS キーにアタッチします。これは、環境が CloudWatch ログ、Amazon SQS キュー、Aurora PostgreSQL データベース、Secrets Manager シークレット、Amazon S3 バケット、DynamoDB テーブルから保管中のデータを暗号化するために必要なきめ細かいアクセス許可によるものです。

Amazon MWAA 環境を作成してカスタマーマネージド KMS キーを指定すると、Amazon MWAA はその許可ポリシーをカスタマーマネージド KMS キーにアタッチします。これらのポリシーにより、`airflow.region}.amazonaws.com` 内の Amazon MWAA はお客様のカスタマーマネージド KMS キーを使用して、お客様に代わって Amazon MWAA が所有するリソースを暗号化することができます。

Amazon MWAA は、お客様に代わって指定された KMS キーに追加の権限を作成し、アタッチします。これには、環境を削除した場合にグラントを廃止するポリシー、クライアント側の暗号化 (CSE) にカスタマーマネージド KMS キーを使用するポリシー、Secrets Manager のカスタマーマネージドキーで保護されたシークレットにアクセスする必要がある AWS Fargate 実行ロールを使用するポリシーが含まれます。

許可ポリシー

Amazon MWAA は、お客様に代わって以下の「リソースベースのポリシー」許可をカスタマーマネージドの KMS キーに追加します。これらのポリシーにより、被付与者とプリンシパル (Amazon MWAA) は、ポリシーで定義されたアクションを実行できます。

許可 1: データプレーンリソースの作成に使用

```
{
            "Name": "mwaa-grant-for-env-mgmt-role-environment name",
            "GranteePrincipal": "airflow.region.amazonaws.com",
            "RetiringPrincipal": "airflow.region.amazonaws.com",
            "Operations": [
              "kms:Encrypt",
              "kms:Decrypt",
              "kms:ReEncrypt*",
              "kms:GenerateDataKey*",
              "kms:CreateGrant",
              "kms:DescribeKey",
              "kms:RetireGrant"
            ]
          }
```

許可 2: **`ControllerLambdaExecutionRole`** アクセスに使用

```
{
            "Name": "mwaa-grant-for-lambda-exec-environment name",
            "GranteePrincipal": "airflow.region.amazonaws.com",
```

```
            "RetiringPrincipal": "airflow.region.amazonaws.com",
            "Operations": [
              "kms:Encrypt",
              "kms:Decrypt",
              "kms:ReEncrypt*",
              "kms:GenerateDataKey*",
              "kms:DescribeKey",
              "kms:RetireGrant"
            ]
          }
```

許可 3: **CfnManagementLambdaExecutionRole** アクセスに使用

```
{
            "Name": " mwaa-grant-for-cfn-mgmt-environment name",
            "GranteePrincipal": "airflow.region.amazonaws.com",
            "RetiringPrincipal": "airflow.region.amazonaws.com",
            "Operations": [
              "kms:Encrypt",
              "kms:Decrypt",
              "kms:ReEncrypt*",
              "kms:GenerateDataKey*",
              "kms:DescribeKey"
            ]
          }
```

許可 4: バックエンドシークレットにアクセスするための Fargate 実行ロールに使用

```
{
              "Name": "mwaa-fargate-access-for-environment name",
              "GranteePrincipal": "airflow.region.amazonaws.com",
              "RetiringPrincipal": "airflow.region.amazonaws.com",
              "Operations": [
                "kms:Encrypt",
                "kms:Decrypt",
                "kms:ReEncrypt*",
                "kms:GenerateDataKey*",
                "kms:DescribeKey",
                "kms:RetireGrant"
              ]
            }
```

カスタマーマネージドキーへのキーポリシーの適用

Amazon MWAA で独自のカスタマーマネージド KMS キーを使用する場合は、Amazon MWAA がキーを使用してデータを暗号化できるように、以下のポリシーをキーにアタッチする必要があります。

Amazon MWAA 環境に使用したカスタマーマネージド KMS キーが で動作するようにまだ設定されていない場合は CloudWatch、暗号化された CloudWatch ログを許可するようにキーポリシーを更新する必要があります。詳細については、「サービス CloudWatch を使用して でログデータを暗号化する AWS Key Management Service」を参照してください。

次の例は、 CloudWatch ログのキーポリシーを表します。リージョンに提供されているサンプル値に置き換えてください。

```
{
          "Effect": "Allow",
          "Principal": {
          "Service": "logs.us-west-2.amazonaws.com"
        },
        "Action": [
          "kms:Encrypt*",
          "kms:Decrypt*",
          "kms:ReEncrypt*",
          "kms:GenerateDataKey*",
          "kms:Describe*"
        ],
        "Resource": "*",
        "Condition": {
          "ArnLike": {
            "kms:EncryptionContext:aws:logs:arn": "arn:aws:logs:us-west-2:*:*"
            }
          }
        }
```

AWS 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するのに役立つ AWS サービスです。IAM 管理者は、誰を認証 (サインイン) し、誰に Amazon

Managed Workflows for Apache Airflow 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、追加料金なしで使用できる AWS のサービスです。

このトピックでは、Amazon MWAA が AWS Identity and Access Management (IAM) を使用する方法の基本的概要を説明します。Amazon MWAA へのアクセスの管理については、「Amazon MWAA 環境へのアクセスの管理」を参照してください。

内容

対象者

AWS Identity and Access Management (IAM) の使用方法は、Amazon MWAA 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 MWAA 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 MWAA 機能を使用して作業を行うには、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しておくと、管理者に適切な許可をリクエストするうえで役立ちます。Amazon MWAA の機能にアクセスできない場合は、「Apache Airflow 用 Amazon マネージドワークフローのアイデンティティ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 MWAA リソースを担当している場合は、通常、Amazon MWAA へのフルアクセスがあります。サービスのユーザーがどの Amazon MWAA 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。会社で Amazon MWAA と IAM を併用する方法の詳細については、「Amazon MWAA で IAM が機能する仕組み」を参照してください。

IAM 管理者 - 管理者は、Amazon MWAA へのアクセス権を管理するポリシーの作成方法の詳細について確認する場合があります。IAM で使用できる Amazon MWAA のアイデンティティベースポリ

シーの例を確認するには、「Amazon MWAA のアイデンティティベースポリシーの例」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（ にサインイン AWS) される必要があります。

ID ソースを介して提供された認証情報を使用して、フェデレーティッド ID AWS として にサインインできます。 AWS IAM Identity Center（ IAM Identity Center) ユーザー、会社のシングルサインオン認証、Google または Facebook の認証情報は、フェデレーション ID の例です。フェデレーティッドアイデンティティ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 AWS を使用して にアクセスすると、間接的にロールを引き受けることになります。

ユーザーのタイプに応じて、 AWS Management Console または AWS アクセスポータルにサインインできます。へのサインインの詳細については AWS、「 ユーザーガイド」の「 にサインインする方法 AWS アカウントAWS サインイン 」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。推奨される方法を使用してリクエストを自分で署名する方法の詳細については、IAM ユーザーガイドの API AWS リクエストの署名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、 AWS では、多要素認証 (MFA) を使用してアカウントのセキュリティを向上させる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「Multi-factor authentication」(多要素認証) および『IAM ユーザーガイド』の「AWSにおける多要素認証 (MFA) の使用」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 AWS のサービス およびリソースへの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。

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対して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権限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、ロールでは一時的な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 (ロールではなく) の作成が適している場合」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。ロール を切り替える AWS Management Console ことで、 で IAM ロールを一時的に引き受けることができます。ロールを引き受けるには、 または AWS API AWS CLI オペレーションを呼び出すか、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「IAM ロールの使用」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アイデンティティに権限を割り当てるには、ロールを作成してそのロールの権限を定義します。フェデレーティッドアイデンティティが認証されると、そのアイデンティティはロールに関連付けられ、ロールで定義されている権限

が付与されます。フェデレーションの詳細については、『IAM ユーザーガイド』の「サードパーティーアイデンティティプロバイダー向けロールの作成」を参照してください。IAM アイデンティティセンターを使用する場合、権限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。権限セットの詳細については、『AWS IAM Identity Center ユーザーガイド』の「権限セット」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) ポリシーをリソースに直接アタッチできます。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 ロールとリソースベースのポリシーとの相違点」を参照してください。
- クロスサービスアクセス — 一部の 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権限、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使用してこれを行う場合があります。
  - 転送アクセスセッション (FAS) – IAM ユーザーまたはロールを使用して でアクションを実行する場合 AWS、ユーザーはプリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行することで、別のサービスの別のアクションがトリガーされることがあります。FAS は、 を呼び出すプリンシパルのアクセス許可を AWS のサービス、ダウンストリームサービス AWS のサービス へのリクエストのリクエストと組み合わせて使用します。FAS リクエストは、サービスが他の AWS のサービス またはリソースとのやり取りを完了する必要があるリクエストを受け取った場合にのみ行われます。この場合、両方のアクションを実行するための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 のサービスにアクセス許可を委任するロールの作成」を参照してください。
  - サービスにリンクされたロール – サービスにリンクされたロールは、 にリンクされたサービスロールの一種です AWS のサービス。サービスは、ユーザーに代わってアクションを実行する

ロールを引き受けることができます。サービスにリンクされたロールは に表示され AWS アカウント 、サービスによって所有されます。IAM 管理者は、サービスにリンクされたロールの権限を表示できますが、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 リクエストを行うアプリケーションの一時的な認証情報を管理できます。これは、EC2 インスタンス内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 ロールを使用して権限を付与する」を参照してください。

IAM ロールと IAM ユーザーのどちらを使用するかについては、『IAM ユーザーガイド』の「(IAM ユーザーではなく) IAM ロールをいつ作成したら良いのか?」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 AWS には、ポリシーを作成し、 AWS ID またはリソースにアタッチします。ポリシーは AWS 、アイデンティティまたはリソースに関連付けられているときにアクセス許可を定義する のオブジェクトです。 は、プリンシパル (ユーザー、ルートユーザー、またはロールセッション) がリクエストを行うときに、これらのポリシー AWS を評価します。ポリシーでの権限により、リクエストが許可されるか拒否されるかが決まります。ほとんどのポリシーは JSON ドキュメント AWS として に保存されます。JSON ポリシードキュメントの構造と内容の詳細については、「IAM ユーザーガイド」の「JSON ポリシー概要」を参照してください。

管理者は AWS JSON 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んなリソースにどんな条件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継ぐことができます。

IAM ポリシーは、オペレーションの実行方法を問わず、アクションの権限を定義します。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、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 権限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。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 ポリシーの作成」を参照してください。

アイデンティティベースポリシーは、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「マネージドポリシーとインラインポリシーの比較」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権限を持つかをコントロールします。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ます。これらのポリシータイプでは、より一般的なポリシータイプで付与された最大の権限を設定できます。

- アクセス許可の境界 - アクセス許可の境界は、アイデンティティベースのポリシーによって 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。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) に対する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、 AWS アカウント ビジネスが所有する複数の をグループ化して一元管理するサービスです。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「SCP の仕組み」を参照してください。
- セッションポリシー - セッションポリシーは、ロールまたはフェデレーションユーザーの一時的なセッションをプログラムで作成する際にパラメータとして渡す高度なポリシーです。結果としてセッションの権限は、ユーザーまたはロールのアイデンティティベースポリシーとセッションポリシーの共通部分になります。また、リソースベースのポリシーから権限が派生する場合もあります。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。詳細については、「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与する場合にリクエストを許可するかどうか AWS を決定する方法については、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。

ユーザー自身のアクセス許可を表示すること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": "*"
        }
    ]
}
```

Apache Airflow 用 Amazon マネージドワークフローのアイデンティティとアクセスのトラブルシューティング

以下の情報を使用して、Amazon MWAA 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と修正に役立てます。

Amazon MWAA でアクションを実行する権限がありません

がアクションを実行する権限がないと AWS Management Console 通知した場合は、管理者に連絡してサポートを依頼する必要があります。担当の管理者はお客様のユーザー名とパスワードを発行した人です。

iam を実行する権限がありません。PassRole

`iam:PassRole` アクションを実行する権限がないというエラーが表示された場合は、ポリシーを更新して Amazon MWAA 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、そのサービスに既存のロールを渡すことができます。そのためには、サービスにロールを渡す権限が必要です。

次の例のエラーは、`marymajor` という IAM ユーザーがコンソールを使用して Amazon MWAA 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。Mary には、ロールをサービスに渡す権限がありません。

```
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。サインイン資格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 MWAA 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 MWAA がこれらの機能をサポートしているかどうかを確認するには、「Amazon MWAA で IAM が機能する仕組み」を参照してください。
- 所有 AWS アカウント している のリソースへのアクセスを提供する方法については、IAM ユーザーガイドの「所有 AWS アカウント している別の の IAM ユーザーへのアクセスを提供する」を参照してください。
- 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) へのアクセス権限」を参照してください。
- クロスアカウントアクセスでのロールとリソースベースのポリシーの使用の違いの詳細については、「IAM ユーザーガイド」の「IAM ロールとリソースベースのポリシーとの相違点」を参照してください。

Amazon MWAA で IAM が機能する仕組み

Amazon MWAA は IAM アイデンティティベースのポリシーを使用して、Amazon MWAA アクションおよびリソースにアクセス許可を付与します。Amazon MWAA リソースへのアクセスを制御するために使用できるカスタム IAM ポリシーの推奨例については、「the section called “Amazon MWAA 環境へのアクセス”」を参照してください。

Amazon MWAA およびその他の AWS のサービスが IAM 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 MWAA アイデンティティベースのポリシー

IAM アイデンティティベースポリシーでは、許可または拒否するアクションとリソース、またアクションを許可または拒否する条件を指定できます。Amazon MWAA は、特定のアクション、リソース、および条件キーをサポートしています。

次のステップは、IAM コンソールを使用して新しい JSON ポリシーを作成する方法を示しています。このポリシーは、Amazon MWAA リソースへの読み取り専用アクセスを提供します。

JSON ポリシーエディタでポリシーを作成するには

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. 左側のナビゲーションペインで、[ポリシー] を選択します。

   初めて [ポリシー] を選択する場合には、[管理ポリシーにようこそ] ページが表示されます。[今すぐ始める] を選択します。
3. ページの上部で、[ポリシーを作成] を選択します。
4. [ポリシーエディタ] セクションで、[JSON] オプションを選択します。
5. 次の JSON ポリシードキュメントを入力します。

```
{
        "Version": "2012-10-17",
        "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "airflow:ListEnvironments",
                "airflow:GetEnvironment",
                "airflow:ListTagsForResource"
            ],
            "Resource": "*"
        }
    ]
}
```

6. [次へ] をクリックします。

> **Note**
>
> いつでも [Visual] と [JSON] エディタオプションを切り替えることができます。ただし、[Visual] エディタで [次] に変更または選択した場合、IAM はポリシーを再構成して visual エディタに合わせて最適化することがあります。詳細については、「IAM ユーザーガイド」の「ポリシーの再構成」を参照してください。

7. [確認と作成] ページで、作成するポリシーの [ポリシー名] と [説明] (オプション) を入力します。[このポリシーで定義されているアクセス許可] を確認して、ポリシーによって付与されたアクセス許可を確認します。

8. [ポリシーの作成] をクリックして、新しいポリシーを保存します。

JSON ポリシーで使用するすべての要素については、「IAM ユーザーガイド」の「IAM JSON ポリシー要素のリファレンス」を参照してください。

アクション

管理者は AWS JSON 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んなリソースにどんな条件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、関連付けられた 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権限のみのアクションなど、いくつかの例外があります。また、ポリシーに複数アクションが必要なオペレーションもあります。これらの追加アクションは、 依存アクション と呼ばれます。

このアクションは、関連付けられたオペレーションを実行するための権限を付与するポリシーで使用されます。

ポリシーステートメントには、Action 要素または NotAction 要素を含める必要があります。Action 要素は、ポリシーによって許可されるアクションをリストします。NotAction 要素は、許可されていないアクションをリストします。

Amazon MWAA のために定義されたアクションには、Amazon MWAA を使用して実行できるタスクが反映されます。Detective のポリシーアクションには、プレフィックス airflow: が付いています。

複数のアクションを指定するために、ワイルドカード (*) を使用することもできます。これらのアクションを個別にリストする代わりに、たとえば environment という単語で終わるすべてのアクションへのアクセス権を付与できます。

Amazon Managed Workflows for Apache Airflow アクションのリストを確認するには、IAM ユーザーガイドの Amazon MWAA で定義されるアクションを参照してください。

Amazon MWAA のアイデンティティベースポリシーの例

Amazon MWAA ポリシーを表示するには、「Amazon MWAA 環境へのアクセスの管理」を参照してください。

デフォルトでは、IAM ユーザーおよびロールには Amazon MWAA リソースを作成または変更するアクセス許可はありません。また、、 AWS Management Console AWS CLI、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、ユーザーとロールに必要な、指定されたリソースで特定の API オペレーションを実行する権限をユーザーとロールに付与する IAM ポリシーを作成する必要があります。その後、管理者はそれらの許可が必要な IAM ユーザーまたはグループにそのポリシーをアタッチします。

> **Important**
>
> IAM ロールと一時的な認証情報を使用して Amazon MWAA リソースにアクセスすることをお勧めします。アクセス権限ポリシーを IAM ユーザーに直接アタッチすることは避けてください。

JSON ポリシードキュメントのこれらの例を使用して、IAM アイデンティティベースポリシーを作成する方法については、「IAM ユーザーガイド」の「JSON タブでのポリシーの作成」を参照してください。

トピック

- ポリシーのベストプラクティス
- Amazon MWAA コンソールの使用
- ユーザー自身のアクセス許可を表示することをユーザーに許可する

ポリシーの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、アカウント内で誰が Amazon MWAA リソースを作成、アクセス、または削除できるを決定します。これらのアクションを実行すると、 AWS アカウントに料金が発生する可能性があります。アイデンティティベースポリシーを作成したり編集したりする際には、以下のガイドラインと推奨事項に従ってください:

- AWS 管理ポリシーを開始し、最小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を付与するには、多くの一般的なユースケースにアクセス許可を付与する AWS 管理ポリシーを使用します。これらは で使用できます AWS アカウント。ユースケースに固有の AWS カスタマー管理ポリシーを定義して、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「AWS ジョブ機能の管理ポリシー」を参照してください。

- 最小特権を適用する – IAM ポリシーで権限を設定するときは、タスクの実行に必要な権限のみを付与します。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、最小特権権限とも呼ばれています。IAM を使用して権限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 でのポリシーと権限」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、 などの特定の を介してサービスアクションが使用される場合に 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 policy elements: Condition] (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 ポリシーの検証」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 保護 API アクセスの設定」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 MWAA コンソールの使用

Amazon MWAA コンソールを使用するには、ユーザーまたはロールが、関連するアクションであって、API の対応するアクションに一致するアクションにアクセスできる必要があります。

