aws

用户指南

Amazon Managed Workflows for Apache Airflow

Amazon Managed Workflows for Apache Airflow: 用户指南

Table of Contents

什么是 Amazon Managed Workflows for Apache Airflow？

Amazon MWAA 是 Apache Airflow 的托管式服务，可以用来在云中大规模设置和运行数据管道。Apache Airflow 是一种开源工具，用于创建、安排和监视工作流程。

借助 Amazon MWAA，您可以使用 Apache Airflow 和 Python 来创建工作流程，而无需管理基础设施以实现可扩展性、可用性和安全性。Amazon MWAA 会自动扩展以满足您的工作流程需求。它与 AWS 安全服务集成，可快速、安全地访问您的数据。

内容

功能

查看以下功能，了解 Amazon MWAA 如何简化您的 Apache Airflow 工作流程的管理。

- 自动 Airflow 设置 — 在创建 Amazon MWAA 环境时，通过选择 Apache Airflow 版本来快速设置 Apache Airflow。Amazon MWAA 使用与互联网上相同的 Apache Airflow 用户界面和开源代码为您设置 Apache Airflow。
- 自动扩展 — 通过设置最小和最大限制，自动扩展 Apache Airflow 工作线程（运行任务的计算资源）。Amazon MWAA 会监控环境中的工作线程，并使用其自动扩缩组件添加工作线程以满足需求，直至达到您定义的最大工作线程数。
- 内置身份验证 — 通过在 (IAM) 中定义访问控制策略，为您的 Apache Airflow 网络服务器启用基于角色的身份验证和授权。 AWS Identity and Access Management Apache Airflow 工作人员采用这些策略来安全访问服务。 AWS
- 内置安全性 — Apache Airflow 工作线程和计划程序在 Amazon MWAA 的 Amazon VPC 中运行。数据还会使用自动加密 AWS Key Management Service，因此您的环境在默认情况下是安全的。
- 公有或私有访问模式 — 使用私有或公有访问模式访问 Apache Airflow Web 服务器。公有网络访问模式使用可通过互联网访问的 Apache Airflow Web 服务器的 VPC 端点。私有网络访问模式使用可在 VPC 中访问的 Apache Airflow Web 服务器的 VPC 端点。在这两种情况下，您的 Apache

Airflow 用户的访问权限都由您 AWS Identity and Access Management 在 (IAM) 和 SSO 中定义的访问控制策略进行控制。 AWS

- 简化的升级和补丁 — Amazon MWAA 会定期提供新版本的 Apache Airflow。Amazon MWAA 团队将更新和修补这些版本的映像。
- 工作流程监控 — 无需其他第三方工具即可访问亚马逊中的 Apache Airflow 日志和 Apache Airflow 指标，即可 CloudWatch 识别 Apache Airflow 任务延迟或工作流程错误。Amazon MWAA 会自动向发送环境指标（如果已启用）Apache Airflow 日志。 CloudWatch
- AWS 集成 — 亚马逊 MWAA 支持与亚马逊 Athena、亚马逊、亚马逊 DynamoDB AWS Batch、亚马逊 EMR、 CloudWatch亚马逊 EKS、Amaz AWS DataSync on Data Firehose、、、 AWS Fargate Amazon Redshi AWS Glue ft、亚马逊 SQS、亚马逊 SNS、 AWS Lambda亚马逊 AI 和亚马逊 S3 的开源集成，以及数百个内置和社区创建的操作员和传感器。 SageMaker
- Worker 实例集 — Amazon MWAA 支持使用容器按需扩展 Worker 实例集，并使用 AWS Fargate 上的 Amazon ECS 减少计划程序中断。支持在 Amazon ECS 容器上调用任务的运算符，以及在 Kubernetes 集群上创建和运行 Pod 的 Kubernetes 运算符。

架构

外箱中包含的所有组件（如下图所示）在您的账户中显示为单个 Amazon MWAA 环境。Apache Airflow 调度程序和工作程序 AWS Fargate 是连接到您环境的 Amazon VPC 中私有子网的容器。每个环境都有自己的 Apache Airflow 元数据数据库，由该数据库管理，调度程序和工作人员 Fargate 容器可 AWS 通过私有保护的 VPC 端点访问该元数据库。

亚马逊 CloudWatch、亚马逊 S3、亚马逊 SQS 和与亚马逊 MWAA AWS KMS 是分开的，需要可以从 Fargate 容器中的 Apache Airflow 调度程序和 Fargate 容器中的工作人员进行访问。多个 Apache Airflow 计划程序仅在 Apache Airflow v2 及更高版本中可用。要详细了解 Apache Airflow 任务生命周期，请参阅《Apache Airflow 参考指南》的概念。

要访问 Apache Airflow Web 服务器，您可以选择公有网络 Apache Airflow 访问模式通过互联网访问，或选择私有网络 Apache Airflow 访问模式在 VPC 内访问。在这两种情况下，您的 Apache Airflow 用户的访问权限都由您在 (IAM) AWS Identity and Access Management 中定义的访问控制策略控制。

> **Note**
>
> 从 Apache Airflow v3 开始，Amazon MWAA Web 服务器还托管 Apache Airflow 的执行 API 服务器。