Amazon MWAA ポリシーを表示するには、「Amazon MWAA 環境へのアクセスの管理」を参照してください。

ユーザー自身のアクセス許可を表示すること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": "*"
        }
    ]
}
```

Apache Airflow 用 Amazon マネージドワークフローのコンプライアンス検証

AWS のサービス が特定のコンプライアンスプログラムの範囲内にあるかどうかを確認するには、コンプライアンスプログラムAWS のサービス による対象範囲内のコンプライアンスプログラムを参照し、関心のあるコンプライアンスプログラムを選択します。一般的な情報については、AWS「コンプライアンスプログラム」を参照してください。

を使用して、サードパーティーの監査レポートをダウンロードできます AWS Artifact。詳細については、「 でのレポートのダウンロード AWS Artifact」の」を参照してください。

を使用する際のお客様のコンプライアンス責任 AWS のサービス は、お客様のデータの機密性、貴社のコンプライアンス目的、適用される法律および規制によって決まります。 では、コンプライアンスに役立つ以下のリソース AWS を提供しています。

- セキュリティとコンプライアンスのクイックスタートガイド – これらのデプロイガイドでは、アーキテクチャ上の考慮事項について説明し、セキュリティとコンプライアンスに重点を置いたベースライン環境 AWS を にデプロイする手順を示します。
- アマゾン ウェブ サービスにおける HIPAA セキュリティとコンプライアンスのアーキテクチャ — このホワイトペーパーでは、企業が AWS を使用して HIPAA 対象アプリケーションを作成する方法について説明します。

> **Note**
>
> すべて AWS のサービス HIPAA の対象となるわけではありません。詳細については、「HIPAA 対応サービスのリファレンス」を参照してください。

- AWS コンプライアンスリソース – このワークブックとガイドのコレクションは、お客様の業界や地域に適用される場合があります。
- AWS カスタマーコンプライアンスガイド – コンプライアンスの観点から責任共有モデルを理解します。このガイドでは、ガイダンスを保護し AWS のサービス 、複数のフレームワーク (米国国立標準技術研究所 (NIST)、Payment Card Industry Security Standards Council (PCI)、国際標準化機構 (ISO) を含む) のセキュリティコントロールにマッピングするためのベストプラクティスをまとめています。
- 「 デベロッパーガイド」の「 ルールによるリソースの評価」 – この AWS Config サービスは、リソース設定が社内プラクティス、業界ガイドライン、および規制にどの程度準拠しているかを評価します。 AWS Config
- AWS Security Hub – これにより AWS のサービス 、 内のセキュリティ状態を包括的に確認できます AWS。Security Hub では、セキュリティコントロールを使用して AWS リソースを評価し、セキュリティ業界標準とベストプラクティスに対するコンプライアンスをチェックします。サポートされているサービスとコントロールのリストについては、「Security Hub のコントロールリファレンス」を参照してください。
- Amazon GuardDuty – これにより AWS アカウント、疑わしいアクティビティや悪意のあるアクティビティがないか環境を監視することで、、、ワークロード、コンテナ、データに対する潜在的な脅威 AWS のサービス を検出します。 GuardDuty は、特定のコンプライアンスフレームワーク

で義務付けられている侵入検知要件を満たすことで、PCI DSS などのさまざまなコンプライアンス要件への対応に役立ちます。

- AWS Audit Manager – これにより AWS のサービス 、 AWS 使用状況を継続的に監査し、リスクの管理方法と規制や業界標準への準拠を簡素化できます。

Amazon Managed Workflows for Apache Airflow におけるレジリエンス

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。リージョンには、低レイテンシー、高いスループット、そして高度の冗長ネットワークで接続されている複数の物理的に独立および隔離されたアベイラビリティーゾーンがあります。アベイラビリティーゾーンでは、ゾーン間で中断することなく自動的にフェイルオーバーするアプリケーションとデータベースを設計および運用することができます。アベイラビリティーゾーンは、従来の単一または複数のデータセンターインフラストラクチャよりも可用性が高く、フォールトトレラントで、スケーラブルです。

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 MWAA 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 Managed Workflows for Apache Airflow は、 AWS グローバル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、「 Security Pillar AWS Well-Architected Framework」の「Infrastructure Protection」を参照してください。

が AWS 公開した API コールを使用して、ネットワーク経由で Amazon MWAA 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 は必須で TLS 1.3 がお勧めです。
- 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。

Amazon MWAA での構成と脆弱性の分析

設定と IT コントロールは、 AWS とお客様の間で共有される責任です。

Apache Airflow 用 Amazon マネージドワークフローに定期的にパッチを適用し、環境の Apache Airflow をアップグレードします。VPC には適切なアクセスポリシーが使用されていることを確認する必要があります。

詳細については、以下のリソース を参照してください。

- Apache Airflow 用 Amazon マネージドワークフロー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
- Amazon MWAA のインフラストラクチャセキュリティ
- Amazon MWAA のセキュリティのベストプラクティス

Amazon MWAA のセキュリティのベストプラクティス

Amazon MWAA 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必ずしも適切または十分でない可能性があるので、処方箋ではなく、あくまで有用な考慮事項とお考えください。

- 最小許可ポリシーを使用します。ユーザーがタスクを実行するのに必要なリソースまたはアクションのみにアクセス許可を付与します。
- AWS CloudTrail を使用して、アカウントのユーザーアクティビティをモニタリングします。
- Amazon MWAA バケットポリシーとオブジェクト ACL が、関連する Amazon MWAA 環境のユーザーにオブジェクトをバケットに入れるアクセス権限を付与していることを確認してください。これにより、バケットにワークフローを追加する権限を持つユーザーには、Airflow でワークフローを実行する権限も付与されます。

- Amazon MWAA 環境に関連付けられた Amazon S3 バケットを使用します。お使いのAmazon S3 バケットにはどのような名前でもかまいません。他のオブジェクトをバケットに保存したり、別のサービスでバケットを使用したりしないでください。

Apache Airflow でのセキュリティのベストプラクティス

Apache Airflow はマルチテナントではありません。特定のユーザーに一部の機能を制限する「アクセス制御手段」があり、「Amazon MWAAが実装」している一方で、DAG 作成者は Apache Airflow ユーザー権限を変更し、基盤となるメタデータベースと対話する DAG を記述する能力を持っています。

Amazon MWAA で Apache Airflow を使用するときは、環境のメタデータデータベースと DAG の安全を確保するために、次の手順を実行することをお勧めします。

- 「Amazon MWAA 実行ロール」または「Apache Airflow 接続」でアクセスできるものには、その環境に書き込むことができるユーザーもアクセスできると仮定して、DAG 書き込みアクセス権を持つチームや Amazon S3 `/dags` フォルダにファイルを追加できる個別のチームには、別々の環境を使用してください。
- Amazon S3 DAG フォルダへの直接アクセスを提供しないでください。代わりに、CI/CD ツールを使用して Amazon S3 に DAG を書き込み、DAG コードがチームのセキュリティガイドラインを満たしていることを確認する検証ステップを行います。
- お使いの環境の Amazon S3 バケットへのユーザーアクセスを防ぎます。代わりに、DAG を格納するお使いの Amazon MWAA Amazon S3 バケットとは別の場所に保存されている YAML、JSON、またはその他の定義ファイルに基づいて DAG を生成する DAG ファクトリを使用してください。
- Secrets Manager でシークレットを管理するには これにより、DAG を作成できるユーザーがシークレットを読み取ることができなくなるわけではありませんが、環境が使用するシークレットをユーザーが変更することはできなくなります。

Apache Airflow ユーザー権限の変更を検出します。

CloudWatch Logs Insights を使用して、Apache Airflow ユーザー権限を変更する DAGs の発生を検出できます。そのためには、 EventBridge スケジュールされたルール、Lambda 関数、および CloudWatch Logs Insights を使用して、DAGs のいずれかが Apache Airflow ユーザー権限を変更するたびに CloudWatch メトリクスに通知を配信できます。

前提条件

このセクションの手順を完了するには、次のものが必要です。

- すべての Apache Airflow ログタイプが INFO ログレベルで有効になっている Amazon MWAA 環境。詳細については、「the section called “Airflow ログの表示”」を参照してください。

Apache Airflow ユーザー権限が変更された際の通知を構成するには

1. 5 つの Amazon MWAA 環境ロググループ (、、、および) に対して次の Logs Insights クエリ文字列を実行する Lambda 関数を作成しますWebServerWorker。 CloudWatch DAGProcessing Scheduler Task

```
fields @log, @timestamp, @message | filter @message like "add-role" | stats count()
 by @log
```

2. 前のステップで作成した Lambda 関数を EventBridge ルールのターゲットとして、スケジュール で実行される ルールを作成します。cron または rate 式を使用して、定期的に実行されるようにスケジュールを構成します。

Apache Airflow 用 Amazon マネージドワークフロー上の Apache Airflow のバージョン

このページでは、Apache Airflow 用 Amazon マネージドワークフローがサポートする Apache Airflow のバージョンと、最新バージョンへのアップグレードに推奨される戦略について説明します。

トピック

Amazon MWAA のバージョンについて

Amazon MWAA は、Apache Airflow リリースを他の一般的なバイナリや Python ライブラリとバンドルしたコンテナイメージを構築します。このイメージは、指定したバージョンの Apache Airflow ベースインストールを使用します。環境を作成する時に、使用するイメージバージョンを指定します。環境が一度作成されると、新しいバージョンにアップグレードするまで、指定されたイメージバージョンが使用され続けます。

最新バージョン

Amazon MWAA は複数の Apache Airflow のバージョンをサポートしています。環境の作成時にイメージバージョンを指定しない場合、Amazon MWAA はサポートされている最新バージョンの Apache Airflow を使用して環境を作成します。

Apache Airflow のバージョン

以下の Apache Airflow バージョンは、Amazon Managed Workflows for Apache Airflow でサポートされています。

Note

- Apache Airflow v2.2.2 以降、Amazon MWAA は Python 要件、プロバイダーパッケージ、カスタムプラグインを Apache Airflow ウェブサーバーに直接インストールすることをサポートしています。
- Apache Airflow v2.7.2 から、要件ファイルには `--constraint` ステートメントを含める必要があります。制約を指定しない場合、要件に記載されているパッケージが使用している Apache Airflow のバージョンと互換性があることを確認するため、Amazon MWAA はお客様に代わって制約を指定します。

  要件ファイルに制約を設定する方法の詳細については、「Python 依存関係のインストール」を参照してください。

Apache Airflow のバージョン	Apache Airflow ガイド	Apache Airflow の制約	Python バージョン
v2.8.1	Apache Airflow v2.8.1 リファレンスガイド	Apache Airflow v2.8.1 制約ファイル	「Python 3.11」
「v2.7.2」	「Apache Airflow v2.7.2 リファレンスガイド」	「Apache Airflow v2.7.2 制約ファイル」	「Python 3.11」
「v2.6.3」	「Apache Airflow v2.6.3 リファレンスガイド」	「Apache Airflow v2.6.3 制約ファイル」	「Python 3.10」
「v2.5.1」	「Apache Airflow v2.5.1 リファレンスガイド」	「Apache Airflow v2.5.1 制約ファイル」	「Python 3.10」
「v2.4.3」	「Apache Airflow v2.4.3 リファレンスガイド」	「Apache Airflow v2.4.3 制約ファイル」	「Python 3.10」

Apache Airflow のバージョン	Apache Airflow ガイド	Apache Airflow の制約	Python バージョン
「v2.2.2」	「Apache Airflow v2.2.2 リファレンスガイド」	「Apache Airflow v2.2.2 制約ファイル」	「Python 3.7」
「v2.0.2」	「Apache Airflow v2.0.2 リファレンスガイド」	「Apache Airflow v2.0.2 制約ファイル」	「Python 3.7」

自己管理型の Apache Airflow デプロイの移行、または既存の Amazon MWAA 環境の移行について、メタデータデータベースのバックアップ手順を含む詳細情報は、「Amazon MWAA 移行ガイド」 を参照してください。

Apache Airflow コンポーネント

このセクションでは、Amazon MWAA の Apache Airflow バージョンごとに利用可能な Apache Airflow スケジューラーとワーカーの数について説明し、各機能をサポートするバージョンを示す Apache Airflow の主要な機能のリストを提供します。

スケジューラ

Apache Airflow のバージョン	スケジューラー (デフォルト)	スケジューラー (最小)	スケジューラー (最大)	
Apache Airflow v2 以上	2	2	5	

ワーカー

Airflow のバージョン	ワーカー (最小)	ワーカー (最大)	ワーカー (デフォルト)	
Apache Airflow v2	1	25	10	

Apache Airflow バージョンのアップグレード

Amazon MWAA はマイナーバージョンアップグレードをサポートしています。つまり、環境をバージョン x.**1**.z から x.**2**.z にアップグレードすることはできますが、新しいメジャーバージョン (たとえば **1**.y.z から **2**.y.z へ) にはアップグレードできません。

> **Note**
>
> ご使用の環境に合わせて Apache Airflow バージョンをダウングレードすることはできません。

詳細情報とワークフローリソースを更新する詳細な手順、および環境を新しいバージョンにアップグレードする方法については、「the section called “エンジンバージョンのアップグレード”」を参照してください。

Apache Airflow の非推奨バージョン

次の表は、Amazon MWAA におけるApache Airflowの非推奨バージョンと、各バージョンの初期リリース日およびサポート終了日を一覧表示します。新しいバージョンへの移行に関する詳細については、「Amazon MWAA 移行ガイド」を参照してください。

Apache Airflow のバージョン	Apache Airflow リリース日	Amazon MWAA 利用開始日	Amazon MWAA 限定サポート日	Amazon MWAA サポート終了日
v1.10.12	2020 年 8 月 25 日	2020 年 11 月 24 日	2023 年 8 月 21 日	2024 年 2 月 21 日

Apache Airflow のバージョン	Apache Airflow リリース日	Amazon MWAA 利用開始日	Amazon MWAA 限定サポート日	Amazon MWAA サポート終了日
v2.0.2	2021 年 4 月 19 日	2021 年 5 月 25 日	2023 年 11 月 23 日	2024 年 4 月 29 日
v2.2.2	2021 年 11 月 15 日	2022 年 1 月 27 日	2024 年 1 月 25 日	2024 年 6 月 27 日

Apache Airflow のバージョンサポートとよくある質問

Apache Airflow コミュニティの「リリースプロセスとバージョンポリシー」に従い、Amazon MWAA は常に Apache Airflow の少なくとも 3 つのマイナーバージョンをサポートすることを約束しています。Apache Airflow における規定のマイナーバージョンのサポート終了日については、サポート終了日の少なくとも 90 日前に発表します。

よくある質問

Q: Amazon MWAA は Apache Airflow バージョンをどのくらいの期間サポートしますか?

A: Amazon MWAA は、Apache Airflow のマイナーバージョンが最初に利用可能になってから最低12ヶ月間サポートします。

Q: Amazon EKS でバージョンのサポートが終了する際には通知が届きますか?

A: はい。アカウント内のいずれかの Amazon MWAA 環境がサポート終了に近いバージョンを実行している場合、Amazon MWAA はサポート終了日 AWS Health Dashboard の を通じて通知を送信します。

Q: 限定サポート日にはどうなりますか?

A: 限定サポート日には、関連付けのバージョンで新しい Amazon MWAA 環境を作成できなくなります。既存の環境は、サポートの終了日まで利用可能です。

Q: サポート終了日にはどうなりますか?

A: サポート終了日には、関連付けられた非推奨バージョンの Apache Airflow を実行する既存の Amazon MWAA 環境に引き続きアクセスできます。Amazon MWAA で Apache Airflow を新しいバージョンにアップグレードする手順については、「Amazon MWAA 移行ガイド」を参照してください。

Important

お客様は、Amazon MWAA バージョンを最新の状態に保つ責任があります。 は、最新のセキュリティ、プライバシー、および可用性保護のメリットを得るために、すべてのお客様に Amazon MWAA 環境を最新バージョンにアップグレードするよう AWS 促します。レガシーバージョン と呼ばれるサポートされていないバージョンまたはソフトウェアで環境を運用している場合、ダウンタイムイベントを含むセキュリティ、プライバシー、運用上のリスクの可能性が高まります。Amazon MWAA 環境をレガシーバージョンで運用することで、これらのリスクを理解し、把握していること、および可能な限り早急に最新バージョンへのアップグレードを完了することに同意したことになります。レガシーバージョンでの環境の継続的な運用には、 AWS サービスの使用に適用される契約が適用されます。
レガシーバージョンは一般公開されていないとみなされ、レガシーバージョンのサポートは AWS 終了しています。その結果、 は、レガシーバージョンが サービス、、 AWSその契約、またはその他の第三者に対してセキュリティまたは責任上のリスク、または損害のリスクをもたらす AWS と判断した場合、いつでもレガシーバージョンへのアクセスまたは使用に制限 AWS がある可能性があります。レガシーバージョンでワークロードを引き続き実行すると、コンテンツが使用できなくなったり、破損したり、回復できなくなったりする可能性があります。レガシーバージョンで実行されている環境は、サービスレベルアグリーメント (SLA) の例外の対象となります。
レガシーバージョンで実行されている環境と関連ソフトウェアには、バグ、エラー、欠陥、および悪意のあるコンポーネントが含まれている可能性があります。したがって、本ライセンス条項または利用規約の義務にかかわらず、 はレガシーバージョンを として AWS 提供します。
の責任共有モデルの詳細については、 AWS Well-Architected フレームワークの AWS「責任共有」を参照してください。

Amazon Managed Workflows for Apache Airflow のサービスエンドポイントとクォータ

Amazon Managed Workflows for Apache Airflow には、以下のサービスクォータとエンドポイントがあります。制限とも呼ばれるサービスクォータは、 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。

目次

サービスエンドポイント

Amazon MWAA のエンドポイントのリストを表示するには、「Apache Airflow エンドポイントとクォータ用の Amazon マネージドワークフロー」を参照してください。

Service Quotas

クォータ名	説明	デフォルトのクォータ	調整可能
環境	リージョンごとに、1 アカウントあたりの Amazon MWAA 環境の最大数。	10	[Yes (はい)]
環境ごとのワーカーの数	Amazon MWAA 環境あたりの最大ワーカー数。	25	はい
環境あたりのウェブサーバー	Amazon MWAA 環境あたりのウェブサーバーの最大数。	5	はい

クォータの増加

「クォータ引き上げリクエスト」を送信することで、調整可能なクォータへの増額をリクエストできます。

Amazon MWAA に関するよくある質問

このページでは、Amazon Managed Workflows for Apache Airflow を使用する際に起こる可能性のある一般的な疑問について説明します。

目次

- PythonVirtualenvOperator は使えますか?
- Amazon MWAA が新しい DAG ファイルを認識するまでどのくらいかかりますか?
- DAG ファイルが Apache Airflow に取り込まれないのはなぜですか?
- 環境から plugins.zip または requirements.txt を削除できますか?
- Apache Airflow v2.0.2 の「管理プラグイン」メニューに自分のプラグインが表示されないのはなぜですか?
- AWS Database Migration Service (DMS) Operators を使用できますか ?

サポートバージョン

Amazon MWAA は Apache Airflow v2 に対して何をサポートしていますか?

Amazon MWAA がサポートしている内容については、「Apache Airflow 用 Amazon マネージドワークフロー上の Apache Airflow のバージョン」を参照してください。

古いバージョンの Apache Airflow がサポートされていないのはなぜですか?

古いバージョンではセキュリティ上の懸念があるため、(発売時点で) 最新の Apache Airflow バージョンの Apache Airflow v1.10.12 のみをサポートしています。

どの Python バージョンを使用する必要がありますか?

以下の Apache Airflow バージョンは、Amazon Managed Workflows for Apache Airflow でサポートされています。

> **Note**
>
> - Apache Airflow v2.2.2 以降、Amazon MWAA は Python 要件、プロバイダーパッケージ、カスタムプラグインを Apache Airflow ウェブサーバーに直接インストールすることをサポートしています。
> - Apache Airflow v2.7.2 から、要件ファイルには `--constraint` ステートメントを含める必要があります。制約を指定しない場合、要件に記載されているパッケージが使用している Apache Airflow のバージョンと互換性があることを確認するため、Amazon MWAA はお客様に代わって制約を指定します。

要件ファイルに制約を設定する方法の詳細については、「Python 依存関係のインストール」を参照してください。

Apache Airflow のバージョン	Apache Airflow ガイド	Apache Airflow の制約	Python バージョン
v2.8.1	Apache Airflow v2.8.1 リファレンスガイド	Apache Airflow v2.8.1 制約ファイル	「Python 3.11」
「v2.7.2」	「Apache Airflow v2.7.2 リファレンスガイド」	「Apache Airflow v2.7.2 制約ファイル」	「Python 3.11」
「v2.6.3」	「Apache Airflow v2.6.3 リファレンスガイド」	「Apache Airflow v2.6.3 制約ファイル」	「Python 3.10」
「v2.5.1」	「Apache Airflow v2.5.1 リファレンスガイド」	「Apache Airflow v2.5.1 制約ファイル」	「Python 3.10」
「v2.4.3」	「Apache Airflow v2.4.3 リファレンスガイド」	「Apache Airflow v2.4.3 制約ファイル」	「Python 3.10」
「v2.2.2」	「Apache Airflow v2.2.2 リファレンスガイド」	「Apache Airflow v2.2.2 制約ファイル」	「Python 3.7」
「v2.0.2」	「Apache Airflow v2.0.2 リファレンスガイド」	「Apache Airflow v2.0.2 制約ファイル」	「Python 3.7」

自己管理型の Apache Airflow デプロイの移行、または既存の Amazon MWAA 環境の移行について、メタデータデータベースのバックアップ手順を含む詳細情報は、「Amazon MWAA 移行ガイド」 を参照してください。

Amazon MWAA はどのバージョンの `pip` を使用していますか?

Apache Airflow v1.10.12 を実行している環境では、Amazon MWAA は `pip` バージョン 21.1.2 をインストールします。

> **Note**
>
> Amazon MWAA は、Apache Airflow v1.10.12 環境では `pip` をアップグレードしません。

Apache Airflow v2 以降の環境では、Amazon MWAA は `pip` バージョン21.3.1 をインストールします。

ユースケース

いつ AWS Step Functions を使用すればよいか Amazon MWAA か

1. Step Functions は 1 件の注文または 100 万件の注文の需要に合わせてスケーリングできるため、Step Functions を使用して個々の顧客の注文を処理できます。
2. 前日の注文を処理する夜間ワークフローを実行している場合は、Step Functions または Amazon MWAA を使用できます。Amazon MWAA では、使用している AWS リソースからワークフローを抽象化するためのオープンソースオプションを使用できます。

環境仕様

各環境で使用できるタスクストレージの容量はどれくらいですか?

タスクストレージは 10 GB に制限されており、Amazon ECS Fargate 1.3 によって指定されています。RAM の容量は指定した環境クラスによって決まります。環境タグ付けの詳細については、「Amazon MWAA 環境クラスの構成」を参照してください。

Amazon MWAA 環境で使用されるデフォルトのオペレーティングシステムは何ですか?

Amazon MWAA 環境は、Amazon Linux AMI を実行するインスタンス上に作成されます。

Amazon MWAA 環境にカスタムイメージを使用することはできますか?

カスタムイメージはサポートされていません。Amazon MWAA は Amazon Linux AMI 上に構築されたイメージを使用します。Amazon MWAA は、環境の Amazon S3 バケットに追加した requirements.txt ファイルに指定されている要件に合わせて`pip3 -r install` を実行することにより、追加の要件をインストールします。

Amazon MWAA HIPAA は準拠していますか?

Amazon は Health Insurance Portability and Accountability Act (HIPAA) に対応しています。HIPAA ビジネス提携契約 (BAA) を で実施している場合は AWS、2022 年 11 月 14 日以降に作成された環境で保護医療情報 (PHI) を処理するワークフローに Amazon MWAA を使用できます。

Amazon MWAA はスポットインスタンスをサポートしていますか?

Amazon MWAA は現在、Apache Airflow 用のオンデマンド Amazon EC2 スポットインスタンスタイプをサポートしていません。ただし、Amazon MWAA 環境では、たとえば Amazon EMR や Amazon EC2 でスポットインスタンスをトリガーできます。

Amazon MWAA はカスタムドメインをサポートしていますか?

Amazon MWAA ホスト名にカスタムドメインを使用できるようにするには、次のいずれかを実行します。

- パブリックウェブサーバーアクセスを使用する Amazon MWAA デプロイでは、Amazon CloudFront と Lambda@Edge を使用してトラフィックを環境に誘導し、カスタムドメイン名をにマッピングできます CloudFront。パブリック環境のカスタムドメインの設定の詳細と例については、Amazon MWAA サンプルリポジトリの「パブリックウェブサーバー用の Amazon MWAA カスタムドメインのサンプル」を参照してください。 GitHub
- プライベートウェブサーバーにアクセスする Amazon MWAA デプロイでは、Application Load Balancer (ALB) を使用してトラフィックを Amazon MWAA に転送し、カスタムドメイン名を ALB にマッピングできます。詳細については、「the section called “Load Balancer の使用 (上級)”」を参照してください。

自分の環境に SSH で接続できますか?

Amazon MWAA 環境ではSSHはサポートされていませんが、`BashOperator` を使用してDAG を作成し、bashコマンドを実行することが可能です。例:

```
from airflow import DAG
from airflow.operators.bash_operator import BashOperator
from airflow.utils.dates import days_ago
with DAG(dag_id="any_bash_command_dag", schedule_interval=None, catchup=False,
 start_date=days_ago(1)) as dag:
      cli_command = BashOperator(
          task_id="bash_command",
          bash_command="{{ dag_run.conf['command'] }}"
      )
```

Apache Airflow UI で DAG をトリガーするには、以下を使用してください。

```
{ "command" : "your bash command"}
```

VPC セキュリティグループに自己参照ルールが必要なのはなぜですか?

自己参照ルールを作成することで、ソースを VPC 内の同じセキュリティグループに制限することができ、ネットワーク全体には公開されません。詳細については、「the section called “VPC におけるセキュリティ”」を参照してください。

IAM の異なるグループの環境を非表示にすることはできますか?

で環境名を指定することでアクセスを制限できます AWS Identity and Access Managementが、可視性フィルタリングは AWS コンソールでは利用できません。ユーザーが 1 つの環境を表示できる場合、すべての環境を表示できます。

一時データを Apache Airflow ワーカーに保存できますか?

Apache Airflow オペレータはワーカーに一時データを保存できます。Apache Airflow ワーカーは、お客様の環境の Fargate コンテナにある `/tmp` の一時ファイルにアクセスできます。

> **Note**
>
> Amazon ECS Fargate1.3 によると、タスクストレージの合計は 10 GB に制限されています。後続のタスクが同じ Fargate コンテナインスタンスで実行される保証はありません。この Fargate コンテナインスタンスは別の `/tmp` フォルダを使用する可能性があります。

25 個以上の Apache Airflow ワーカーを指定できますか?

はい。Amazon MWAA コンソールでは最大 25 の Apache Airflow ワーカーを指定できますが、クォータの引き上げをリクエストすることで、1 つの環境に最大 50 のワーカーを設定できます。詳細については、「Requesting a quota increase」(クォータ引き上げのリクエスト) を参照してください。

Amazon MWAA は共有の Amazon VPC をサポートしていますか、それとも共有サブネットをサポートしていますか?

Amazon MWAA は共有の Amazon VPC または共有サブネットをサポートしていません。環境を作成するときに選択する Amazon VPC は、環境を作成しようとしているアカウントが所有している必要があります。ただし、Amazon MWAA アカウント内の Amazon VPC から共有 VPC にトラフィックをルーティングすることはできます。詳細および共有 Amazon VPC へのトラフィックのルーティングの例については、「Amazon VPC トランジットゲートウェイガイド」の「インターネットへの集中型アウトバウンドルーティング」を参照してください。

メトリクス

ワーカーをスケーリングするかどうかの判断にはどのようなメトリックスが使用されますか?

Amazon MWAA は QueuedTasksおよび RunningTasksをモニタリング CloudWatch して、Apache Airflow ワーカーを環境でスケーリングするかどうかを決定します。詳細については、「モニタリングおよびメトリクス」を参照してください。

でカスタムメトリクスを作成できます CloudWatchか ?

CloudWatch コンソールには表示されません。ただし、 でカスタムメトリクスを書き込む DAG を作成できます CloudWatch。詳細については、「the section called “DAG を使用してカスタムメトリクスを記述する”」を参照してください。

DAG、オペレータ、接続、その他の質問

`PythonVirtualenvOperator` は使えますか?

Amazon MWAA では`PythonVirtualenvOperator`は明示的にサポートされていませんが、`PythonVirtualenvOperator` を使用するカスタムプラグインを作成することができます。サ

ンプルコードについては、「the section called “Python 仮想環境オペレータにパッチを適用するカスタムプラグイン ”」を参照してください。

Amazon MWAA が新しい DAG ファイルを認識するまでどのくらいかかりますか?

DAG は、Amazon S3 バケットからお客様の環境に定期的に同期されます。新しい DAG ファイルを追加した場合、Amazon MWAA が新しいファイルの使用を開始するまでに約 300 秒かかります。既存の DAG を更新する場合、Amazon MWAA が更新を認識するまでに約 30 秒かかります。

新しい DAG に対しては 300秒、既存の DAG の更新に対しては 30秒、というこれらの値は、それぞれ Apache Airflow の構成オプション `dag_dir_list_interval` および `min_file_process_interval` に対応しています。

DAG ファイルが Apache Airflow に取り込まれないのはなぜですか?

このセクションでは、以下の問題の発生時に考えられる対策について説明します。

1. 実行ロールに Amazon S3 バケットに対する十分なアクセス権限があることを確認します。詳細については、「Amazon MWAA 実行ロール」を参照してください。
2. Amazon S3 バケットに Block Public Access が設定され、バージョニングが有効になっていることを確認します。詳細については、「Amazon MWAA 用の Amazon S3 バケットの作成」を参照してください。
3. DAG ファイル自体を確認します。たとえば、各 DAG に固有の DAG ID があることを確認してください。

環境から **plugins.zip** または **requirements.txt** を削除できますか?

現在、plugins.zip や requirements.txt を追加したら環境から削除する方法はありませんが、現在対応中です。当面の回避策は、それぞれ空のテキストまたは zip ファイルを指すことです。詳細については、「Amazon S3 でファイルの削除」を参照してください。

Apache Airflow v2.0.2 の「管理プラグイン」メニューに自分のプラグインが表示されないのはなぜですか?

セキュリティ上の理由から、Amazon MWAA 上の Apache Airflow ウェブサーバーのネットワーク出力は制限されており、バージョン 2.0.2 環境の Apache Airflow ウェブサーバーにはプラグインや

Python の依存関係を直接インストールしません。図に示すプラグインにより、Amazon MWAA は AWS Identity and Access Management (IAM) で Apache Airflow ユーザーを認証できます。

プラグインと Python の依存関係をウェブサーバーに直接インストールできるようにするには、Apache Airflow v2.2 以降で新しい環境を作成することをお勧めします。Amazon MWAA は、Apache Airflow v2.2 以降では Python の依存関係とカスタムプラグインをウェブサーバーに直接インストールします。

AWS Database Migration Service (DMS) Operators を使用できますか？

Amazon MWAA は DMS オペレーターをサポートしています。ただし、このオペレータを使用して Amazon MWAA 環境に関連付けられた Amazon Aurora PostgreSQL メタデータデータベースに対してアクションを実行することはできません。

Amazon Managed Workflows for Apache Airflow のトラブルシューティング

このトピックでは、Apache Airflow 用 Amazon マネージドワークフローで Apache Airflow を使用する際に発生する可能性のある一般的な問題とエラー、およびこれらのエラーを解決するための推奨手順について説明します。

目次

- 「log_group: airflow-*{*environmentName}-Task log_stream:* {*DAG_ID}/*{*TASK_ID}/*{*time}/*{*n}.log からリモートログを読み込めませんでした」と表示されました。自分のタスクログ

トラブルシューティング Apache Airflow v2 における DAG、オペレータ、接続、その他の問題

このページのトピックでは、Apache Airflow v2 Python の依存関係、カスタムプラグイン、DAGs、オペレータ、接続、タスク、およびウェブサーバが Amazon でホストされている Apache Airflow ワークフロー環境で発生する可能性のある問題の解決方法について説明します。

目次

接続

以下のトピックでは、Apache Airflow の接続を使用する際や別の AWS データベースを使用する際に受け取る可能性のあるエラーについて説明しています。

Secrets Manager に接続できません。

次のステップを推奨します。

1. Apache Airflow 接続と変数のシークレットキーを作成する方法を「the section called “Secrets Manager の設定”」 で学習できます。
2. `test-variable` で ApacheAirflow 変数 (Apache Airflow 変数の AWS Secrets Manager におけるシークレットキーの使用) のシークレットキーを使用する方法を学習します。
3. AWS Secrets Manager の Apache Airflow 接続でのシークレットキーの使用 で ApacheAirflow 接続 (`myconn`) のシークレットキーを使用する方法を学習します。

`secretsmanager:ResourceTag/<tag-key>` シークレットマネージャの条件またはリソース制限は、実行ロールポリシーでどのように設定しますか。

> **Note**
>
> Apache Airflow バージョン 2.0 やそれ以前のバージョンに適用されます。

現時点では、Apache Airflow の既知の問題のため、環境の実行ロールで条件キーやその他のリソース制限を使用して Secrets Manager secrets へのアクセスを制限することはできません。

Snowflake に接続できません。

次のステップを推奨します。

1. GitHub の「aws-mwaa-local-runner」を使用して、DAG、カスタムプラグイン、Python の依存関係をローカルでテストします。
2. ご使用の環境に適した requirements.txt に次のエントリを追加します。

   ```
   apache-airflow-providers-snowflake==1.3.0
   ```

3. 以下のインポートを DAG に追加する：

```
from airflow.providers.snowflake.operators.snowflake import SnowflakeOperator
```

Apache Airflow 接続オブジェクトに、次のキーバリューのペアが含まれていることを確認します。

1. 接続 ID: snowflake_conn
2. コーンタイプ: スノーフレーク
3. ホスト： <my account> <my region if not us-west-2>.snowflakecomputing.com
4. スキーマ: <my schema>
5. ログイン： <my user name>
6. パスワード： ********
7. ポート: <port, if any>
8. エキストラ：

```
{
    "account": "<my account>",
    "warehouse": "<my warehouse>",
    "database": "<my database>",
    "region": "<my region if not using us-west-2 otherwise omit this line>"
}
```

例:

```
>>> import json
>>> from airflow.models.connection import Connection
>>> myconn = Connection(
...    conn_id='snowflake_conn',
...    conn_type='Snowflake',
...    host='YOUR_ACCOUNT.YOUR_REGION.snowflakecomputing.com',
...    schema='YOUR_SCHEMA'
...    login='YOUR_USERNAME',
...    password='YOUR_PASSWORD',
...    port='YOUR_PORT'
...    extra=json.dumps(dict(account='YOUR_ACCOUNT', warehouse='YOUR_WAREHOUSE',
 database='YOUR_DB_OPTION', region='YOUR_REGION')),
... )
```

Airflow UI　に接続が表示されない

Apache Airflow は Apache Airflow UI に接続テンプレートを用意しています。接続タイプに関係なく、接続 URI 文字列を生成します。Apache Airflow UI に接続テンプレートがない場合は、HTTP 接続テンプレートを使用するなど、代替の接続テンプレートを使用して接続 URI 文字列を生成できます。

次のステップを推奨します。

1. Amazon MWAA が Apache Airflow UI で提供している接続タイプについては、「Amazon MWAA 環境にインストールされている Apache エアフロープロバイダーパッケージ」を参照してください。
2. Apache Airflow CLI コマンドリファレンス のCLI でApache Airflow接続を作成するコマンドを確認します。
3. 接続タイプの概要 Amazon MWAA上のApache Airflow UI では使用できない接続タイプのために、Apache Airflow UI で接続テンプレートを交換して使用する方法を学習します。

ウェブサーバ

次のトピックでは、Amazon MWAA 上の Apache Airflow ウェブサーバーで発生する可能性のある エラーについて説明します。

ウェブサーバーにアクセスすると、5xx エラーが表示されます。

次のステップを推奨します。

1. Apache Airflow 構成オプションを確認してください。Apache Airflow 構成オプションとして指定したキーと値のペア（例： AWS Secrets Manager ）が正しく設定されていることを確認してください。詳細については、「the section called “Secrets Manager に接続できません。”」を参照してください。
2. `requirements.txt` をチェックしてください。Apache Airflow バージョンと互換性のある、`requirements.txt` にリストされている Airflow の「extras」パッケージおよびその他のライブラリを確認してください。
3. `requirements.txt` ファイルに Python の依存関係を指定する方法については、「requirements.txt での Python 依存関係の管理」を参照してください。

「スケジューラーは実行されていないようです」というエラー表示があります。

スケジューラが実行されていないように見える場合や、最後の「ハートビート」が数時間前に受信された場合は、DAGs が Apache Airflow に表示されず、新しいタスクがスケジューリングされない可能性があります。

次のステップを推奨します。

1. VPC セキュリティグループがポート 5432 へのインバウンドアクセスを許可していることを確認します。これは、ご使用の環境の Amazon Aurora PostgreSQL メタデータデータベースに接続するために必要のポートです。このルールを追加したら、 Amazon MWAA を数分間待つとエラーは消えるはずです。詳細については、「the section called “VPC におけるセキュリティ”」を参照してください。

   > **Note**
   >
   > - Aurora PostgreSQL メタデータベースは「Amazon MWAAサービスアーキテクチャ」の一部であり、AWS アカウント では表示されません。
   > - データベース関連のエラーは通常、スケジューラー障害の症状であり、根本的な原因ではありません。

2. スケジューラーが動作していない場合は、「依存関係のインストールの失敗」や「スケジューラーの過負荷」など、さまざまな要因が原因である可能性があります。CloudWatch Logs で対応するロググループを表示して、DAGs、プラグイン、および要件が正しく機能していることを確認します。詳細については、「モニタリングおよびメトリクス」を参照してください。

タスク

次のトピックでは、Apache Airflow ログを表示した際に発生する可能性があるエラーについて説明します。

タスクが行き詰まっているか、完了していません。

Apache Airflow タスクが「行き詰まっている」か、完了していない場合は、次のステップを実行することをお勧めします。

1. 多数の DAG が定義されている可能性があります。DAG の数を減らし、環境の更新 (ログレベルの変更など) を実行して強制的にリセットしてください。

a. Airflow は DAG が有効かどうかに関係なく解析します。環境の 50% を超える容量を使用していると、Apache Airflow スケジューラに負荷がかかり始める可能性があります。これにより、CloudWatch メトリクスの合計解析時間が長くなったり、CloudWatCloudWatch Logs の DAG 処理時間が長くなることがあります。Apache Airflow の設定を最適化する方法は他にもありますが、このガイドの対象範囲には含まれていません。

b. ご使用の環境のパフォーマンスを調整するために推奨するベスト・プラクティスの詳細については、「the section called “Apache Airflow のパフォーマンス調整”」 を参照してください。

2. キューには、多数のタスクがある可能性があります。これは通常、「なし」状態のタスクが多数発生しているか、CloudWatch でキューに入れられたタスクや保留中のタスクが多数発生している場合に発生します。これは以下のような原因で起こりうる：

a. 実行するタスクの数が環境の実行能力を超えている場合や、自動スケーリング前にキューに入れられたタスクの数が多い場合は、タスクを検出して追加のワーカー.をデプロイする時間があります。

b. 実行するタスクの数が、実行可能な環境よりも多い場合は、DAG が同時に実行するタスクの数を減らすか、Apache Airflow ワーカーの最小数を増やすことをお勧めします。

c. 自動スケーリングで追加のワーカーを検出してデプロイする時間ができる前に、大量のタスクがキューに入れられている場合は、タスクの配備をずらすか、最小 Apache Airflow ワーカスレッドを増やすことをお勧めします。

d. AWS Command Line Interface (AWS CLI) の 「update-environment」 コマンドを使用して、環境で実行されるワーカーの最小数または最大数を変更できます。

```
aws mwaa update-environment --name MyEnvironmentName --min-workers 2 --max-
workers 10
```

e. ご使用の環境のパフォーマンスを調整するために推奨するベスト・プラクティスの詳細については、「the section called “Apache Airflow のパフォーマンス調整”」 を参照してください。

3. タスクログとして表示され、それ以上の指示なしに Apache Airflow で停止したタスクが、実行中に削除された可能性があります。これは以下のような原因で起こりうる：

a. もし、1) 現在のタスクが現在の環境のキャパシティを超え、2) 数分間、タスクが実行されなかったり、キューに入れられなかったりした後、3) 新しいタスクがキューに入れられたりする瞬間があります。

b. Amazon MWAA 自動スケーリングは、最初のシナリオに応じてワーカーを追加します。2 番目のシナリオでは、追加のワーカーが削除されます。キューに入っているタスクの中には、ワーカーが削除される過程にあり、コンテナが削除された時点で終了するものもあります。

c. 環境内の最小ワーカー数を増やすことをお勧めします。もう 1 つの選択肢は、DAG とタスクの時間を調整して、これらの状況が発生しないようにすることです

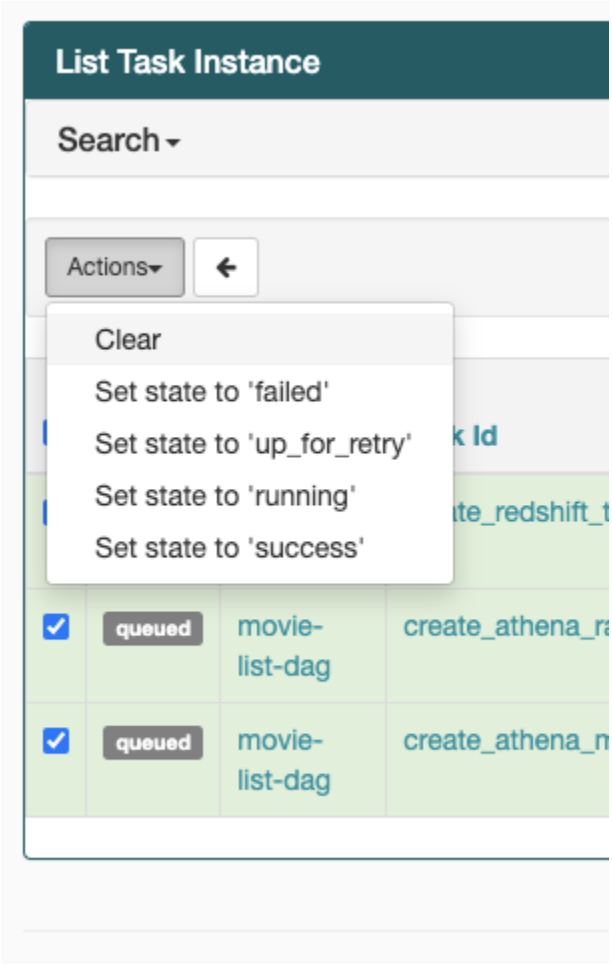
d. また、最小ワーカー数を環境内の最大ワーカー数と同じようにに設定して、オートスケーリングを効果的に無効にすることもできます。AWS Command Line Interface (AWS CLI) の「update-environment」コマンドを使用して、ワーカーの最小数と最大数を同じに設定して自動スケーリングを無効にします。

```
aws mwaa update-environment --name MyEnvironmentName --min-workers 5 --max-
workers 5
```

e. ご使用の環境のパフォーマンスを調整するために推奨するベスト・プラクティスの詳細については、「the section called “Apache Airflow のパフォーマンス調整”」を参照してください。

4. タスクが「実行中」状態のままになっている場合は、タスクをクリアしたり、成功または失敗のマークを付けることもできます。これにより、環境の自動スケーリングコンポーネントは、環境で実行されているワーカー数をスケールダウンできます。次の図は、取り残されたタスクの例です。

- 取り残されたタスクの円を選択し、クリアを選択します (図を参照) 。これにより Amazon MWAA はワーカーをスケールダウンできます。そうしない場合、Amazon MWAA はどの DAG が有効か無効かを判断できず、キューにタスクが残っている場合はスケールダウンできません。



5. Apache Airflow タスクライフサイクルの詳細については、Apache Airflow リファレンスガイドの「概念」を参照してください。

CLI

次のトピックでは、AWS Command Line Interface で Airflow CLI コマンドを実行したときに発生する可能性があるエラーについて説明します。

CLI で DAG をトリガーすると「503」エラーが表示されます

Airflow CLI は Apache Airflow ウェブサーバー上でで実行され、同時実行性が制限されています。通常、 CLI コマンドを最大 4 つ同時に実行できます。

`dags backfill` Apache エアフロー CLI コマンドが失敗した原因は？ 回避策はありますか？

> **Note**
>
> 以下は Apache Airflow v2.0.2 の環境にのみ適用されます。

他の Apache Airflow CLI コマンドと同様に、`backfill` コマンドはすべての DAGs を処理する前に、CLI 操作がどの DAGs に適用されているかに関係なく、すべての DAGs をローカルに解決します。Apache Airflow v2.0.2 を使用する Amazon MWAA 環境では、CLI コマンドの実行時にプラグインと要件がウェブサーバにインストールされていないため、解決操作が失敗し、`backfill` 操作が呼び出されません。環境に要件やプラグインがない場合、`backfill` 操作は成功します。

`backfill` CLI コマンドを実行できるようにするには、bash 演算子で呼び出すことをお勧めします。bash オペレータでは、`backfill` はワーカーから起動されます。これにより、必要な要件とプラグインがすべて使用可能になり、インストール後、 DAG が正常に解析できるようになります。次の例は、`BashOperator` を実行するための `backfill` 付き DAG を作成する方法を示しています。

```
from airflow import DAG
from airflow.operators.bash_operator import BashOperator
from airflow.utils.dates import days_ago

with DAG(dag_id="backfill_dag", schedule_interval=None, catchup=False,
 start_date=days_ago(1)) as dag:
    cli_command = BashOperator(
        task_id="bash_command",
        bash_command="airflow dags backfill my_dag_id"
    )
```

演算子

次のトピックでは、Operators を使用するときに受け取る可能性のあるエラーについて説明します。

S3-Transform オペレータの使用中に `PermissionError: [Errno 13] Permission denied` エラーが発生しました。

S3Transform オペレータを使用してシェルスクリプトを実行しようとして `PermissionError: [Errno 13] Permission denied` エラーが発生する場合は、次の手順を実行することをお勧

めします。次のステップは、既存の plugins.zip ファイルがあることを前提としています。新しい plugins.zip を作成する場合は、「カスタムプラグインのインストール」 を参照してください。

1. GitHub の「aws-mwaa-local-runner」を使用して、DAG、カスタムプラグイン、Python の依存関係をローカルでテストします。
2. 「変換」スクリプトを作成します。

```
#!/bin/bash
cp $1 $2
```

3. (オプション) macOS と Linux ユーザーは、次のコマンドを実行して、スクリプトが実行可能であることを確認しなければならないことがあります。

```
chmod 777 transform_test.sh
```

4. スクリプトを plugins.zip に追加します。

```
zip plugins.zip transform_test.sh
```

5. 「plugins.zip を Amazon S3 にアップロードする」 のステップに従ってください。
6. 「Amazon MWAA コンソールのplugins.zip バージョンの指定」 のステップに従ってください。
7. 以下の DAG を作成します。

```
from airflow import DAG
from airflow.providers.amazon.aws.operators.s3_file_transform import
 S3FileTransformOperator
from airflow.utils.dates import days_ago
import os

DAG_ID = os.path.basename(__file__).replace(".py", "")

with DAG (dag_id=DAG_ID, schedule_interval=None, catchup=False,
 start_date=days_ago(1)) as dag:
    file_transform = S3FileTransformOperator(
        task_id='file_transform',
        transform_script='/usr/local/airflow/plugins/transform_test.sh',
        source_s3_key='s3://YOUR_S3_BUCKET/files/input.txt',
        dest_s3_key='s3://YOUR_S3_BUCKET/files/output.txt'
    )
```

8. 「Amazon S3 への DAG コードのアップロード」 のステップに従ってください。

トラブルシューティング:Apache Airflow v1 の DAGs、オペレータ、接続、およびその他の問題

このページのトピックには、Amazon Managed Workflows for Apache Airflow環境で遭遇する可能性があるApache Airflow v1.10.12のPython依存関係、カスタムプラグイン、DAG、オペレータ、接続、タスク、およびウェブサーバーの問題に対する解決策が含まれています。

目次

requirements.txt の更新

以下のトピックでは、`requirements.txt` を更新する際に受け取る可能性のあるエラーについて説明しています。

apache-airflow-providers-amazon を追加すると、環境が失敗します。

`apache-airflow-providers-`**`xyz`** はApache Airflow v2とのみ互換性があり、`apache-airflow-backport-providers-`**`xyz`** はApache Airflow 1.10.12と互換性があります。

Broken DAG

次のトピックでは、DAGs の実行時に発生する可能性があるエラーについて説明します。

Amazon DynamoDB オペレーターを使用しているときに「DAG が壊れました」というエラーが表示されました

次のステップを推奨します。

1. GitHub の「aws-mwaa-local-runner」を使用して、DAG、カスタムプラグイン、Python の依存関係をローカルでテストします。
2. 以下のパッケージを `requirements.txt` に追加します。

   ```
   boto
   ```

3. `requirements.txt` ファイルに Python の依存関係を指定する方法については、「requirements.txt での Python 依存関係の管理」を参照してください。

「壊れた DAG：psycopg2 という名前のモジュールはありません」というエラーが表示されました。

次のステップを推奨します。

1. GitHub の「aws-mwaa-local-runner」を使用して、DAG、カスタムプラグイン、Python の依存関係をローカルでテストします。
2. `requirements.txt` Apache Airflowのバージョンに次の項目を追加します。例:

```
apache-airflow[postgres]==1.10.12
```

3. requirements.txt ファイルに Python の依存関係を指定する方法については、「requirements.txt での Python 依存関係の管理」を参照してください。

Slack オペレータを使用しているときに「DAG が壊れました」というエラーが表示されました。

次のステップを推奨します。

1. GitHub の「aws-mwaa-local-runner」を使用して、DAG、カスタムプラグイン、Python の依存関係をローカルでテストします。
2. 以下のパッケージを requirements.txt に追加し、Apache Airflowのバージョンを指定してください。例:

```
apache-airflow[slack]==1.10.12
```

3. requirements.txt ファイルに Python の依存関係を指定する方法については、「requirements.txt での Python 依存関係の管理」を参照してください。

Google/GCP/BigQuery のインストール中に、さまざまなエラーが発生しました。

Amazon MWAA は Amazon Linux を使用しているため、特定のバージョンの Cython と暗号化ライブラリが必要です。次のステップを推奨します。

1. GitHub の「aws-mwaa-local-runner」を使用して、DAG、カスタムプラグイン、Python の依存関係をローカルでテストします。
2. 以下のパッケージを requirements.txt に追加します。

```
grpcio==1.27.2
cython==0.29.21
pandas-gbq==0.13.3
cryptography==3.3.2
apache-airflow-backport-providers-amazon[google]
```

3. リバース・ポート・プロバイダを使用しない場合は、次の機能を使用できます :

```
grpcio==1.27.2
```

```
cython==0.29.21
pandas-gbq==0.13.3
cryptography==3.3.2
apache-airflow[gcp]==1.10.12
```

4. `requirements.txt` ファイルに Python の依存関係を指定する方法については、「requirements.txt での Python 依存関係の管理」を参照してください。

「DAG が壊れました：Cython という名前のモジュールはありません」というエラーが表示されました。

Amazon MWAA は、特定のバージョンの Cython を必要とする Amazon Linux を使用しています。次のステップを推奨します。

1. GitHub の「aws-mwaa-local-runner」を使用して、DAG、カスタムプラグイン、Python の依存関係をローカルでテストします。
2. 以下のパッケージを `requirements.txt` に追加します。

```
cython==0.29.21
```

3. Cython ライブラリには、必要なさまざまな pip 依存バージョンがあります。例えば、`awswrangler==2.4.0` を使用するためには `pyarrow<3.1.0,>=2.0.0` が必要ですので、pip3が `pyarrow==3.0.0` をインストールしようとして Broken DAG エラーが発生します。許容できる、最も古いバージョンを明示的に指定することをおすすめします。例えば、最小値 `pyarrow==2.0.0` を `awswrangler==2.4.0` よりも前に指定すると、エラーは解消され、`requirements.txt` が正しくインストールされます。最終的な要件は、以下のようになります。

```
cython==0.29.21
pyarrow==2.0.0
awswrangler==2.4.0
```

4. `requirements.txt` ファイルに Python の依存関係を指定する方法については、「requirements.txt での Python 依存関係の管理」を参照してください。

演算子

次のトピックでは、Operators を使用するときに受け取る可能性のあるエラーについて説明します。

BigQuery オペレータの使用中にエラーが発生しました。

Amazon MWAA は UI 拡張機能があるオペレータをサポートしていません。次のステップを推奨します。

1. GitHub の「aws-mwaa-local-runner」を使用して、DAG、カスタムプラグイン、Python の依存関係をローカルでテストします。
2. 回避策は、問題のオペレータをインポートした後に <operator name>.operator_extra_links = None を設定するために、DAGに行を追加して拡張機能を上書きすることです。例:

```
from airflow.contrib.operators.bigquery_operator import BigQueryOperator
BigQueryOperator.operator_extra_links = None
```

3. 上記をプラグインに追加することで、このアプローチをすべての DAG に使用できます。例については、「the section called “Python 仮想環境オペレータにパッチを適用するカスタムプラグイン ”」を参照してください。

接続

以下のトピックでは、Apache Airflow の接続を使用する際や別の AWS データベースを使用する際に受け取る可能性のあるエラーについて説明しています。

Snowflake に接続できません。

次のステップを推奨します。

1. GitHub の「aws-mwaa-local-runner」を使用して、DAG、カスタムプラグイン、Python の依存関係をローカルでテストします。
2. ご使用の環境に適した requirements.txt に次のエントリを追加します。

```
asn1crypto == 0.24.0
snowflake-connector-python == 1.7.2
```

3. 以下のインポートを DAG に追加する：

```
from airflow.contrib.hooks.snowflake_hook import SnowflakeHook
from airflow.contrib.operators.snowflake_operator import SnowflakeOperator
```

Apache Airflow 接続オブジェクトに、次のキーバリューのペアが含まれていることを確認します。

1. 接続 ID: snowflake_conn
2. コーンタイプ: スノーフレーク
3. ホスト： <my account> <my region if not us-west-2>.snowflakecomputing.com
4. スキーマ: <my schema>
5. ログイン： <my user name>
6. パスワード： ********
7. ポート: <port, if any>
8. エキストラ：

```
{
    "account": "<my account>",
    "warehouse": "<my warehouse>",
    "database": "<my database>",
    "region": "<my region if not using us-west-2 otherwise omit this line>"
}
```

例:

```
>>> import json
>>> from airflow.models.connection import Connection
>>> myconn = Connection(
...    conn_id='snowflake_conn',
...    conn_type='Snowflake',
...    host='YOUR_ACCOUNT.YOUR_REGION.snowflakecomputing.com',
...    schema='YOUR_SCHEMA'
...    login='YOUR_USERNAME',
...    password='YOUR_PASSWORD',
...    port='YOUR_PORT'
...    extra=json.dumps(dict(account='YOUR_ACCOUNT', warehouse='YOUR_WAREHOUSE',
 database='YOUR_DB_OPTION', region='YOUR_REGION')),
... )
```

Secrets Manager に接続できません。

次のステップを推奨します。

1. Apache Airflow 接続と変数のシークレットキーを作成する方法を「the section called “Secrets Manager の設定”」 で学習できます。
2. `test-variable` で ApacheAirflow 変数 (Apache Airflow 変数の AWS Secrets Manager におけるシークレットキーの使用) のシークレットキーを使用する方法を学習します。
3. AWS Secrets Manager の Apache Airflow 接続でのシークレットキーの使用 で ApacheAirflow 接続 (`myconn`) のシークレットキーを使用する方法を学習します。

「<DB-identifier-name>.cluster-id.rds.amazonaws.com」 の MySQL サーバに接続できません。

Amazon MWAA のセキュリティグループと RDS セキュリティグループには、トラフィックが相互に行き交うことを可能にする入口が必要です。次のステップを推奨します。

1. Amazon MWAA の VPC セキュリティグループからのトラフィックをすべて許可するように RDS セキュリティグループを変更します。
2. Amazon MWAA の VPC セキュリティグループを変更し、RDS セキュリティグループからのすべてのトラフィックを許可します。
3. タスクを再実行し、CloudWatch Logs の Apache Airflow ログをチェックして、 SQL クエリが成功したかどうかを確認します。

ウェブサーバ

次のトピックでは、Amazon MWAA 上の Apache Airflow ウェブサーバーで発生する可能性のあるエラーについて説明します。

BigQueryOperator を使用しており、そのせいで web サーバーがクラッシュした。

次のステップを推奨します。

1. `BigQueryOperator` や `QuboleOperator` などの `operator_extra_links` を含む Apache Airflow オペレータを使用すると、Apache Airflow ウェブサーバーがクラッシュすることがあります。これらのオペレータは Web サーバにコードをロードしようとしますが、これはセキュリティ上の理由から許可されていません。インポートステートメント後に次のコードを追加して、DAG 内のオペレータにパッチを適用することをお勧めします。