集成

活跃且不断壮大的Apache Airflow开源社区为Apache Airflow提供了与服务集成的运营商（简化服务连接的插件）。 AWS 这包括亚马逊 S3、Amazon Redshift、亚马逊 EMR AWS Batch和亚马逊 A SageMaker I 等服务，以及其他云平台上的服务。

将 Apache Airflow 与亚马逊 MWAA 配合使用完全支持与服务和流行的第三方工具（例如 Apache Hadoop、Presto、Hive 和 Spark）AWS 集成以执行数据处理任务。亚马逊 MWAA 致力于保持与 Apache Airflow API 的兼容性，亚马逊 MWAA 打算为服务提供可靠的 AWS 集成并将其提供给社区，并参与社区功能开发。

有关示例代码，请参阅 Amazon MWAA 的代码示例。

支持的版本

Amazon MWAA 支持多个版本的 Apache Airflow。有关我们支持的 Apache Airflow 版本以及每个版本中包含的 Apache Airflow 组件的更多信息，请参阅 Amazon MWAA 上的 Apache Airflow 版本。

接下来做什么？

- 从一个 CloudFormation 模板开始，该模板可为您的 Airflow DAGs 和支持文件创建 Amazon S3 存储桶、具有公共路由功能的 Amazon VPC，并在中创建 Amazon MWAA 环境。Amazon MWAA 的快速入门教程
- 通过为您的 Airflow DAGs 和支持文件创建 Amazon S3 存储桶，从三个 Amazon VPC 联网选项中进行选择，然后在中创建 Amazon MWAA 环境，逐步入门。开始使用 Amazon MWAA

Amazon MWAA 的快速入门教程

本快速入门教程使用 AWS CloudFormation 模板来同时创建 Amazon VPC 基础设施、带 dags 文件夹的 Amazon S3 存储桶和 Amazon MWAA 环境。

主题

在本教程中：

根据本教程使用三个 AWS Command Line Interface (AWS CLI) 命令，将 DAG 上传到 Amazon S3、在 Apache Airflow 中运行 DAG 以及在 CloudWatch 中访问日志。最后，您将学习为 Apache Airflow 开发团队创建 IAM 策略。

> **Note**
>
> 此页面上的 CloudFormation 模板为 CloudFormation 中可用的最新版本 Apache Airflow 创建了 Amazon MWAA 环境。可用的最新版本是 Apache Airflow v3.0.6。

CloudFormation 模板创建以下各项：

- VPC 基础设施。模板使用 通过互联网进行公共路由。它为 `WebserverAccessMode: PUBLIC_ONLY` 中的 Apache Airflow Web 服务器使用 公有网络访问模式。
- Amazon S3 桶。模板将创建带有 dags 文件夹的 Amazon S3 存储桶。将其配置为在启用存储桶版本控制的情况下阻止所有公共访问，如 为 Amazon MWAA 创建 Amazon S3 存储桶。 中所定义。
- Amazon MWAA 环境。该模板创建了与 Amazon S3 存储桶上的 dags 文件夹关联的 Amazon MWAA 环境、拥有 Amazon MWAA 所用的 AWS 服务权限的执行角色以及使用 AWS 自有密钥进行加密的默认角色，如 创建 Amazon MWAA 环境 中所定义。

- CloudWatch 日志 该模板启用 CloudWatch 中的信息及以上级别的 Apache Airflow 日志，用于 Airflow 计划程序日志组、Airflow Web 服务器日志组、Airflow 工作线程日志组、Airflow DAG 处理日志组和 Airflow 任务日志组，如 访问 Amazon 中的 Airflow 日志 CloudWatch 中所定义。

在本教程中，您将完成以下任务：

- 上传并运行 DAG。将 Amazon MWAA 支持的最新 Apache Airflow 版本的 Apache Airflow 教程 DAG 上传到 Amazon S3，然后在 Apache Airflow UI 中运行，如 添加或更新 DAG 中所定义。
- 访问日志：在 CloudWatch Logs 中访问 Airflow Web 服务器日志组，如 访问 Amazon 中的 Airflow 日志 CloudWatch 中所定义。
- 创建访问控制策略。在 IAM 中为 Apache Airflow 开发团队创建访问控制策略，如 访问 Amazon MWAA 环境 中所定义。

> **Note**
>
> 在托管 Amazon MWAA 环境的 VPC 中，将所有连接的子网的 `assignIpv6AddressOnCreation` 设置为 `true`。此设置可确保自动将 Internet 协议版本 6 (IPv6) 地址分配给这些子网中的资源。

先决条件

AWS Command Line Interface (AWS CLI) 是一种开源工具，您可以用来在命令行 Shell 中使用命令与 AWS 服务进行交互。要完成本节中的步骤，您需要以下满足以下条件：

- AWS CLI – 安装版本 2。
- AWS CLI – 使用 `aws configure` 进行快速配置。

步骤 1：将 CloudFormation 模板保存到本地

- 复制以下模板的内容并将其作为 `mwaa-public-network.yml` 保存在本地中。您也可以使用下载模板。

```
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

  MwaaExecutionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: 2012-10-17&TCX5-2025-waiver;
        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&TCX5-2025-waiver;
        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}"
```

步骤 2：使用 AWS CLI 创建堆栈

1. 在命令提示符下，导航到存储 mwaa-public-network.yml 的目录。例如：

```
cd mwaaproject
```

2. 输入 aws cloudformation create-stack 命令来使用 AWS CLI 创建堆栈。

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

> **Note**
>
> 创建 Amazon VPC 基础设施、Amazon S3 存储桶和 Amazon MWAA 环境需要 30 多分钟。

步骤 3：将 DAG 上传到 Amazon S3 并在 Apache Airflow UI 中运行

1. 复制支持的最新 Apache Airflow 版本的 tutorial.py 文件内容，然后在本地另存为 tutorial.py。

2. 在命令提示符下，导航到存储 `tutorial.py` 的目录。例如：

```
cd mwaaproject
```

3. 以下示例列出所有 Amazon S3 存储桶。

```
aws s3 ls
```

4. 使用以下命令列出 Amazon S3 存储桶中适合环境的文件和文件夹。

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

5. 使用以下脚本将 `tutorial.py` 文件上传到 `dags` 文件夹。替换 *amzn-s3-demo-bucket* 中的样本值。

```
aws s3 cp tutorial.py s3://amzn-s3-demo-bucket/dags/
```

6. 在 Amazon MWAA 控制台上打开环境页面。
7. 选择环境。
8. 选择打开 Airflow UI。
9. 在 Apache Airflow UI 上，从可用 DAG 列表中选择教程 DAG。
10. 在 DAG 详细信息页面上，选择 DAG 名称旁边的暂停/取消暂停 DAG 开关以取消暂停 DAG。
11. 选择触发 DAG。

步骤 4：在 CloudWatch Logs 中访问日志

您可以在 CloudWatch 控制台中访问 Apache Airflow 日志，了解 CloudFormation 堆栈启用的所有 Apache Airflow 日志。下一节介绍如何访问 Airflow Web 服务器日志组的日志。

1. 在 Amazon MWAA 控制台上打开环境页面。
2. 选择环境。
3. 在监控窗格上选择 Airflow Web 服务器日志组。
4. 在日志流中选择 `webserver_console_ip` 日志。

接下来做什么？

- 要详细了解如何上传 DAG，如何指定 `requirements.txt` 中的 Python 依赖项和 `plugins.zip` 中的自定义插件，请参阅 在 Amazon MWAA 上使用 DAG。
- 要详细了解我们推荐的调整环境性能的最佳实践，请参阅 Amazon MWAA 上的 Apache Airflow 的性能调整。
- 为环境创建监控面板，请参阅 监控 Amazon MWAA 上的控制面板和警报。
- 运行一些 DAG 代码示例，请参阅 Amazon MWAA 的代码示例。

开始使用 Amazon MWAA

Amazon MWAA 使用 Amazon S3 存储桶中的 Amazon VPC、DAG 文件和支持文件来创建环境。本章描述了开始使用 Amazon MWAA 所需的先决条件和所需 AWS 资源。

主题

先决条件

要创建 Amazon MWAA 环境，确保您有权访问创建所需的 AWS 资源。

- AWS 账户 — 有权使用 Amazon MWAA 以及环境所用的 AWS 服务和资源的 AWS 账户。

关于本指南

本指南介绍您将创建的 AWS 基础设施和资源。

- Amazon VPC — Amazon MWAA 环境所需的 Amazon VPC 网络组件。您可以配置满足这些（高级）要求的现有 VPC，如 关于在 Amazon MWAA 上联网 中所示，也可以创建 VPC 和网络组件，如 the section called “创建 VPC 网络” 中所述。
- Amazon S3 存储桶 — 用于存储 DAG 和关联文件（例如 `plugins.zip` 和 `requirements.txt`）的 Amazon S3 存储桶。Amazon S3 存储桶必须配置为阻止所有公开访问，并启用存储桶版本控制，如 为 Amazon MWAA 创建 Amazon S3 存储桶。 中所定义。
- Amazon MWAA 环境 — 配置了 Amazon S3 存储桶的位置、DAG 代码和任何自定义插件或 Python 依赖项的路径以及 Amazon VPC 及其安全组，如 创建 Amazon MWAA 环境 中所定义。

开始前的准备工作

要创建 Amazon MWAA 环境，您可以在创建环境之前采取额外步骤来创建和配置其他 AWS 资源。

要创建环境，您需要具备以下条件：

- AWS KMS 密钥 — 用于在环境中进行数据加密的 AWS KMS 密钥。您可以在 Amazon MWAA 控制台上选择默认选项以在创建环境时创建 AWS 自有密钥，也可以指定现有的由客户托管的密钥，该密钥具有访问您配置的环境所用的其他 AWS 服务的权限（高级）。要了解更多信息，请参阅 使用由客户托管的密钥进行加密。
- 执行角色 — 允许 Amazon MWAA 访问环境中的 AWS 资源的执行角色。创建环境时，您可以在 Amazon MWAA 控制台上选择默认选项来创建执行角色。要了解更多信息，请参阅 Amazon MWAA 执行角色。
- VPC 安全组 — 允许 Amazon MWAA 访问 VPC 网络中的其他 AWS 资源。您可以在创建环境时选择 Amazon MWAA 控制台上的默认选项来创建安全组，或者为安全组提供相应的入站和出站规则（高级）。要了解更多信息，请参阅 Amazon MWAA 上的 VPC 安全。