```
BigQueryOperator.operator_extra_links = None
```

2. GitHub の「aws-mwaa-local-runner」を使用して、DAG、カスタムプラグイン、Python の依存関係をローカルでテストします。

ウェブサーバーにアクセスすると、5xx エラーが表示されます。

次のステップを推奨します。

1. Apache Airflow 構成オプションを確認してください。Apache Airflow 構成オプションとして指定したキーと値のペア (例： AWS Secrets Manager) が正しく設定されていることを確認してください。詳細については、「the section called “Secrets Manager に接続できません。”」を参照してください。
2. `requirements.txt` をチェックしてください。Apache Airflow バージョンと互換性のある、`requirements.txt` にリストされている Airflow の「extras」パッケージおよびその他のライブラリを確認してください。
3. `requirements.txt` ファイルに Python の依存関係を指定する方法については、「requirements.txt での Python 依存関係の管理」を参照してください。

「スケジューラーは実行されていないようです」というエラー表示があります。

スケジューラが実行されていないように見える場合や、最後の「ハートビート」が数時間前に受信された場合は、DAGs が Apache Airflow に表示されず、新しいタスクがスケジューリングされない可能性があります。

次のステップを推奨します。

1. VPC セキュリティグループがポート 5432 へのインバウンドアクセスを許可していることを確認します。これは、ご使用の環境の Amazon Aurora PostgreSQL メタデータデータベースに接続するために必要のポートです。このルールを追加したら、 Amazon MWAA を数分間待つとエラーは消えるはずです。詳細については、「the section called “VPC におけるセキュリティ”」を参照してください。

> **Note**
>
> - Aurora PostgreSQL メタデータベースは 「Amazon MWAAサービスアーキテクチャ」の一部であり、AWS アカウント では表示されません。

- データベース関連のエラーは通常、スケジューラー障害の症状であり、根本的な原因ではありません。

2. スケジューラーが動作していない場合は、「依存関係のインストールの失敗」や「スケジューラーの過負荷」など、さまざまな要因が原因である可能性があります。CloudWatch Logs で対応するロググループを表示して、DAGs、プラグイン、および要件が正しく機能していることを確認します。詳細については、「モニタリングおよびメトリクス」を参照してください。

タスク

次のトピックでは、Apache Airflow ログを表示した際に発生する可能性があるエラーについて説明します。

タスクが行き詰まっているか、完了していません。

Apache Airflow タスクが「行き詰まっている」か、完了していない場合は、次のステップを実行することをお勧めします。

1. 多数の DAG が定義されている可能性があります。DAG の数を減らし、環境の更新 (ログレベルの変更など) を実行して強制的にリセットしてください。

   a. Airflow は DAG が有効かどうかに関係なく解析します。環境の 50% を超える容量を使用していると、Apache Airflow スケジューラに負荷がかかり始める可能性があります。これにより、CloudWatch メトリクスの合計解析時間が長くなったり、CloudWatCloudWatch Logs の DAG 処理時間が長くなることがあります。Apache Airflow の設定を最適化する方法は他にもありますが、このガイドの対象範囲には含まれていません。

   b. ご使用の環境のパフォーマンスを調整するために推奨するベスト・プラクティスの詳細については、「the section called “Apache Airflow のパフォーマンス調整”」を参照してください。

2. キューには、多数のタスクがある可能性があります。これは通常、「なし」状態のタスクが多数発生しているか、CloudWatch でキューに入れられたタスクや保留中のタスクが多数発生している場合に発生します。これは以下のような原因で起こりうる：

   a. 実行するタスクの数が環境の実行能力を超えている場合や、自動スケーリング前にキューに入れられたタスクの数が多い場合は、タスクを検出して追加のワーカー.をデプロイする時間があります。

b. 実行するタスクの数が、実行可能な環境よりも多い場合は、DAG が同時に実行するタスクの数を減らすか、Apache Airflow ワーカーの最小数を増やすことをお勧めします。

c. 自動スケーリングで追加のワーカーを検出してデプロイする時間ができる前に、大量のタスクがキューに入れられている場合は、タスクの配備をずらすか、最小 Apache Airflow ワーカスレッドを増やすことをお勧めします。

d. AWS Command Line Interface (AWS CLI) の「update-environment」コマンドを使用して、環境で実行されるワーカーの最小数または最大数を変更できます。

```
aws mwaa update-environment --name MyEnvironmentName --min-workers 2 --max-
workers 10
```

e. ご使用の環境のパフォーマンスを調整するために推奨するベスト・プラクティスの詳細については、「the section called “Apache Airflow のパフォーマンス調整”」を参照してください。

3. タスクログとして表示され、それ以上の指示なしに Apache Airflow で停止したタスクが、実行中に削除された可能性があります。これは以下のような原因で起こりうる：

a. もし、1) 現在のタスクが現在の環境のキャパシティを超え、2) 数分間、タスクが実行されなかったり、キューに入れられなかったりした後、3) 新しいタスクがキューに入れられたりする瞬間があります。

b. Amazon MWAA 自動スケーリングは、最初のシナリオに応じてワーカーを追加します。2 番目のシナリオでは、追加のワーカーが削除されます。キューに入っているタスクの中には、ワーカーが削除される過程にあり、コンテナが削除された時点で終了するものもあります。

c. 環境内の最小ワーカー数を増やすことをお勧めします。もう 1 つの選択肢は、DAG とタスクの時間を調整して、これらの状況が発生しないようにすることです

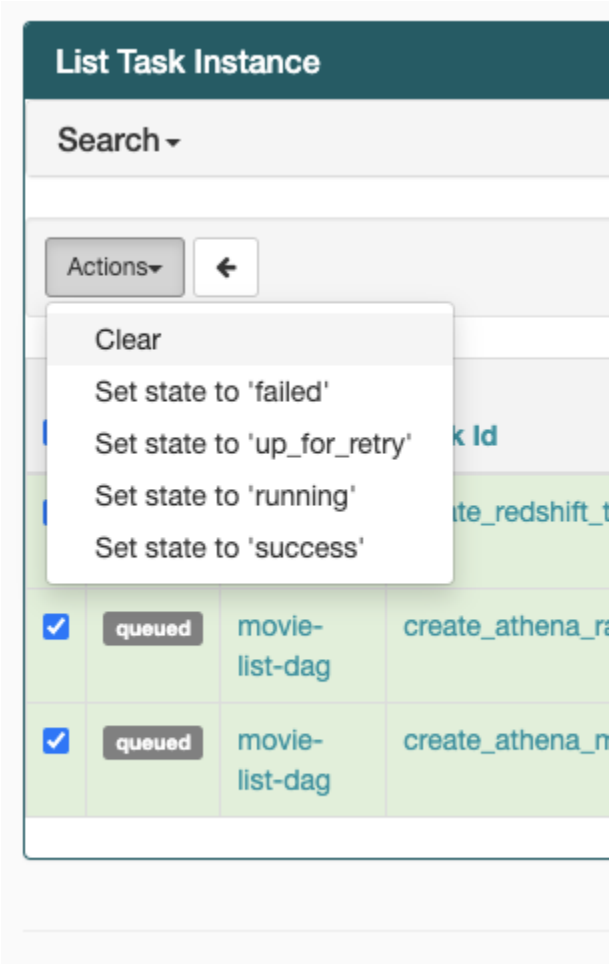
d. また、最小ワーカー数を環境内の最大ワーカー数と同じようにに設定して、オートスケーリングを効果的に無効にすることもできます。AWS Command Line Interface (AWS CLI) の「update-environment」コマンドを使用して、ワーカーの最小数と最大数を同じに設定して自動スケーリングを無効にします。

```
aws mwaa update-environment --name MyEnvironmentName --min-workers 5 --max-
workers 5
```

e. ご使用の環境のパフォーマンスを調整するために推奨するベスト・プラクティスの詳細については、「the section called “Apache Airflow のパフォーマンス調整”」を参照してください。

4. タスクが「実行中」状態のままになっている場合は、タスクをクリアしたり、成功または失敗のマークを付けることもできます。これにより、環境の自動スケーリングコンポーネントは、環境で実行されているワーカー数をスケールダウンできます。次の図は、取り残されたタスクの例です。

   - 取り残されたタスクの円を選択し、「クリア」を選択します (図を参照) 。これにより Amazon MWAA はワーカーをスケールダウンできます。そうしない場合、Amazon MWAA はどの DAG が有効か無効かを判断できず、キューにタスクが残っている場合はスケールダウンできません。



5. Apache Airflow タスクライフサイクルの詳細については、Apache Airflow リファレンスガイドの「概念」を参照してください。

CLI

次のトピックでは、AWS Command Line Interface で Airflow CLI コマンドを実行したときに発生する可能性があるエラーについて説明します。

CLI で DAG をトリガーすると「503」エラーが表示されます

Airflow CLI は Apache Airflow ウェブサーバー上でで実行され、同時実行性が制限されています。通常、 CLI コマンドを最大 4 つ同時に実行できます。

トラブルシューティング:Amazon MWAA 環境の作成と更新

このページのトピックでは、Apache Airflow 環境用 Amazon マネージドワークフローを作成および更新する際に発生する可能性のあるエラーと、これらのエラーの解決方法を記載しています。

目次

requirements.txt の更新

以下のトピックでは、requirements.txt を更新する際に受け取る可能性のあるエラーについて説明しています。

新しいバージョンを指定しましたが、requirements.txt 環境の更新に 20 分以上かかります。

お客様の環境で requirements.txt ファイルの新しいバージョンをインストールするのに 20 分以上かかる場合、環境の更新は失敗し、Amazon MWAA はコンテナイメージの最新の安定したバージョンにロールバックします。

1. パッケージのバージョンを確認します。requirements.txt 内のPython依存関係には、常に特定のバージョン (==) または最上位バージョン (>=) を指定することをお勧めします。
2. Apache Airflow のログを確認してください。Apache Airflow ログを有効にした場合は、CloudWatch コンソールの「ロググループ」ページでロググループが正常に作成されたことを確認します。空のログが表示される場合、最も一般的な理由は、ログが書き込まれる CloudWatch または Amazon S3 の実行ロールにアクセス権限がないことが原因です。詳細については、「実行ロール」を参照してください。
3. Apache Airflow 構成オプションを確認してください。Secrets Manager を使用している場合は、Apache Airflow 構成オプションとして指定したキーと値のペアが正しく設定されていることを確認してください。詳細については、「the section called “Secrets Manager の設定”」を参照してください。
4. VPCのネットワーク構成を確認します。詳細については、「the section called “環境スタック”」を参照してください。
5. 実行ロールの権限を確認します。実行ロールは、AWS ユーザーに代わって他のサービス AWS Identity and Access Management (Amazon S3、CloudWatch、Amazon SQS、Amazon ECR など) のリソースを呼び出すアクセス権限を Amazon MWAA に付与するアクセス権限ポリシーを持つ (IAM) ロールです。「顧客管理キー」または「AWS 所有キー」にもアクセスを許可する必要があります。詳細については、「実行ロール」を参照してください。

6. Amazon MWAA 環境の Amazon VPC ネットワークのセットアップと設定を確認するトラブルシューティングスクリプトを実行するには、GitHub AWS Support ツールにある「環境の検証」スクリプトを参照してください。

プラグイン

次のトピックでは、Apache Airflow プラグインを設定または更新する際に発生する可能性のある問題について説明します。

Amazon MWAA はカスタム UI の実装をサポートしていますか。

Apache Airflow v2.2.2 以降では、Amazon MWAA は Apache Airflow ウェブサーバーへのプラグインのインストールとカスタム UI の実装をサポートしています。Amazon MWAA 環境で Apache Airflow v2.0.2 以前を実行している場合、カスタム UI を実装することはできません。

バージョン管理と既存環境のアップグレードの詳細については、「バージョン」を参照してください。

プラグインを使用して「Amazon MWAA ローカルランナー」にカスタム UI の変更を実装できますが、Amazon MWAA で同じことを行おうとしても、変更内容は表示されず、エラーも発生しません。この問題が発生する理由

Amazon MWAA ローカルランナーでは、すべての Apache Airflow コンポーネントが 1 つのイメージにバンドルされているため、カスタム UI プラグインの変更を適用できます。

バケットを作成する

次のトピックでは、Amazon S3 バケットを作成する時に発生する可能性があるエラーについて説明します。

S3 のパブリックアクセスのブロック設定を選択できない

Amazon MWAA環境の「実行ロール」は、Amazon S3バケット上の `GetBucketPublicAccessBlock` 操作に対する権限を必要とし、バケットがパブリックアクセスをブロックしていることを確認します。次のステップを推奨します。

1. 手順に従って「JSON ポリシーを実行ロールに追加します」。
2. 以下の JSON ポリシーを添付する：

```
{
```

```
    "Effect":"Allow",
    "Action":[
       "s3:GetObject*",
       "s3:GetBucket*",
       "s3:List*"
    ],
    "Resource":[
       "arn:aws:s3:::YOUR_S3_BUCKET_NAME",
       "arn:aws:s3:::YOUR_S3_BUCKET_NAME/*"
    ]
}
```

YOUR_S3_BUCKET_NAME のサンプルプレースホルダーを、*my-mwaa-unique-s3-bucket-name* などの Amazon S3 バケット名に置き換えてください。

3. Amazon MWAA 環境の Amazon VPC ネットワークのセットアップと設定を確認するトラブルシューティングスクリプトを実行するには、GitHub AWS Support ツールにある「環境の検証」スクリプトを参照してください。

環境を作成する

次のトピックでは、環境を作成する時に発生する可能性がある エラーについて説明します。

環境を作成しようとしましたが、「作成中」の状態のままです。

次のステップを推奨します。

1. VPC ネットワークをパブリックルーティングで確認してください。インターネットにアクセスできる Amazon VPC を使用している場合は、以下を確認してください。

   - Amazon VPCは、 AWS で定義されているように、Amazon MWAA環境で使用される異なる「the section called “ネットワークについて”」 リソース間のネットワークトラフィックを許可するように構成されています。例えば、VPC セキュリティグループは、自己参照ルールですべてのトラフィックを許可するか、オプションで HTTPS ポート範囲 443 と TCP ポート範囲 5432 のポート範囲を指定する必要があります。

2. プライベートルーティングでVPCネットワークを確認します。インターネットにアクセスせずに Amazon VPC を使用している場合は、以下を確認してください。

   - Amazon VPC が、the section called “ネットワークについて” で定義されているように、Amazon MWAA 環境の異なる AWS リソース間のネットワーク・トラフィックを許可

するように構成されています。たとえば、2 つのプライベートサブネットには NAT ゲートウェイ (または NAT インスタンス) へのルートテーブルやインターネットゲートウェイを持つことはできません。

3. Amazon MWAA 環境の Amazon VPC ネットワークのセットアップと設定を確認するトラブルシューティングスクリプトを実行するには、GitHub AWS Support ツールにある「環境の検証」スクリプトを参照してください。

環境を作成しようとしましたが、ステータスが「Create failed」と表示されます。

次のステップを推奨します。

1. VPCのネットワーク構成を確認します。詳細については、「the section called “環境スタック”」を参照してください。
2. ユーザー権限を確認してください。Amazon MWAA は、環境を作成する前に、ユーザーの認証情報に対してドライランを実行します。お客様の AWS アカウントには、環境用のリソースの一部を作成する権限が AWS Identity and Access Management (IAM) にない場合があります。たとえば、プライベートネットワークの Apache Airflow アクセスモードを選択した場合、AWS アカウントに VPC エンドポイントの作成を許可する環境の「AmazonMWAAFullConsoleAccess」アクセスコントロールポリシーへのアクセス権限が管理者から付与されている必要があります。
3. 実行ロールの権限を確認します。実行ロールは、AWS ユーザーに代わって他のサービス AWS Identity and Access Management (Amazon S3、CloudWatch、Amazon SQS、Amazon ECR など) のリソースを呼び出すアクセス権限を Amazon MWAA に付与するアクセス権限ポリシーを持つ (IAM) ロールです。「顧客管理キー」または「 AWS 所有キー」にもアクセスを許可する必要があります。詳細については、「実行ロール」を参照してください。
4. Apache Airflow のログを確認してください。Apache Airflow ログを有効にした場合は、CloudWatch コンソールの「ロググループ」ページでロググループが正常に作成されたことを確認します。空のログが表示される場合、最も一般的な理由は、ログが書き込まれる CloudWatch または Amazon S3 の実行ロールにアクセス権限がないことが原因です。詳細については、「実行ロール」を参照してください。
5. Amazon MWAA 環境の Amazon VPC ネットワークのセットアップと設定を確認するトラブルシューティングスクリプトを実行するには、GitHub AWS Support ツールにある「環境の検証」スクリプトを参照してください。
6. インターネットにアクセスせずに Amazon VPC を使用している場合は、Amazon S3 ゲートウェイエンドポイントを作成し、Amazon S3 にアクセスするために必要最小限のアクセス権限を Amazon ECR に付与していることを確認してください。Amazon S3 ゲートウェイエンドポイントの作成についての詳細は、以下を参照してください。

- 「インターネットに接続せずに Amazon VPC ネットワークを作成する」
- Amazon Elastic Container Registry ユーザーガイドの「Amazon S3 ゲートウェイのエンドポイントの作成」

VPC を選択したときに「ネットワーク障害」エラーが表示されました

次のステップを推奨します。

- 環境の作成時に Amazon VPC を選択したときに「Network Failure」エラーが表示される場合は、実行中のブラウザ内プロキシをすべてオフにしてから、もう一度試してください。

環境を作成したときに、サービス、パーティション、またはリソースに「渡さなければなりません」というエラーが表示されました。

次のステップを推奨します。

- Amazon S3 バケットに指定した URI の URI の末尾に「/」が含まれている場合、このエラーが表示される可能性があります。パスの '/' を削除することをお勧めします。値は以下のフォーマットでなければならない：

```
s3://your-bucket-name
```

環境を作成しようとしたところ、ステータスが「使用可能」と表示されているのに、Airflow UI にアクセスしようとすると、「サーバーからの返信が空です」または「502 Bad ゲートウェイ」というエラーが表示されます。

次のステップを推奨します。

1. VPC セキュリティグループの設定を確認する。詳細については、「the section called “環境スタック”」を参照してください。
2. リストアップされている Apache Airflow パッケージが、Amazon MWAA で実行している Apache Airflow `requirements.txt` バージョンに対応していることを確認してください。詳細については、「Python 依存関係のインストール」を参照してください。

3. Amazon MWAA 環境の Amazon VPC ネットワークのセットアップと設定を確認するトラブルシューティングスクリプトを実行するには、GitHub AWS Support ツールにある「環境の検証」スクリプトを参照してください。

ユーザー名がランダムな文字名の集まりになっている環境を作成しようとしました

- Apache Airflow のユーザ名は最大 64 文字です。AWS Identity and Access Management (IAM) ロールがこの長さを超えた場合は、ハッシュアルゴリズムを使用して一意性を保ちながら短縮します。

update-environment

以下のトピックでは、環境を更新する際に受け取る可能性のあるエラーについて説明します。

環境クラスを変更しようとしましたが、更新に失敗しました

環境を別の環境カテゴリ (`mw1.medium` を `mw1.small` に変更するなど) に更新し、環境の更新リクエストに失敗した場合、環境の状態は `UPDATE_FAILED` 状態になり、以前の安定したバージョンの環境にロールバックされ、そのバージョンに応じて課金されます。

次のステップを推奨します。

1. GitHub の「aws-mwaa-local-runner」を使用して、DAG、カスタムプラグイン、Python の依存関係をローカルでテストします。
2. Amazon MWAA 環境の Amazon VPC ネットワークのセットアップと設定を確認するトラブルシューティングスクリプトを実行するには、GitHub AWS Support ツールにある「環境の検証」スクリプトを参照してください。

アクセス環境

以下のトピックでは、環境にアクセスする際に受け取る可能性のあるエラーについて説明します。

Apache Airflow UI にアクセスできません

次のステップを推奨します。

1. ユーザー権限を確認してください。Apache Airflow UIを表示できるアクセス権ポリシーの権限が与えられていない可能性があります。詳細については、「the section called “Amazon MWAA 環境へのアクセス”」を参照してください。
2. ネットワークアクセスを確認してください。[プライベートネットワーク] アクセスモードを選択したことが原因と考えられます。Apache Airflow UI の URL が以下の形式の `387fbcn-8dh4-9hfj-0dnd-834jhdfb-vpce.c10.us-west-2.airflow.amazonaws.com` である場合、これは Apache Airflow ウェブサーバー用にプライベートルートを使用していることを意味します。Apache Airflow アクセスモードをパブリックネットワークアクセスモードに更新するか、Apache Airflow ウェブサーバーの VPC エンドポイントにアクセスするメカニズムを作成できます。詳細については、「the section called “VPC エンドポイントへのアクセスの管理”」を参照してください。

トラブルシューティング:CloudWatch ログ記録と CloudTrail のエラー

このページのトピックには、Apache Airflow 用 Amazon マネージドワークフロー環境で発生する可能性のある Amazon CloudWatch Logs と AWS CloudTrail エラーの解決策が含まれています。

目次

ログ

次のトピックでは、Apache Airflow ログを表示する際に発生する可能性があるエラーについて説明します。

タスクログが表示されないか、「Cloudwatch log_group からリモートログを読み取っています」というエラーが表示されます。

Amazon MWAA は、Amazon CloudWatch Logs との間で直接ログを読み書きするように Apache Airflow を設定しています。ワーカーがタスクの開始に失敗したり、ログの書き込みに失敗したりすると、エラーが表示される：