可用区

Amazon MWAA 在以下 AWS 区域 中可用。要了解有关每个区域（例如默认启用或禁用的区域）的更多信息，请参阅 AWS 区域。

代码	名称
us-east-1	美国东部（弗吉尼亚州北部）
us-east-2	美国东部（俄亥俄州）
us-west-1	美国西部（加利福尼亚北部）
us-west-2	美国西部（俄勒冈州）
af-south-1	非洲（开普敦）
ap-east-1	亚太地区（香港）
ap-south-2	亚太地区（海得拉巴）

代码	名称
ap-southeast-3	亚太地区（雅加达）
ap-southeast-5	亚太地区（马来西亚）
ap-southeast-4	亚太地区（墨尔本）
ap-south-1	亚太地区（孟买）
ap-northeast-3	亚太地区（大阪）
ap-northeast-2	亚太地区（首尔）
ap-southeast-1	亚太地区（新加坡）
ap-southeast-2	亚太地区（悉尼）
ap-northeast-1	亚太地区（东京）
ca-central-1	加拿大（中部）
ca-west-1	加拿大西部（卡尔加里）
eu-central-1	欧洲地区（法兰克福）
eu-west-1	欧洲地区（爱尔兰）
eu-west-2	欧洲地区（伦敦）
eu-south-1	欧洲地区（米兰）
eu-west-3	欧洲地区（巴黎）
eu-south-2	欧洲地区（西班牙）
eu-north-1	欧洲地区（斯德哥尔摩）
eu-central-2	欧洲（苏黎世）
il-central-1	以色列（特拉维夫）

代码	名称
me-south-1	中东（巴林）
me-central-1	中东（阿联酋）
sa-east-1	南美洲（圣保罗）

为 Amazon MWAA 创建 Amazon S3 存储桶。

本指南介绍了创建 Amazon S3 存储桶以存储 Apache Airflow 定向无环图 DAGs ()、文件中的自定义插件以及`plugins.zip`文件中的 Python 依赖项的步骤。`requirements.txt`

目录

开始前的准备工作

- 创建 Amazon S3 存储桶后，您无法更改存储桶名称。有关更多信息，请参阅《Amazon Simple Storage Service 用户指南》中的存储桶命名规则。
- 在启用存储桶版本控制的情况下，必须将用于 Amazon MWAA 环境的 Amazon S3 存储桶配置为阻止所有公共访问。
- 用于亚马逊 MWAA 环境的 Amazon S3 存储桶必须与亚马逊 MWAA 环境位于同一 AWS 区域 位置。要访问亚马逊 MWAA AWS 区域 的列表，请参阅中的亚马逊 MWAA 终端节点和配额。AWS 一般参考

创建存储桶。

本节介绍为环境创建 Amazon S3 存储桶的步骤。

创建存储桶

1. 登录 AWS 管理控制台 并打开 Amazon S3 控制台，网址为https://console.aws.amazon.com/s3/。
2. 选择创建桶。

3. 在 Bucket name（桶名称）中，输入符合 DNS 标准的桶名称。

   桶名称必须满足以下要求：

   - 在所有 Amazon S3 中是唯一的。
   - 长度必须介于 3 到 63 个字符之间。
   - 不包含大写字符。
   - 以小写字母或数字开头。

   > **Important**
   >
   > 避免在存储桶名称中包含敏感信息，如账号。存储桶名称在指向存储桶中的对象的位置中可用。 URLs

4. AWS 区域 在区域内选择一个。这必须与您的 Amazon MWAA 环境 AWS 区域 相同。

   - 请选择一个靠近您的区域，这样可最大程度地减少延迟和成本并满足法规要求。

5. 请选择阻止所有公有访问。
6. 在存储桶版本控制中选择启用。
7. 可选 - 标签。在标签中添加键值标签对以识别 Amazon S3 存储桶。例如 `Bucket：Staging`。
8. 可选-服务器端加密。您可以选择在 Amazon S3 存储桶上启用以下加密选项之一。

   a. 在服务器端加密中选择 Amazon S3 密钥（SSE-S3）以启用存储桶的服务器端加密。

   b. 选择AWS Key Management Service 密钥 (SSE-KMS) 以使用密 AWS KMS 钥对您的 Amazon S3 存储桶进行加密：

      i. AWS 托管密钥 (aws/s3)-如果您选择此选项，则可以使用由 Amazon MWAA 管理的AWS自有密钥，也可以指定客户管理的密钥来加密您的 Amazon MWAA 环境。

      ii. 从密钥中选择或输入 AWS KMS 密 AWS KMS 钥 ARN-如果您选择在此步骤中指定客户管理的密钥，则必须指定 AWS KMS 密钥 ID 或 ARN。AWS KMS A@@ mazon MWAA 不支持别名和多区域密钥。您指定的 AWS KMS 密钥还必须用于在您的 Amazon MWAA 环境中进行加密。

9. 可选 - 高级设置。如果要启用 Amazon S3 对象锁定：

   a. 选择高级设置、启用。

> **Important**
>
> 启用对象锁定将永久允许锁定此存储桶中的对象。要了解更多信息，请参阅《Amazon Simple Storage Service 用户指南》中的使用 Amazon S3 对象锁定来锁定对象。

b. 选择确认。

10. 选择 创建存储桶 。

接下来做什么？

- 要了解如何为环境创建所需的 Amazon VPC 网络，请参阅 创建 VPC 网络。
- 要了解如何管理访问权限，请参阅如何设置 ACL 存储桶权限？
- 要了解如何删除存储桶，请参阅如何删除 S3 存储桶？。

创建 VPC 网络

Amazon MWAA 需要 Amazon VPC 和特定的网络组件来支持环境。本指南介绍了为 Amazon MWAA 环境创建 Amazon VPC 网络的不同选项。

> **Note**
>
> Apache Airflow 在低延迟网络环境中效果最好。如果您使用的是将流量路由到其他区域或本地环境的现有 Amazon VPC，我们建议您为 Amazon SQS、CloudWatch、Amazon S3 和 AWS KMS 添加 AWS PrivateLink 端点。有关为 Amazon MWAA 配置 AWS PrivateLink 的更多信息，请参阅创建没有互联网访问权限的 Amazon VPC 网络。

目录

先决条件

AWS Command Line Interface (AWS CLI) 是一种开源工具，您可以用来在命令行 Shell 中使用命令与 AWS 服务进行交互。要完成本节中的步骤，您需要以下满足以下条件：

- AWS CLI – 安装版本 2。
- AWS CLI – 使用 `aws configure` 进行快速配置。

开始前的准备工作

- 环境创建后您无法更改为环境指定的 VPC 网络。
- 您可以为 Amazon VPC 和 Apache Airflow Web 服务器使用私有或公共路由。要访问选项列表，请参阅 the section called “Amazon VPC 和 Apache Airflow 访问模式的示例用例”。

创建 Amazon VPC 网络的选项

下一节介绍可用于为环境创建 Amazon VPC 网络的选项。

> **Note**
>
> Amazon MWAA 不支持在美国东部（弗吉尼亚州北部）区域中使用 `use1-az3` 可用区（AZ）。在美国东部（弗吉尼亚州北部）区域中创建用于 Amazon MWAA 的 VPC 时，必须在 CloudFormation（CFN）模板中显式分配 `AvailabilityZone`。分配的可用区名称不得映射到 `use1-az3`。您可以通过运行以下命令来检索可用区名称与其对应可用区 ID 的详细映射：
>
> ```
> aws ec2 describe-availability-zones --region us-east-1
> ```

选项一：在 Amazon MWAA 控制台上创建 VPC 网络

下一部分说明如何在 Amazon MWAA 控制台上创建 VPC 网络。此选项使用 通过互联网进行公共路由。它可用于具有私有网络或公有网络访问模式的 Apache Airflow Web 服务器。

下图显示了在 Amazon MWAA 控制台上哪里可以找到创建 MWAA VPC 按钮。

选项二：创建可访问互联网的 Amazon VPC 网络

以下 CloudFormation 模板在默认 AWS 区域 中创建可访问互联网的 Amazon VPC 网络。此选项使用通过互联网进行公共路由。此模板可用于具有私有网络或公有网络访问模式的 Apache Airflow Web 服务器。

1. 复制以下模板的内容并将其作为 `cfn-vpc-public-private.yaml` 保存在本地中。您也可以使用下载模板。

```
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. 在命令提示符下，导航到存储 cfn-vpc-public-private.yaml 的目录。例如：

```
cd mwaaproject
```

3. 输入 aws cloudformation create-stack 命令来使用 AWS CLI 创建堆栈。

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

> **Note**
>
> 创建 Amazon VPC 基础设施需要大约 30 分钟。

选项三：创建不可访问互联网的 Amazon VPC 网络

以下 CloudFormation 模板将在默认 AWS 区域 中创建不可访问互联网的 Amazon VPC 网络。

此选项使用 无法访问互联网的私有路由。此模板可用于仅有私有网络访问模式的 Apache Airflow Web 服务器。它为环境使用的 AWS 服务创建所需的 VPC 端点。

1. 复制以下模板的内容并将其作为 cfn-vpc-private.yaml 保存在本地中。您也可以使用下载模板。

```
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


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. 在命令提示符下，导航到存储 `cfn-vpc-private.yml` 的目录。例如：

```
cd mwaaproject
```

3. 输入 [aws cloudformation create-stack](#) 命令来使用 AWS CLI 创建堆栈。

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

> **Note**
>
> 创建 Amazon VPC 基础设施需要大约 30 分钟。

4. 您需要创建一种机制，以便从计算机访问这些 VPC 端点。要了解更多信息，请参阅 [在 Amazon MWAA 上管理对服务特定 Amazon VPC 端点的访问](#)。

> **Note**
>
> 您可以在 Amazon MWAA 安全组的 CIDR 中进一步限制出站访问。例如，您可以通过添加自引用出站规则、Amazon S3 的前缀列表和 Amazon VPC 的 CIDR 来限制自身。