```
*** Reading remote log from Cloudwatch log_group: airflow-environmentName-Task
 log_stream: DAG_ID/TASK_ID/timestamp/n.log.Could not read remote logs from log_group:
 airflow-environmentName-Task log_stream: DAG_ID/TASK_ID/time/n.log.
```

- 次のステップを推奨します。

  a. 環境の INFO レベルでタスクログを有効にしていることを確認します。詳細については、「Amazon での Airflow ログの表示 CloudWatch」を参照してください。

  b. 環境「実行ロール」に正しい権限ポリシーがあることを確認します。

  c. オペレータまたはタスクが正しく動作していること、DAG を解析するのに十分なリソースがあること、ロードする適切な Python ライブラリがあることを確認します。正しい依存関係があることを確認するには、問題の原因となっているインポートが見つかるまで、インポートを削除してみます。「Amazon MWAA ローカルランナーツール」を使用して Python の依存関係をテストすることをお勧めします。

ログなしでタスクが失敗しました

ワークフロー内のタスクが失敗し、失敗したタスクのログが見つからない場合は、次に示すように、デフォルトパラメータに queue パラメータが設定されていることを確認してください。

```
from airflow import DAG
from airflow.operators.bash_operator import BashOperator
from airflow.utils.dates import days_ago

# Setting queue argument to default.
default_args = {
 "start_date": days_ago(1),
```

```
 "queue": "default"
}

with DAG(dag_id="any_command_dag", schedule_interval=None, catchup=False,
 default_args=default_args) as dag:
    cli_command = BashOperator(
        task_id="bash_command",
        bash_command="{{ dag_run.conf['command'] }}"
    )
```

問題を解決するには、コードから queue を削除し、DAG を再実行してください。

CloudTrail に「リソースはすでに存在している例外」というエラーが表示されます

```
"errorCode": "ResourceAlreadyExistsException",
    "errorMessage": "The specified log stream already exists",
    "requestParameters": {
        "logGroupName": "airflow-MyAirflowEnvironment-DAGProcessing",
        "logStreamName": "scheduler_cross-account-eks.py.log"
    }
```

特定の Python 要件、例えば Amazon MWAA が CloudWatch と通信するために使用する `watchtower` ライブラリを以前のバージョンに戻す `apache-airflow-backport-providers-amazon` があるかもしれません。次のステップを推奨します。

- 以下のライブラリを `requirements.txt` に追加する

  ```
  watchtower==1.0.6
  ```

CloudTrail に「リクエストが無効です」というエラーが表示されます。

```
Invalid request provided: Provided role does not have sufficient permissions for s3
 location airflow-xxx-xxx/dags
```

同じDXテンプレートを使用してAmazon AWS CloudFormation 環境とAmazon S3バケットを作成している場合は、AWS CloudFormation テンプレートに `DependsOn` セクションを追加する必要があります。この2つのリソース (MWAA 環境とMWAA 実行ポリシー) は、AWS CloudFormation 内で依存関係にあります。次のステップを推奨します。

- AWS CloudFormation のテンプレートに以下の **`DependsOn`** 文を追加してください。

```
...
      MaxWorkers: 5
      NetworkConfiguration:
        SecurityGroupIds:
          - !GetAtt SecurityGroup.GroupId
        SubnetIds: !Ref subnetIds
      WebserverAccessMode: PUBLIC_ONLY
    DependsOn: MwaaExecutionPolicy

    MwaaExecutionPolicy:
    Type: AWS::IAM::ManagedPolicy
    Properties:
      Roles:
        - !Ref MwaaExecutionRole
      PolicyDocument:
        Version: 2012-10-17
        Statement:
          - Effect: Allow
            Action: airflow:PublishMetrics
            Resource:
...
```

例については、「Amazon Managed Workflows for Apache Airflow のクイックスタートチュートリアル」を参照してください。

Apache Airflow ログに「64 ビット Oracle クライアントライブラリが見つかりませんでした:「libclntsh.so: 共有オブジェクトファイルを開くことができません:そのようなファイルまたはディレクトリはありません」と表示されます。

- 次のステップを推奨します。
  - Apache Airflow v2 を使用している場合、`core.lazy_load_plugins : False` を Apache Airflow の構成オプションとして追加してください。詳細については、「2 の設定オプションによるプラグインの読み込み」を参照してください。

ディスパッチャの口グに psycopg2 「サーバが接続を誤って閉じました」とあります。

次のようなエラーが表示される場合は、Apache Airflow スケジューラリソースが不足している可能性があります。

```
2021-06-14T10:20:24.581-05:00    sqlalchemy.exc.OperationalError:
 (psycopg2.OperationalError) server closed the connection unexpectedly
2021-06-14T10:20:24.633-05:00    This probably means the server terminated abnormally
2021-06-14T10:20:24.686-05:00    before or while processing the request.
```

次のステップを推奨します。

- 最大 5 つのスケジューラを指定できる Apache Airflow v2.0.2 へのアップグレードを検討してください。

「タスクインスタンス%sが完了したとExecutorから報告されました(%s)。ただし、タスクはDAG処理口グに%s表示されています。」が表示されました。

次のようなエラーが表示される場合は、長時間タスクの実行が Amazon MWAA のタスク時間制限に達している可能性があります。Amazon MWAAでは、どの Airflow タスクにも12時間の制限があり、タスクがキューに詰まったり、自動スケーリングなどのアクティビティをブロックしたりすることができないようになっています

```
Executor reports task instance %s finished (%s) although the task says its %s. (Info:
 %s) Was the task killed externally
```

次のステップを推奨します。

- タスクを実行時間の短い複数のタスクに分解することを考慮してください。Airflow には通常、オペレーターが非同期になるというモデルがあります。外部システム上のアクティビティを呼び出し、Apache Airflow センサーがポーリングを行って処理が完了したことを確認します。センサーに障害が発生しても、オペレーターの機能に影響を与えることなく、安全に再試行できます。

「log_group: airflow-*{*environmentName}-Task log_stream:* {*DAG_ID}/*{*TASK_ID}/*{*time}/*{*n}.log からリモートログを読み込めませんでした」と表示されました。自分のタスクログ

次のようなエラーが表示される場合、ご使用の環境の実行ロールに、タスクログのログストリームを作成するためのアクセス権限ポリシーが含まれていない可能性があります。