接下来做什么？

- 要了解如何创建 Amazon MWAA 环境，请参阅 创建 Amazon MWAA 环境。
- 要了解如何使用私有路由创建从计算机到 Amazon VPC 的 VPN 隧道，请参阅 教程：使用配置私有网络访问权限 AWS Client VPN。

创建 Amazon MWAA 环境

Amazon Managed Workflows for Apache Airflow 使用与 Apache 相同的开源 Apache Airflow 和用户界面，在您所选版本的环境中设置 Apache Airflow。本指南介绍创建 Amazon MWAA 环境的步骤。

目录

开始前的准备工作

- 环境创建后，您将无法修改为环境指定的 VPC 网络。
- 您需要将 Amazon S3 存储桶配置为阻止所有公开访问并启用存储桶版本控制。
- 您需要 AWS 账户 具有使用 Amazon MWAA 的权限，以及 AWS Identity and Access Management (IAM) 中的权限才能创建 IAM 角色。如果您为 Apache Airflow Web 服务器选择了私有网络访问模式，由于该模式会限制 Amazon VPC 内的 Apache Airflow 访问权限，因此您需要 IAM 中的权限才能创建 Amazon VPC 端点。

> **Note**
>
> Amazon MWAA 会在创建过程中动态确定网络。如果您使用 IPv6 子网，Amazon MWAA 会创建与数据库和 Web 服务器的 IPv6 私有链路连接。Amazon MWAA 不支持在网络类型之间过渡，也无法将现有环境升级到 IPv6。

Apache Airflow 版本

Amazon MWAA 上支持以下 Apache Airflow 版本。

> **Note**
>
> - 自 2025 年 12 月 30 日起，Amazon MWAA 将终止对 Apache Airflow 版本 v2.4.3、v2.5.1 和 v2.6.3 的支持。有关更多信息，请参阅Apache Airflow 版本支持和常见问题。
> - 从 Apache Airflow v2.2.2 开始，Amazon MWAA 支持直接在 Apache Airflow Web 服务器上安装 Python 要求、提供程序包和自定义插件。
> - 从 Apache Airflow v2.7.2 开始，要求文件必须包含一条 `--constraint` 语句。如果您未提供约束条件，Amazon MWAA 将为您指定一个约束条件，以确保您的要求中列出的程序包与您正在使用的 Apache Airflow 版本兼容。
>
>   有关在需求文件中设置约束条件的更多信息，请参阅安装 Python 依赖项。

Apache Airflow 版本	Apache Airflow 发布日期	Amazon MWAA 上市日期	Apache Airflow 约束条件	Python 版本
v3.2.1	2026年4月22日	2026年5月19日	v3.2.1 约束文件	Python 3.12
v2.11.0	2025年5月20日	2026年1月7日	v2.11.0 约束文件	Python 3.12
v3.0.6	2025 年 8 月 29 日	2025 年 10 月 1 日	v3.0.6 约束文件	Python 3.12
v2.10.3	2024 年 11 月 4 日	2024 年 12 月 18 日	v2.10.3 约束文件	Python 3.11

Apache Airflow 版本	Apache Airflow 发布日期	Amazon MWAA 上市日期	Apache Airflow 约束条件	Python 版本
v2.10.1	2024 年 9 月 5 日	2024 年 9 月 26 日	v2.10.1 约束文件	Python 3.11
v2.9.2	2024 年 6 月 10 日	2024 年 7 月 9 日	v2.9.2 约束文件	Python 3.11
v2.8.1	2024 年 1 月 19 日	2024 年 2 月 23 日	v2.8.1 约束文件	Python 3.11
v2.7.2	2023 年 10 月 12 日	2023 年 11 月 6 日	v2.7.2 约束文件	Python 3.11

有关迁移自管理的 Apache Airflow 部署或迁移现有 Amazon MWAA 环境的更多信息，包括备份元数据数据库的说明，请参阅 Amazon MWAA 迁移指南。

创建环境

下一节介绍创建 Amazon MWAA 环境的步骤。

步骤 1：指定详细信息

要指定环境的详细信息，请执行以下操作

1. 打开 Amazon MWAA 控制台。
2. 选择你的 AWS 区域。
3. 选择创建环境。
4. 在指定详细信息页面上，在环境详细信息下：
    a. 在名称中为您的环境输入唯一的名称。
    b. 在 Airflow 版本中选择 Apache Airflow 版本。

> **Note**
>
> 如果未指定任何值，则默认为最新的 Apache Airflow 版本。可用的最新版本是 Apache Airflow v3.0.6。

5. 在 Amazon S3 的 DAG 代码下指定以下内容：

   a. S3 Bucket。选择浏览 S3 并选择 Amazon S3 存储桶，或者输入 Amazon S3 URI。

   b. DAG 文件夹。选择浏览 S3，然后选择 Amazon S3 存储桶中的 `dags` 文件夹，或者输入 Amazon S3 URI。

   c. 插件文件-可选。选择浏览 S3，然后选择 Amazon S3 存储桶上的 `plugins.zip` 文件，或者输入 Amazon S3 URI。

   d. 要求文件-可选。选择浏览 S3，然后选择 Amazon S3 存储桶上的 `requirements.txt` 文件，或者输入 Amazon S3 URI。

   e. 启动脚本文件-可选，选择浏览 S3并选择 Amazon S3 存储桶上的脚本文件，或者输入 Amazon S3 URI。

6. 选择下一步。

步骤 2：配置高级设置

配置高级设置

1. 在配置高级设置页面上，在联网下：

   - 选择 Amazon VPC。

     此步骤将填充 Amazon VPC 中的两个私有子网。

2. 在 Web 服务器访问下，选择您首选的 Apache Airflow 访问模式：

   a. 私有网络。这限制了对 Apache Airflow UI 的访问，只有在 Amazon VPC 中已获得 环境的 IAM 策略访问权限的用户才能访问。您需要获得权限才能为此步骤创建 Amazon VPC 端点。

> **Note**
>
> 如果 Apache Airflow UI 只能在公司网络中访问，并且不需要访问公共存储库即可进行 Web 服务器要求安装，请选择私有网络选项。如果您选择此访问模式选项，则需要创

建一种机制来访问 Amazon VPC 中的 Apache Airflow Web 服务器。有关更多信息，请参阅访问 Apache Airflow Web 服务器的 VPC 端点（私有网络访问）。

b. 公有网络。这使获得环境的 IAM 策略访问权限的用户可以通过互联网访问 Apache Airflow UI。

c. 公用和私有网络访问。适用于 Apache Airflow 版本 3.2.1 及更高版本。这允许通过互联网访问 Apache Airflow 用户界面，而工作人员则通过私有 VPC 端点与网络服务器通信。如果托管您的环境的 Amazon VPC 无法访问互联网，请选择此选项。

3. 在安全组下，选择用于保护 Amazon VPC 的安全组：

a. 默认情况下，Amazon MWAA 会在 Amazon VPC 中创建一个安全组，并在创建新安全组中使用特定的入站和出站规则。

b. 可选。取消选中创建新安全组中的复选框可选择最多 5 个安全组。

> **Note**
>
> 现有 Amazon VPC 安全组必须配置特定的入站和出站规则，才能允许网络流量。要了解更多信息，请参阅 Amazon MWAA 上的 VPC 安全。

4. 在环境类下，选择一个环境类。

我们建议选择支持您的工作负载所需的最小尺寸。您可以随时更改环境类。

5. 对于最大工作线程计数，请指定要在环境中运行的 Apache Airflow 工作线程的最大数量。

有关更多信息，请参阅高性能用例示例。

6. 指定最大 Web 服务器数和最小 Web 服务器数，以配置 Amazon MWAA 如何在环境中扩展 Apache Airflow Web 服务器。

有关 Web 服务器自动扩展的更多信息，请参阅 the section called “配置 Web 服务器自动扩缩”。

7. 在加密下，选择一个数据加密选项：

a. 默认情况下，Amazon MWAA 使用 AWS拥有的密钥来加密您的数据。

b. 可选。选择“自定义加密设置（高级）”以选择其他 AWS KMS 密钥。如果您选择在此步骤中指定Customer-managed 密钥，则必须指定 AWS KMS 密钥 ID 或 ARN。AWS KMS A@@mazon MWAA 不支持别名和多区域密钥。如果您在 Amazon S3 存储桶上指定了用于服务器端加密的 Amazon S3 密钥，则必须为 Amazon MWAA 环境指定相同的密钥。

> **Note**
>
> 您必须拥有该密钥的权限才能在 Amazon MWAA 控制台上选择该密钥。您还必须通过 附加密钥政策 中所述的附加策略授予 Amazon MWAA 使用密钥的权限。

8. 推荐。在 “监控” 下，为 Airflo w 日志配置选择一个或多个日志类别，将 Apache Airflow 日志发送到日志： CloudWatch

   a. Airflow 任务日志。选择要发送到 “登录日志” 级别的 Apache Airflow 任务日志 CloudWatch 的类型。

   b. Airflow Web 服务器日志。选择要发送到 “登录日志” 级别的 Apache Airflow 网络服务器日志的类型 CloudWatch 。

   c. Airflow 计划程序日志。选择要发送到 “登录日志” 级别的 Apache Airflow 调度程序日志的类型 CloudWatch 。

   d. Airflow 工作线程日志。选择要发送到 “登录日志” 级别的 Apache Airflow 工作日志 CloudWatch 的类型。

   e. Airflow DAG 处理日志。选择要发送到 “登录日志” 级别的 Apache Airflow DAG 处理日志 CloudWatch 的类型。

9. 可选。对于 Airflow 配置选项，选择添加自定义配置选项。

   您可以从 Apache Airflow 版本的 Apache Airflow 配置选项的建议下拉列表中进行选择，也可以指定自定义配置选项。例如 `core.default_task_retries`：3。