```
Could not read remote logs from log_group: airflow-*{*environmentName}-Task
 log_stream:* {*DAG_ID}/*{*TASK_ID}/*{*time}/*{*n}.log.
```

次のステップを推奨します。

- 「the section called “実行ロール”」 にあるサンプルポリシーのいずれかを使用して、環境の実行ロールを変更します。

`requirements.txt` ファイルでApache Airflowバージョンと互換性のないプロバイダパッケージを指定している場合もあります。たとえば、Apache Airflow v2.0.2 を使用している場合、apache-airflow-providers-databricks パッケージのように、Airflow 2.1+とのみ互換性のあるパッケージを指定している可能性があります。

次のステップを推奨します。

1. Apache Airflow v2.0.2 を使用している場合は、`requirements.txt` ファイルを変更して `apache-airflow[databricks]` を追加します。これにより、Apache Airflow v2.0.2 と互換性のある正しいバージョンの Databricks パッケージがインストールされます。
2. GitHub の「aws-mwaa-local-runner」を使用して、DAG、カスタムプラグイン、Python の依存関係をローカルでテストします。

Amazon MWAA のドキュメント履歴

次の表は、2020 年 11 月以降の Amazon MWAAサービスのドキュメントへの重要な追加項目をまとめたものです。このドキュメントの更新に関する通知については、「RSS フィード」を参照してください。

変更	説明	日付
Amazon MWAA がウェブサーバーの自動スケーリングと Apache Airflow REST API をサポート	Amazon MWAA は、ウェブサーバーの自動スケーリングと、Apache Airflow REST API にアクセスして使用する機能をサポートするようになりました。 • the section called “ウェブサーバーのオートスケーリングの設定” • the section called “Apache Airflow REST API の使用”	2024 年 5 月 16 日
自動スケーリング動作の説明を改善	Fargate ワーカーがダウンスケールとして新しいタスクを選択した際の新しい Amazon MWAA 自動スケーリング動作を反映するために、次のトピックを更新しました。 • the section called “ワーカーの自動スケーリングの設定”	2024 年 5 月 10 日
より大きなインスタンスサイズのサポート	Amazon MWAA は、より大きなワークロードに対して `mw1.xlarge` 、 の 2 つのより大きなインスタンスサイズオプションをサポートするよ	2024 年 4 月 16 日

	うになりました。 `mw1.2xlarge` • the section called “環境機能”	
Apache Airflow の新しいバージョン	Amazon MWAA が Apache Airflow v2.8.1 をサポートするようになりました。この更新には、更新されたプロバイダーパッケージに関する情報と、Amazon MWAA での Apache Airflow v2.8.1 の使用に関する詳細が含まれています。 • バージョン • the section called “Apache Airflow v2.8.1 接続のプロバイダーパッケージ”	2024 年 2 月 22 日
Amazon VPC のサポート	Amazon MWAA は、Amazon OpenSearch Service を使用して、所有者アカウント内の中央共有 Amazon VPC を使用して Amazon MWAA リソースを管理する組織のクロスアカウント環境の作成をサポートしています。このリリースの一環として、Amazon MWAA では独自の Amazon VPC エンドポイントを作成して管理することを選択できます。 • the section called “独自の Amazon VPC エンドポイントの管理”	2023 年 11 月 15 日

Apache Airflow の新しいバージョン	Amazon MWAA は Apache Airflow v2.7.2 をサポートするようになりました。このアップデートには、更新されたプロバイダーパッケージに関する情報と、Amazon MWAA での Apache Airflow v2.7.2 の使用に関する詳細が含まれています。 • バージョン • the section called “Apache Airflow v2.7.2 接続用のプロバイダーパッケージ”	2023 年 11 月 6 日
Apache Airflow の新しいバージョン	Amazon MWAA は Apache Airflow v2.6.3 をサポートするようになりました。このアップデートには、更新されたプロバイダーパッケージに関する情報と、Amazon MWAA での Apache Airflow v2.6.3 の使用に関する詳細が含まれています。 • バージョン • the section called “Apache Airflow v2.6.3 接続用のプロバイダーパッケージ”	2023 年 8 月 9 日

バージョン廃止情報	バージョン廃止に関するトピックを更新し、Apache Airflow v2.0.2 と Apache Airflow v2.2.2 の廃止に関する通知とタイムラインが追加されました。 • the section called “Apache Airflow の非推奨バージョン”	2023 年 7 月 31 日
新しいトピックとユースケース	Amazon MWAA はマイナーバージョンアップグレードをサポートしています。この更新には、環境をアップグレードし、ワークフローリソースがアップグレード先の Apache Airflow のバージョンと互換性があることを確認する方法を説明する次の新しいトピックが含まれています。 • the section called “エンジンバージョンのアップグレード”	2023 年 6 月 5 日

トピックの更新	Amazon MWAA へのユーザーの完全なコンソールおよびAPIアクセスを付与するための顧客管理型IAMポリシーが更新されました。この更新では、Amazon MWAA に役割を渡すためにユーザーに `iam:PassRole` の権限を付与する必要がある理由について説明しています。Amazon MWAA はこれらの権限を使用してユーザーに代わってアクションを実行します。 • the section called “Amazon MWAA 環境へのアクセス”	2023 年 4 月 12 日

新しいガイダンス	Amazon MWAA のバックエンド AWS Secrets Manager としての の設定に関するトピックを更新し、ルックアップパターンの使用に関するガイダンスを提供しました。ルックアップパターンを使用すると、Apache Airflow が検索するシークレットが絞り込まれ、Amazon MWAA が接続と変数を取得するために Secrets Manager に対して行う API 呼び出しの数が減ります。これにより、Secrets Manager をバックエンドとして使用することに伴うコストが削減されます。 • Secrets Manager のバックエンドを Apache Airflow 構成オプションとして作成する	2023 年 4 月 12 日
Apache Airflow の新しいバージョン	Amazon MWAA は、Apache Airflow v2.5.1 をサポートするようになりました。このアップデートには、更新されたプロバイダーパッケージに関する情報と、Amazon MWAA での Apache Airflow v2.5.1 の使用に関する詳細が含まれています。 • バージョン • the section called “Apache Airflow v2.5.1 接続用のプロバイダーパッケージ”	2023 年 4 月 11 日

新しいトピックとユースケース	Amazon MWAA 環境でのスタートアップスクリプトの使用に関する新しいトピックが追加されました。このトピックでは、既存の環境用のスタートアップスクリプトの設定、そのスタートアップスクリプトによる Linux ランタイムのインストール、および環境変数の設定について説明します。 • the section called “スタートアップスクリプトの使用”	2023 年 4 月 3 日
プライベート Web サーバーへのアクセスに関して更新されたセクション	プライベート Web サーバーアクセスに関する次のトピックが更新されました。この更新により、プライベート Web サーバーにアクセスできる環境では、Python wheel archive (`.whl`) を使用して依存関係をパッケージ化およびインストールする必要があることが明確になりました。 • プライベート Web サーバーアクセスモード	2023 年 2 月 24 日

非推奨の Apache Airflow バージョンに関して追加された情報	バージョントピックを新しい情報で更新し、Amazon MWAA がApache Airflow のバージョンを廃止する方法に関する情報が追加されました。Apache Airflow の新しいバージョンへのアップグレードに関するセクションと、Apache Airflow v1 と Apache Airflow v2 の間の変更点を説明するセクションが削除されました。Apache Airflow の新しいバージョンへの移行に関する詳細については、「Amazon MWAA 移行ガイド」を参照してください。 • the section called “Apache Airflow の非推奨バージョン” • the section called “Apache Airflow のバージョンサポートとよくある質問”	2023 年 2 月 17 日

Amazon MWAA コンテナメトリクスの修正点	コンテナメトリックスのトピックを更新し、Cluster ディメンションの下に存在しなかった一連の誤ったメトリックスが削除されました。特定の時点で環境が利用している追加のワーカーの数を評価する方法について説明している追加のセクションが追加されました。これは、CPUUtilization または MemoryUtilization メトリックを AdditionalWorker コンポーネントに対してグラフ化し、統計タイプを Sample Count に設定することで行います。 • the section called “追加のワーカーコンテナとウェブサーバーコンテナの数の評価”	2023 年 1 月 20 日

Apache Airflow の新しいバージョン	Amazon MWAA は Apache Airflow v2.4.3 をサポートするようになりました。このアップデートには、アップデートされたプロバイダパッケージに関する情報、Amazon MWAA で Apache Airflow v2.4.3を使用する方法の詳細、および Amazon MWAA で各 Apache Airflow バージョンでサポートされている機能に関する情報が含まれています。 • バージョン • the section called “Apache Airflow v2.4.3 接続用のプロバイダーパッケージ”	2023 年 1 月 5 日
サービスにリンクされたロールに関して更新されたトピック	Amazon MWAA がユーザーに代わって AWS リソースを作成および管理するために使用するサービスにリンクされたロールに関する情報を更新しました。これには、不要になったサービスにリンクされたロールを削除する方法に関する情報が含まれます。これには、Amazon MWAA が`AWS/MWAA`名前空間の下に追加の CloudWatch メトリクスを発行できるようにする、更新されたサービスにリンクされたロールのアクセス許可ポリシーが含まれます。 • the section called “サービスリンクロール”	2022 年 11 月 18 日

サービスメトリックスに関する新しいトピック	Amazon MWAA が AWS/MWAA 名前空間の下で発行するサービスメトリクスに関する新しいトピックが追加されました。これには、Amazon ECS クラスターメトリクスのスケジューラ、ワーカー、および Webサーバー、Amazon MWAA がスケジューラとワーカーを分離するためのキューの Amazon SQS メトリクス、およびメタデータデータベースの Amazon RDSメトリクスが含まれています。 • the section called “コンテナ、キュー、データベースメトリクス”	2022 年 11 月 18 日
新しいトピック	Amazon MWAA 環境で使用するプロバイダーパッケージの新しいバージョンを指定するための制約ファイルの変更に関する新しいガイダンスが追加されました。 • the section called “新しいプロバイダーパッケージの指定”	2022 年 11 月 18 日
FAQ エントリが更新されました。	Amazon MWAA の HIPAA 資格に関する情報が更新されました。 • the section called “HIPAA への準拠”	2022 年 11 月 15 日

新しいトピック	Amazon MWAA の実行ロール信頼ポリシーで aws:SourceArn と aws:SourceAccount のグローバル条件コンテキストキーを使用して、クロスサービスの混乱したディピュティを防ぐための新しいトピックが追加されました。 • the section called “サービス間での不分別な代理処理の防止”	2022 年 10 月 21 日
新しいサンプルコード	にカスタム OS レベルのメトリクスを書き込む手順と DAG コード例を更新しましたCloudWatch。 • the section called “DAG を使用してカスタムメトリクスを記述する”	2022 年 9 月 13 日
新しいサンプルコード	Apache Airflow CLI トークンを取得し、指定された Amazon MWAA 環境で DAG を呼び出すための更新された手順と新しい AWS Lambda Python コード例を追加しました。 • the section called “Lambda を使用して DAG を呼び出す”	2022 年 9 月 12 日

新しいアーキテクチャ図	パブリックウェブサーバーとプライベートウェブサーバーを備えた Amazon MWAA 環境を示す新しいアーキテクチャ図が追加されました。 • the section called “Apache Airflow のアクセスモード”	2022 年 9 月 12 日
新しいサンプルコード	更新された手順と、Apache Airflow CLI トークンを取得し、別の Amazon MWAA 環境で別の DAG を呼び出す新しい DAG コード例が追加されました。 • the section called “さまざまな環境での DAG の呼び出し”	2022 年 8 月 16 日
新しいサンプルコード	環境の Aurora PostgreSQL にメタデータ情報をクエリし、結果を CSV ファイルに書き込み、ファイルを Amazon S3 に保存する手順が更新され、新しい DAG が追加されました。 • the section called “Amazon S3 への環境のメタデータのエクスポート”	2022 年 8 月 12 日

新しいサンプルコード	実行時に AWS CodeArtifact トークンを更新し、結果を Amazon S3 に保存する更新された手順と新しい DAG を追加しました。 • the section called “実行時に AWS CodeArtifact トークンを更新する。”	2022 年 8 月 3 日
新しいサンプルコード	Amazon MWAA ECSOperator でを使用するための最新の手順と DAG コードサンプルが追加されました。 • the section called “ECSOperator を使用する場合”	2022 年 7 月 26 日
新しいサンプルコード	Amazon MWAA SSHOperator でを使用するための最新の手順と DAG コードサンプルが追加されました。 • the section called “SSHOperator の使用”	2022 年 7 月 15 日
新しいサンプルコード	Amazon MWAA で dbt Postgres を使用するための新しい手順と DAG コードサンプルが追加されました。 • the section called “Amazon MWAA での dbt の使用”	2022 年 6 月 17 日

新しいトピックとユースケース	パブリックアクセスとプライベートアクセスがある Amazon MWAA 環境に Python ホイールファイルを使用して依存関係をインストールするための新しい手順と DAG コードサンプルが追加されました。 • Python ホイールを使った依存関係の管理	2022 年 5 月 13 日
新しいトピックとユースケース	Amazon MWAA が に送信する Apache Airflow メトリクスの選択に関する新しいガイダンスを追加しました CloudWatch。 • どの Apache Airflow メトリクスをレポートするかを選択する	2022 年 4 月 19 日
新しいガイド	Amazon MWAA では、Apache Airflow ワークフローを自己管理型のデプロイや既存の Amazon MWAA 環境から移行するための移行ガイドを提供しています。 • Amazon MWAA 移行ガイド	2022 年 3 月 7 日

新しいトピックとユースケース	Apache Airflow のユーザー権限の変更を検出するソリューションなど、Apache Airflow と連携するための新しいセキュリティのベストプラクティスが追加されました。 • the section called “Apache Airflow でのセキュリティのベストプラクティス”	2022 年 2 月 18 日
新しいサンプルコード	Pendulum を使用してタイムゾーン対応の DAG を作成する新しいコードサンプルを追加し、カスタムプラグインを使用して Apache Airflow ログが作成されるタイムゾーンを変更する方法を明確にしました。 • the section called “DAG のタイムゾーンの変更”	2022 年 2 月 11 日

Apache Airflow v2.2.2 の発売	Amazon Managed Workflows for Apache Airflow が Apache Airflow v2.2.2 をサポートするようになりました。v2.2 以降、Amazon MWAA は Python パッケージとカスタムプラグインを Apache Airflow ウェブサーバーに直接インストールするようになり、環境をより柔軟に管理できるようになりました。詳細については、以下を参照してください。 • Apache Airflow 用 Amazon マネージドワークフロー上の Apache Airflow のバージョン. • the section called “Apache Airflow v2.2.2 接続用のプロバイダーパッケージ”. • Apache Airflow v2.2.2 の変更履歴は、Apache Airflow ドキュメンテーションウェブサイトにあります。	2022 年 1 月 27 日

新規チュートリアル	新しいチュートリアルが追加されました。このチュートリアルでは、新しいカスタムApache Airflow ロールを作成し、IAMからマップされたApache Airflow ユーザーにそのロールを割り当て、ユーザーのアクセスを特定の DAG のサブセットに制限する方法を示しています。 • the section called “チュートリアル:ユーザーを DAG のサブセットに制限する”	2021 年 12 月 8 日

修正内容	`scheduler.min_file_process_interval` の値を設定するための最適な手法の推奨内容が修正され、CPU の使用が最適化されました。実行ロールの Secrets Manager リソースへのアクセスを許可する IAM ポリシーの例が追加されました。Secrets Manager の条件キーの使用に関するトラブルシューティングトピックが追加されました。 • スケジューラーが DAG を解析する方法のパフォーマンスチューニング • Amazon MWAA にSecrets Manager のシークレットキーにアクセスする権限を付与する • Secrets Manager の Amazon MWAA 実行ロールでの条件キーの設定	2021 年 11 月 22 日

新しいサンプルコード	以下は、カスタムプラグインを使用して DAG が処理されるタイムゾーンを変更するための新しいコードサンプルと、bashオペレータから `dags backfill` Apache Airflow CLI コマンドを呼び出す際のトラブルシューティングトピックが追加されました。 • the section called “DAG のタイムゾーンの変更” • bash 演算子を使用した CLI コマンドのバックフィル	2021 年 11 月 1 日
修正内容	Amazon ECS オペレータコードサンプルの問題を修正し、環境が CloudWatch Logs の Amazon ECS タスクロググループにアクセスできるようにするために Amazon MWAA 実行ロールに必要な追加のアクセス許可を明確にしました。 • Amazon ECS オペレーターのアクセス許可。	2021 年 10 月 26 日

新しいサンプルコード	新しいコードサンプルが追加されました。このサンプルは、Aurora PostgreSQL データベースからDAG実行に関連する情報をクエリし、結果を Amazon S3 に保存された CSV ファイルに書き込みます。 • the section called “Amazon S3 への環境のメタデータのエクスポート”.	2021 年 10 月 1 日
修正内容	Amazon MWAA が新しいオブジェクトや変更されたオブジェクトをターゲット Amazon S3 バケットからスケジューラとワーカーに自動的に同期する方法に関する情報を修正しました。 • DAG フォルダーの仕組み。	2021 年 10 月 1 日
現在サポート中	Amazon MWAA は Apache Airflow 2.0 以降のプロバイダーパッケージを追加でサポートするようになりました。サポートされているパッケージの詳細については、次を参照してください。 • the section called “Apache Airflow v2.0.2 接続用のプロバイダーパッケージ”.	2021 年 9 月 24 日

新しいコマンドとプロシージャ	インターネットにアクセスせずに Amazon VPC を使用する場合に Amazon S3 ゲートウェイエンドポイントを作成するための追加のガイダンスと AWS CLI コマンド例を追加しました。 • インターネットにアクセスせずに Amazon VPC ネットワークを作成する。	2021 年 9 月 24 日
新しいトピックとユースケース	次の変更が追加されました。 • the section called “ECSOperator を使用する場合” で Amazon Elastic Container Service オペレータを使用する新しいコードサンプルが追加されました。 • the section called “プラグイン” で Apache Airflow プラグインを設定する際の問題に関する新しいトラブルシューティングトピックが追加されました。	2021 年 9 月 19 日

新しくサポートされるリージョン	Amazon MWAA は、以下のリージョンでご利用いただけるようになりました。 • アジアパシフィック (ムンバイ) ap-south-1 • アジアパシフィック (ソウル) ap-northeast-2 • 欧州 (ロンドン) eu-west-2 • 欧州 (パリ) eu-west-3 • カナダ (中部) ca-central-1 • 南米 (サンパウロ) sa-east-1 リージョンの利用可能性とエンドポイントについては、以下を参照してください。 • AWS 全般のリファレンスの「Amazon CloudWatch エンドポイントとクォータ」	2021 年 8 月 31 日
新しいトピックとユースケース	次の変更が追加されました。 • Amazon MWAA がアカウントレベルの Amazon S3 設定 (`s3:GetAccountPublicAccessBlock`) を Amazon MWAA 実行ロールで取得できるようにサンプルポリシーが更新されました。	2021 年 8 月 27 日

修正内容	次の変更が追加されました。 • のセキュリティグループに自己参照インバウンドルールを使用するように AWS CloudFormation テンプレートを修正しましたVPCネットワークを作成する。 • のセキュリティグループに自己参照インバウンドルールを使用するように AWS CloudFormation テンプレートを修正しましたAmazon Managed Workflows for Apache Airflow のクイックスタートチュートリアル。	2021 年 8 月 27 日
新しいトピックとユースケース	次の変更が追加されました。 • Apache Airflow v2.0.2 Apache Airflow 用 Amazon マネージドワークフロー上の Apache Airflow のバージョン でサポートされているもののリストに DAG デコレーターが追加されました。	2021 年 8 月 20 日

新しいトピックとユースケース	次の変更が追加されました。 • `celery.sync_parallelism` ユースケースが Amazon MWAA での Apache Airflow のパフォーマンス調整 に追加されました。 • クォータページにサービスエンドポイントが追加され、名前が Amazon Managed Workflows for Apache Airflow のサービスエンドポイントとクォータ に変更されました。 • Amazon Managed Workflows for Apache Airflowを使い始める でのユーザーからのフィードバックに基づいて、ネットワークの前提条件が明確化されました。 • Apache Airflow CLI コマンドリファレンス でサポートされていない Airflow CLI コマンドに `dags list-runs` および `dags next-execution` が移動されました。	2021 年 8 月 13 日

新しいサンプルコード	次の変更が追加されました。 • Apache Airflow CLI コマンドリファレンス で Apache Airflow v2.0.2 変数を設定、取得、または削除するための bash の例が追加されました。 • Apache Airflow v2.0.2 の依存関係とエアフロー接続例が Amazon RDS for Microsoft SQL Server で Amazon MWAA を使用する に追加されました。	2021 年 8 月 13 日
修正内容	次の変更が追加されました。 • SSHOperator を使用して SSH 接続の作成 でのユーザーフィードバックに基づく Python コードサンプルを修正しました。	2021 年 8 月 13 日

新しいトピックとユースケース	次の変更が追加されました。 • `variables set`でサポートされている Airflow CLI コマンドが Apache Airflow CLI コマンドリファレンス に移動されました。 • ユーザーフィードバックに基づいて、Airflow バージョンページから v2.0.2での変更点の概要が Python 依存関係のインストール に追加されました。 • ユーザーフィードバックに基づいて、Airflow バージョンページから v2.0.2での変更点の概要が Apache Airflow CLI コマンドリファレンス に追加されました。 • ユーザーフィードバックに基づいて、Airflow バージョンページから v2.0.2での変更点の概要が 接続タイプの概要 に追加されました。 • ユーザーフィードバックに基づいて、Airflow バージョンページから v2.0.2での変更点の概要が カスタムプラグインのインストール に追加されました。 • ユーザーフィードバックに基づいて、Airflow バージョンページから v2.0.2での変更点の概要が DAG の追加と更新 に追加されました。	2021 年 8 月 6 日

新しいサンプルコード	次の変更が追加されました。 • Apache Airflow v2.0.2 のサンプルコードが DAG を使用して CLI に変数をインポートする に追加されました。 • Apache Airflow v2.0.2 のサンプルコードが Lambda 関数を使用して DAG を呼び出す に追加されました。	2021 年 8 月 6 日
新しいトピックとユースケース	次の変更が追加されました。 • 「Airflow UI に接続が表示されない」のトラブルシューティングトピックが Amazon Managed Workflows for Apache Airflow のトラブルシューティング に追加されました。 • Amazon MWAA がサポートするAmazon VPC のリストが Amazon MWAA でのネットワーキングについて に追加されました。	2021 年 7 月 29 日

修正内容	次の変更が追加されました。 • Apache Airflow ウェブサーバーアクセストークンを作成する でのユーザーフィードバックに基づいて、Python コードサンプルを修正し、Web ログイントークンを表示するようにしました。 • ユーザーフィードバックに基づいて、Snowflake 接続のトピックを修正し、Amazon Managed Workflows for Apache Airflow のトラブルシューティング で倉庫パラメータにシングルクォートを使用するようにしました。	2021 年 7 月 29 日
削除または移動されたトピック	次の変更が追加されました。 • 既存のページを再構築して、すべての監視およびメトリクスのドキュメントページが Apache Airflow 用 Amazon マネージドワークフローのモニタリングとメトリクス に含められました。 • の Apache Airflow v2 環境メトリクス CloudWatchを監視およびメトリクスのナビゲーションメニューに移動されました。	2021 年 7 月 23 日

新しいガイド	次の変更が追加されました。 • Amazon MWAA 環境にインストールされている Apache エアフロープロバイダーパッケージ が作成されました。 • Amazon MWAA でのモニタリングの概要 が作成されました。 • での監査ログの表示 AWS CloudTrail が作成されました。 • Amazon での Airflow ログの表示 CloudWatch が作成されました。	2021 年 7 月 23 日
修正内容	次の変更が追加されました。 • ユーザーフィードバックに基づく Python コードサンプルが修正され Airflow 接続文字列を正しい順序で生成し、AWS Secrets Manager シークレットを使用した Apache Airflow 接続の設定 にポートパラメータが追加されました。 • ユーザーフィードバックに基づいて、Oracle でのカスタムプラグインの作成にローカルにunzipパッケージをインストールする手順が追加されました。	2021 年 7 月 23 日

新しいトピックとユースケース	次の変更が追加されました。 • AWS DMS Operators のトピックが に追加されましたAmazon MWAA に関するよくある質問。 • リモートログエラーのトラブルシューティングトピックが Amazon Managed Workflows for Apache Airflow のトラブルシューティング に追加されました。 • Apache Airflow CLI コマンドリファレンス でサポートされていないAirflow CLI コマンドが `variables set` に移動されました。	2021年7月16日

新しいトピックとユースケース	次の変更が追加されました。 • Python 依存関係のインストール でのユーザーフィードバックに基づいて requirements.txt ファイルを作成する一連の手順が追加されました。 • カスタムプラグインのインストール でのユーザーフィードバックに基づいて plugins.zip ファイルを作成する一連の手順が追加されました。 • ユーザーガイド全体に、Amazon Managed Workflows for Apache Airflow API リファレンスガイドの API リファレンスガイドの相互参照リンクが追加されました。 • Amazon MWAA に関するよくある質問 での [Airflow 2.0 管理] > [プラグイン] メニューにプラグインが表示されない理由についてのトピックが追加されました。	2021 年 7 月 9 日
新しいガイド	次の変更が追加されました。 • Amazon S3 でファイルの削除 が作成されました。	2021 年 7 月 9 日

新しいトピックとユースケース	次の変更が追加されました。 • 暗号化のためのカスタマーマネージドキーの使用 でサポートされている値のリストが追加されました。 • ユーザーフィードバックに基づいて、requirements.txt での Python 依存関係の管理 内でプライベートリポジトリ URL の例が更新され、明確化されました。	2021 年 7 月 2 日
新しいサンプルコード	次の変更が追加されました。 • Apache Airflow v1.10.12 サンプルコードを追加し、の SSH 接続 AWS Secrets Manager に のプライベートキーを使用するようにしましたSSHOperator を使用して SSH 接続の作成。	2021 年 7 月 2 日
新しいトピックとユースケース	次の変更が追加されました。 • に StartedTaskInstances および FinishedTaskInstances メトリクスを追加しましたの Apache Airflow v2 環境メトリクス CloudWatch。	2021 年 6 月 25 日
新しいサンプルコード	次の変更が追加されました。 • Apache Airflow v2.0.2 のサンプルコードが Amazon EKS での Amazon MWAA の使用 に追加されました。	2021 年 6 月 25 日

新しいガイド	次の変更が追加されました。 • Amazon MWAA での Apache Airflow のパフォーマンス調整 が作成されました。	2021 年 6 月 25 日

新しいトピックとユースケース	次の変更が追加されました。 • Apache Airflow CLI コマンドリファレンス でサポートされている Apache Airflow v2.0.2 CLI コマンドに `connections add` および `connections delete` が追加されました。 • で利用可能な最新バージョン AWS CloudFormation が の Apache Airflow v2.0.2 であることを追加しましたAmazon Managed Workflows for Apache Airflow のクイックスタートチュートリアル。 • Apache Airflow ワーカーに一時データを保存するための質問が Amazon MWAA に関するよくある質問 に追加されました。 • 「Executor がタスクインスタンス %s を終了したと報告します」エラーのトピックが Amazon Managed Workflows for Apache Airflow のトラブルシューティング に追加されました。 • 「サーバーが接続を予期せず終了しました」ログのトピックが Amazon Managed Workflows for Apache Airflow のトラブル	2021 年 6 月 18 日

シューティング に追加されました。

- 踏み台ホストへの SSH トンネルで CLI コマンドを実行する例が Apache Airflow CLI トークンの作成 に追加されました。
- ランダムに生成されたユーザー名のトピックが Amazon Managed Workflows for Apache Airflow のトラブルシューティング に追加されました。
- CLI で DAG を実行する場合の 503 エラーに関するトピックが Amazon Managed Workflows for Apache Airflow のトラブルシューティング に追加されました。
- 各 Airflow プロセスの開始時にプラグインをロードしてバージョンのデフォルト設定を上書きして Amazon MWAA での Apache Airflow 構成オプションの使用 にする `core.lazy_load_plugins : False` のAirflow 設定オプションを必要とする、Apache Airflow v2.0.2 のカスタムプラグインに関するトピックをが追加されました。
- Apache Hive と Hadoop を使ったカスタムプラグイン

	の作成 での Apache Airflow v2.0.2 プラグインのサンプルコード用 Airflow 設定オプションステップが追加されました。 • ランタイム環境変数を生成するカスタムプラグインを作成します での Apache Airflow v2.0.2 プラグインのサンプルコード用 Airflow 設定オプションステップが追加されました。 • Apache Airflow Python VirtualEnv オペレータ用のカスタムプラグインを作成する での Apache Airflow v2.0.2 プラグインのサンプルコード用 Airflow 設定オプションステップが追加されました。 • Oracle でのカスタムプラグインの作成 での Apache Airflow v2.0.2 プラグインのサンプルコード用 Airflow 設定オプションステップが追加されました。	
新しいサンプルコード	次の変更が追加されました。 • Apache Airflow Snowflake 接続のサンプルコードが Apache Airflow Snowflake 接続において、AWS Secrets Manager でのシークレットキーを使用する に追加されました。	2021 年 6 月 18 日

新しいトピックとユースケース	次の変更が追加されました。 • サーバー側の暗号化ガイダンスが Amazon MWAA 用の Amazon S3 バケットの作成 に追加されました。 • Apache Airflow v2.0.2 のシークレットバックエンドが AWS Secrets Manager シークレットを使用した Apache Airflow 接続の設定 に追加されました。 • Apache Airflow ワーカーのクォータ増加リクエストに関する質問が Amazon MWAA に関するよくある質問 に追加されました。 • Apache Airflow Workers をスケールするかどうかの決定にどのメトリックが使用されるのかという質問が Amazon MWAA に関するよくある質問 に追加されました。 • でカスタムメトリクスを作成するための質問が CloudWatch に追加されましたAmazon MWAA に関するよくある質問。 • プライベートルーティングを使用する VPC の Amazon S3 VPC インターフェイスエンドポイントのプライベート IP アドレスを有効にする手順をAmazon VPC に	2021 年 6 月 2 日

	必要な VPC サービスエンドポイントをプライベートルーティングで作成する に追加されました。 • チュートリアル: Linux 踏み台ホストを使用したプライベートネットワークアクセスの設定 でローカルポートフォワーディングを使用して SSH トンネルをセットアップするには	
新しいサンプルコード	次の変更が追加されました。 • Amazon Aurora PostgreSQL メタデータデータベースをクエリし、カスタムメトリクスを CloudWatch で Amazon に発行する DAG のサンプルコードを追加しましたDAG を使用して CloudWatch にカスタムメトリクスを書き込む。	2021 年 6 月 2 日
新しいガイド	次の変更が追加されました。 • 接続タイプの概要 の Apache Airflow UI で接続テンプレートを互換的に使用する方法に関するガイドを作成しました。	2021 年 6 月 2 日

修正内容	次の変更が追加されました。 • オプション 3: へのインターネットアクセスのない VPC ネットワークの作成の AWS CloudFormation テンプレートに Apache Airflow VPC エンドポイントを追加しましたVPCネットワークを作成する。	2021 年 6 月 2 日

Apache Airflow v2.0.2 をスタートアップ	Apache Airflow v2.0.2 の一般提供開始 • Apache Airflow 用 Amazon マネージドワークフロー上の Apache Airflow のバージョン が作成されました。 • の Apache Airflow v2 環境メトリクス CloudWatch が作成されました。 • Apache Airflow v2.0.2 のバージョン固有のリンクが Amazon MWAA での Apache Airflow 構成オプションの使用 に追加されました。 • Apache Airflow v2.0.2 バージョン固有のガイダンスが Python 依存関係のインストール に追加されました。 • Apache Airflow v2.0.2 バージョン固有のガイダンスが requirements.txt での Python 依存関係の管理 に追加されました。 • Apache Airflow v2.0.2 のサンプルプラグインが カスタムプラグインのインストール に追加されました。 • Apache Airflow v2.0.2 のサンプルコードが Amazon MWAA 環境でのAurora PostgreSQL データベースのクリーンアップ に追加されました。	2021 年 5 月 26 日

- Apache Airflow v2.0.2 のサンプルコードが AWS Secrets Manager の Apache Airflow 接続でのシークレットキーの使用 に追加されました。
- Apache Airflow v2.0.2 のサンプルコードが Apache Airflow Python VirtualEnv オペレータ用のカスタムプラグインを作成する に追加されました。
- Apache Airflow v2.0.2 コマンドが Apache Airflow CLI コマンドリファレンス に追加されました。
- Apache Airflow v2.0.2 スクリプトが Apache Airflow CLI トークンの作成 に追加されました。
- Amazon MWAA はデフォルトで最新の Apache Airflow バージョンを使用するという注記が Amazon MWAA 環境を作成する に追加されました。

新しいトピックとユースケース	次の変更が追加されました。 • Airflow タスクが行き詰まっている、または実行されない場合のトラブルシューティングに関するガイダンスが Amazon Managed Workflows for Apache Airflow のトラブルシューティング に追加されました。	2021 年 5 月 14 日
修正内容	次の変更が追加されました。 • Apache Hive と Hadoop を使ったカスタムプラグインの作成 の最新の Java バージョンを使用するようにサンプルプラグインコードが 更新されました。以前は、`os.environ["JAVA_HOME"]="/usr/lib/jvm/jre-1.8.0-openjdk-1.8.0.272.b10-1.amzn2.0.1.x86_64"` でした。	2021 年 5 月 12 日
削除または移動されたトピック	次の変更が追加されました。 • Amazon Managed Workflows for Apache Airflow のトラブルシューティング のトピックがカテゴリごとに新しいページに移動されました。	2021 年 5 月 10 日

新しいトピックとユースケース	次の変更が追加されました。 • Amazon S3 バケットの概要が Amazon MWAA での DAG の取り扱い に追加されました。	2021 年 5 月 10 日
削除または移動されたトピック	次の変更が追加されました。 • Apache Airflow へのアクセス がトップレベルのナビゲーションに移動され、Apache Airflow ウェブサーバーアクセストークンを作成する、Apache Airflow CLI トークンの作成、および Apache Airflow CLI コマンドリファレンス のページが追加されました。	2021 年 5 月 7 日

新しいトピックとユースケース	次の変更が追加されました。 • Apache Airflow CLI コマンドリファレンス でサポートされている、またはサポートされていないすべての Airflow CLI コマンドに関する Apache Airflow リファレンスガイドにバージョン固有のリンクが追加されました。 • Amazon MWAA での Apache Airflow 構成オプションの使用 のすべての設定オプションに関する Apache Airflow リファレンスガイドにバージョン固有のリンクが追加されました。 • Amazon MWAA CLI ユーティリティが requirements.txt での Python 依存関係の管理 に追加されました。	2021 年 5 月 7 日

新しいトピックとユースケース	次の変更が追加されました。 • カスタムプラグインのインストール で、plugins.zip を構造化する方法のフラットおよびネストされた例が追加されました。 • DAG の追加と更新、カスタムプラグインのインストールとPython 依存関係のインストールページに Amazon MWAA CLI ユーティリティが追加されました。 • カスタムプラグインのインストール および Python 依存関係のインストール ページでのユーザーフィードバックに基づいて、コンテンツを概要に再構成し、Amazon S3 にアップロードし、Amazon MWAA セクションにインストールしました。 • Amazon MWAA でのネットワーキングについて でインターネットにアクセスせずに、必要な VPC エンドポイントを作成して既存の Amazon VPC にアタッチするユースケースの例が追加されました。	2021 年 4 月 30 日

新しいサンプルコード	次の変更が追加されました。 • Apache Airflow 変数の AWS Secrets Manager におけるシークレットキーの使用 でApache Airflow の変数に Secrets Managerでのシークレットキーを使用するサンプルコードが追加されました。	2021 年 4 月 30 日
新しいガイド	次の変更が追加されました。 • Amazon VPC に必要な VPC サービスエンドポイントをプライベートルーティングで作成する が作成されました。	2021 年 4 月 30 日
修正内容	次の変更が追加されました。 • おっと! Amazon MWAA での Apache Airflow 構成オプションの使用 で `core.default_ui_timezone` が `webserver.default_ui_timezone` にアップデートされました。	2021 年 4 月 30 日

新しいトピックとユースケース	次の変更が追加されました。 • チュートリアル: Linux 踏み台ホストを使用したプライベートネットワークアクセスの設定への SSH トンネル用の Windows (PuTTY) ステップが追加されました。 • Apache Airflow 2.0とのみ互換性のある `apache-airflow-providers-amazon` のためのトピックが Amazon Managed Workflows for Apache Airflow のトラブルシューティング に追加されました。	2021 年 4 月 23 日
新しいサンプルコード	次の変更が追加されました。 • AWS Secrets Manager の Apache Airflow 接続でのシークレットキーの使用 で Apache Airflow 接続に Secrets Manager のシークレットキーを使用するサンプルコードが追加されました。	2021 年 4 月 23 日

新しいガイド	次の変更が追加されました。 • Amazon MWAA でのネットワーキングについて が作成されました。 • Amazon MWAA の VPC のセキュリティ が作成されました。 • Amazon MWAA のサービス固有の Amazon VPC エンドポイントへのアクセスの管理 が作成されました。	2021 年 4 月 23 日

新しいトピックとユースケース	次の変更が追加されました。 • でインターネットにアクセスせずに Amazon VPC ネットワークを作成するための新しい AWS CloudFormation テンプレートを追加しましたVPCネットワークを作成する。 • で を作成するための新しいチュートリアルを追加 AWS Client VPN しましたチュートリアル: AWS Client VPN を使用したプライベートネットワークアクセスの設定。 • ユーザーからのフィードバックに基づいて、ネットワークアクセスページの名前を Apache Airflow アクセスモードに変更され、Apache Airflow のアクセスモード のドキュメントが効率化されました。 • ユーザーフィードバックに基づいて、Amazon VPC の入門情報とテンプレートのみを含むように VPCネットワークを作成する のドキュメントが 簡素化されました。 • BigQuery オペレーターの回避策を に追加しましたAmazon Managed Workflows for Apache	2021 年 4 月 16 日

	Airflow のトラブルシューティング。 • Apache Airflow v1.10.12 制約ファイルのベストプラクティスが Python 依存関係のインストール に追加されました。	
新しいサンプルコード	次の変更が追加されました。 • Oracle でのカスタムプラグインの作成 で Oracleを使用してカスタムプラグインを作成するためのサンプルコードが追加されました。 • ランタイム環境変数を生成するカスタムプラグインを作成します でランタイム環境変数を生成するカスタムプラグインを作成するサンプルコードが追加されました。 •	2021 年 4 月 16 日

新しいトピックとユースケース	次の変更が追加されました。 • VPC セキュリティグループの自己参照ルール要件のトピックが Amazon MWAA に関するよくある質問 に追加されました。 • カスタムプラグインディレクトリとサイズ制限が カスタムプラグインのインストール に追加されました。 • 要件ディレクトリとサイズ制限が Python 依存関係のインストール に追加されました。 • requirements.txt での Python 依存関係の管理 で `foo.pass` および `foo.user` の Apache Airflow 構成オプションが明確にされました。 • 設定オプションの概要が Amazon MWAA での Apache Airflow 構成オプションの使用 に追加されました。	2021 年 4 月 9 日

新しいサンプルコード	次の変更が追加されました。 • で を使用してカスタムプラグインを作成するためのサンプルコードを追加 PythonVirtualenvOperator しましたApache Airflow Python VirtualEnv オペレータ用のカスタムプラグインを作成する。 • Apache Hive と Hadoop を使ったカスタムプラグインの作成 で Apache Hive と Hadoop を使用してカスタムプラグインを作成するためのサンプルコードが追加されました。	2021 年 4 月 9 日
修正内容	次の変更が追加されました。 • おっと! Python 依存関係のインストール で requirements.txt のフォーマットを更新し、Apache Airflow v1.10.12 と互換性のあるサンプルが追加されました。	2021 年 3 月 31 日

新しいトピックとユースケース	次の変更が追加されました。 • Amazon MWAA に関するよくある質問にrequirements.txt や plugins.zip を削除するための回避策が追加されました。 • 環境上の SSH 用の bash 回避策が Amazon MWAA に関するよくある質問 に追加されました。 • CloudTrail ResourceAlreadyExistsException エラーのトピックが に追加されましたAmazon Managed Workflows for Apache Airflow のトラブルシューティング。	2021 年 3 月 26 日

新しいトピックとユースケース	次の変更が追加されました。 • AWS で使用されるサービスのリストを追加しましたAmazon MWAA 実行ロール。 • AWS で使用されるサービスのリストを追加しましたAmazon MWAA のサービスにリンクされたロール。 • Amazon MWAA 用の Python 3.7 バージョンに関する質問が Amazon MWAA に関するよくある質問 に追加されました。 • に に関する質問を追加 PythonVirtualenvOperator しましたAmazon MWAA に関するよくある質問。 • Amazon Managed Workflows for Apache Airflow のトラブルシューティング での VPC と環境設定に関連するすべてのトピックについて、次のステップとしてトラブルシューティングスクリプトが追加されました。 • Linux bastion は チュートリアル: Linux 踏み台ホストを使用したプライベートネットワークアクセスの設定 の環境と同じリージョンにある必要があるというドキュメントを明確にしました。	2021 年 3 月 19 日

新しいガイド	次の変更が追加されました。 • で 用の Apache Airflow 接続ガイドを作成 AWS Secrets Manager しましたAWS Secrets Manager シークレットを使用した Apache Airflow 接続の設定。 • テンプレートを使用して、で Amazon VPC インフラストラクチャ、Amazon S3 バケット、および Amazon MWAA 環境を作成 AWS CloudFormation するためのクイックスタートチュートリアルを作成しましたAmazon Managed Workflows for Apache Airflow のクイックスタートチュートリアル。	2021 年 3 月 19 日
新しいトピックとユースケース	次の変更が追加されました。 • Amazon S3 バケット作成のトラブルシューティングトピックが Amazon Managed Workflows for Apache Airflow のトラブルシューティング に追加されました。 • JSON ポリシーを作成してにアタッチする手順が Amazon MWAA 実行ロール に追加されました。	2021 年 3 月 12 日

新しいサンプルコード	次の変更が追加されました。 • DAG をトリガーする際に構成を追加するサンプルコードが Apache Airflow へのアクセス に追加されました。	2021 年 3 月 12 日
新しいガイド	次の変更が追加されました。 • ベストプラクティスガイドが requirements.txt での Python 依存関係の管理 で作成されました。	2021 年 3 月 12 日

新しいトピックとユースケース	次の変更が追加されました。 • Google/GCP/BigQuery トラブルシューティングのトピックを に追加しましたAmazon Managed Workflows for Apache Airflow のトラブルシューティング。 • Cython のトラブルシューティングトピックが Amazon Managed Workflows for Apache Airflow のトラブルシューティング に追加されました。 • MySQL のトラブルシューティングトピックが Amazon Managed Workflows for Apache Airflow のトラブルシューティング に追加されました。 • 5xx Web サーバーエラーのトラブルシューティングトピックが Amazon Managed Workflows for Apache Airflow のトラブルシューティング に追加されました。	2021 年 3 月 5 日

現在サポート中	次の変更が追加されました。 • 以前は、 backend_kwargs は でサポートされておらず、Secrets Manager 関数呼び出しを上書きするための回避策 AWS Secrets Manager が必要でした。現在、backend_kwargs はサポートされています。の AWS Secrets Manager トラブルシューティングトピックを参照してくださいAmazon Managed Workflows for Apache Airflow のトラブルシューティング。	2021 年 3 月 4 日
修正内容	次の変更が追加されました。 • おっと! Amazon MWAA 環境クラスの構成 で各環境クラスのサイズを実際のギガバイトに反映するように更新しました。	2021 年 3 月 4 日

新しいトピックとユースケース	次の変更が追加されました。 • VPC エンドポイントポリシーを使用したプライベートネットワークアクセスが Apache Airflow のアクセスモード に追加されました。 • 環境を作成する際のトラブルシューティングトピックに追加のチェックが Amazon Managed Workflows for Apache Airflow のトラブルシューティング に追加されました。 • `requirements.txt` の口グを表示する手順が Python 依存関係のインストール に追加されました。	2021 年 2 月 26 日

新しいトピックとユースケース	次の変更が追加されました。 • Apache Hive のユースケースが Python 依存関係のインストール に追加されました。 • `requirements.txt` の Python 依存関係のインストール ファイルに Apache Airflow パッケージの必要な依存関係を含める必要があることがドキュメントで明確にされました。 • requirements.txt の更新に関するトラブルシューティングのトピックが Amazon Managed Workflows for Apache Airflow のトラブルシューティング に追加されました。	2021 年 2 月 25 日
新規チュートリアル	次の変更が追加されました。 • プライベートネットワークのチュートリアルが チュートリアル: Linux 踏み台ホストを使用したプライベートネットワークアクセスの設定 に追加されました。	2021 年 2 月 22 日

新しいトピックとユースケース	次の変更が追加されました。 • プライベートネットワーク構成とパブリックネットワーク構成が Apache Airflow のアクセスモード に追加されました。 • 開発グループのユースケースとユーザーシナリオが Amazon MWAA 実行ロール に追加されました。	2021 年 2 月 22 日
新しいサンプルコード	次の変更が追加されました。 • Web ログイントークンと CLI トークンのサンプル Python スクリプトが Apache Airflow へのアクセス に追加されました。 • 別の環境で DAG をトリガーするサンプルコードが Amazon Managed Workflows for Apache Airflow に追加されました。 • Lambda 関数を使用して DAG をトリガーするサンプルコードが Lambda 関数を使用して DAG を呼び出す に追加されました。	2021 年 2 月 22 日
新しいコマンドとプロシージャ	次の変更が追加されました。 • Apache Airflow へのアクセス でのすべてのスクリプトに手順が段階的に追加されました。	2021 年 2 月 22 日

新しいサンプルコード	次の変更が追加されました。 • Apache Airflow へのアクセス での Web ログイントークンのためのcurlの例が更新されました。 • Amazon RDS Microsoft SQL Server に接続するためのサンプルコードが Amazon RDS for Microsoft SQL Server で Amazon MWAA を使用する に追加されました。	2021 年 2 月 17 日

新しいコマンドとプロシージャ	次の変更が追加されました。 • Amazon MWAA での DAG の取り扱い ページに AWS CLI コマンドを追加しました。 • Apache Airflow は CLI コマンドでのシリアル化された DAG をサポートしていません。CLI は、セキュリティ上の理由からプラグインや要件がないウェブサーバー上で実行されるため、plugins.zip または requirements.txt を使用する MWAA 環境ではこれらのコマンドはサポートされません。Apache Airflow `list_dags` と `backfill` コマンドが、Apache Airflow へのアクセス でサポートされていないコマンドに移動されました。	2021 年 2 月 17 日
GitHub 起動	ユーザーガイドのドキュメントが でオープンソースになりました GitHub。任意のページの「このページの編集 GitHub」を選択します。	2021 年 2 月 17 日

新しいトピックとユースケース	次の変更が追加されました。 • Step Functions と Amazon MWAA ユースケースに関する質問が Amazon Managed Workflows for Apache Airflow のトラブルシューティング に追加されました。 • CLI アクセスポリシーが Amazon MWAA 環境へのアクセス に追加されました。 • 任意のサポートされている Apache Airflow 構成オプションが Amazon MWAA での Apache Airflow 構成オプションの使用 で指定できることがドキュメントで明確にされました。 • VPCネットワークを作成する で Fargate コンテナが1つのアベイラビリティゾーンで失敗した場合、MWAA は異なるアベイラビリティゾーンの他のコンテナに切り替わることがドキュメントで明確にされました。	2021 年 2 月 12 日
新しいトピックとユースケース	次の変更が追加されました。 • Amazon MWAA 環境クラスの構成 が追加されました。	2021 年 2 月 5 日

削除または移動されたトピック	次の変更が追加されました。 • Amazon Managed Workflows for Apache Airflowを使い始める において、Amazon S3 バケット名が `airflow-` で始まる必要がないように要件が削除されました。 • Amazon MWAA 環境へのアクセス と Amazon MWAA 実行ロールが Amazon MWAA 環境へのアクセスの管理 に移動されました。	2021 年 2 月 4 日
Amazon MWAA CloudFormation	パラメータを更新して、Amazon MWAA CloudFormationで環境を作成します。 • を削除します SubnetList。 • を削除します TagList。 • を追加します NetworkConfiguration。 • を追加します TagMap。 • 環境作成リクエストの例が追加されました。	2021 年 2 月 4 日

新しいトピックとユースケース	次の変更が追加されました。 • メール設定の例が Amazon MWAA での Apache Airflow 構成オプションの使用 に追加されました。 • PostgresHook トラブルシューティングトピックを に追加しましたAmazon Managed Workflows for Apache Airflow のトラブルシューティング。 • AWS Secrets Manager トラブルシューティングトピックを に追加しましたAmazon Managed Workflows for Apache Airflow のトラブルシューティング。 • 高性能ユースケースが Amazon MWAA ワーカーの自動スケーリングの設定 に追加されました。	2021 年 1 月 29 日
Amazon MWAA のローンチ	Amazon Managed Workflows for Apache Airflow の一般提供が開始されました。 • ユーザーガイドのドキュメント • AWS CloudFormation ドキュメント	2020 年 11 月 24 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。