10. 可选。在标签下，选择添加新标记，将标签与环境相关联。例如，`Environment`：`Staging`。

11. 在权限下，选择一个执行角色。

    a. 默认情况下，Amazon MWAA 会在创建新角色中创建一个执行角色。您必须具有创建 IAM 角色的权限，才能使用此选项。

    b. 可选。选择输入角色 ARN 输入现有执行角色的 Amazon 资源名称（ARN）。

12. 选择下一步。

步骤 3：查看和创建

要查看环境摘要，请执行以下操作

- 查看环境摘要，选择创建环境。

> **Note**
>
> 创建环境大约需要二十到三十分钟。

接下来做什么？

- 要了解如何创建 Amazon S3 存储桶，请参阅 为 Amazon MWAA 创建 Amazon S3 存储桶。。

管理对 Amazon MWAA 环境的访问

需要允许适用于 Apache Airflow 的亚马逊托管工作流程使用 AWS 环境使用的其他服务和资源。您还需要获得访问亚马逊 MWAA 环境和您的 Apache Airflow 用户界面的权限 (IAM)。 AWS Identity and Access Management 本节介绍用于授予环境 AWS 资源访问权限的执行角色以及如何添加权限，以及访问您的 Amazon MWAA 环境和 Apache Airflow AWS 账户 用户界面所需的权限。

主题

访问 Amazon MWAA 环境

要使用 Amazon MWAA，您必须使用账户和具有必要权限的 IAM 实体。本主题介绍了您可以为 Amazon MWAA 环境的 Apache Airflow 开发团队和 Apache Airflow 用户附加的访问策略。

我们建议使用临时凭证并使用群组和角色配置联合身份来访问 Amazon MWAA 资源。最佳实践是，避免将策略直接附加到 IAM 用户。相反，可以定义组或角色以提供对 AWS 资源的临时访问权限。

IAM 角色是可在账户中创建的一种具有特定权限的 IAM 身份。IAM 角色与 IAM 用户类似，因为它是一个具有权限策略的 AWS 身份，该策略决定了该身份可以做什么和不能做什么 AWS。但是，角色旨在让需要它的任何人代入，而不是唯一地与某个人员关联。此外，角色没有关联的标准长期凭证（如密码或访问密钥）。相反，当你担任角色时，它会为你的角色会话提供临时安全证书。

要向联合身份分配权限，您可以创建角色并为角色定义权限。当联合身份进行身份验证时，该身份将与角色相关联并被授予由此角色定义的权限。有关用于联合身份验证的角色的信息，请参阅《IAM 用户指南》中的针对第三方身份提供者创建角色（联合身份验证）。如果您使用 IAM Identity Center，则需要配置权限集。为控制您的身份在进行身份验证后可以访问的内容，IAM Identity Center 将权限集与 IAM 中的角色相关联。有关权限集的信息，请参阅《AWS IAM Identity Center 用户指南》中的权限集。

您可以使用账户中的 IAM 角色授予其他访问您账户资源的 AWS 账户 权限。有关示例，请参阅 IAM 用户指南中的 IAM 教程： AWS 账户 使用 IAM 角色委派访问权限。

Sections

工作原理

并非所有 AWS Identity and Access Management (IAM) 实体都无法访问 Amazon MWAA 环境中使用的资源和服务。您必须创建一个策略，授予 Apache Airflow 用户访问这些资源的权限。例如，您需要向 Apache Airflow 开发团队授予访问权限。

Amazon MWAA 使用这些策略来验证用户是否具有在 AWS 控制台上或通过环境使用的 API 执行操作所需的权限。

您可以使用本主题中的 JSON 策略在 IAM 中为 Apache Airflow 用户创建一个策略，然后将该策略附加到 IAM 中的用户、群组或角色。

- AmazonMWAAFullConsoleAccess— 使用此策略授予在 Amazon MWAA 控制台上配置环境的权限。
- AmazonMWAAFullApiAccess— 使用此政策授予访问用于管理环境的所有 Amazon MWAA API 的权限。
- AmazonMWAAReadOnlyAccess— 使用此策略授予对 Amazon MWAA 控制台上环境使用的资源的访问权限。
- AmazonMWAAWebServerAccess— 使用此策略授予对 Apache Airflow 网络服务器的访问权限。
- AmazonMWAAAirflowCliAccess— 使用此策略授予运行 Apache Airflow CLI 命令的访问权限。

要提供访问权限，请为您的用户、组或角色添加权限：

- 中的用户和群组 AWS IAM Identity Center：

  创建权限集合。按照《AWS IAM Identity Center 用户指南》中创建权限集的说明进行操作。
- 通过身份提供者在 IAM 中托管的用户：

  创建适用于身份联合验证的角色。按照《IAM 用户指南》中针对第三方身份提供者创建角色（联合身份验证）的说明进行操作。
- IAM 用户：
  - 创建您的用户可以担任的角色。按照《IAM 用户指南》中为 IAM 用户创建角色的说明进行操作。
  - （不推荐使用）将策略直接附加到用户或将用户添加到用户组。按照《IAM 用户指南》中向用户添加权限（控制台）中的说明进行操作。

完整的控制台访问策略： AmazonMWAAFullConsoleAccess

如果用户需要在 Amazon MWAA 控制台上配置环境，则可能需要访问 `AmazonMWAAFullConsoleAccess` 权限策略。

> **Note**
>
> 完整控制台访问策略必须包含执行 `iam:PassRole` 的权限。这允许用户将与服务相关的角色和执行角色传递给 Amazon MWAA。Amazon MWAA 扮演每个角色代表您呼叫其他 AWS 服务。以下示例使用 `iam:PassedToService` 条件键将 Amazon MWAA 服务主体 (`airflow.amazonaws.com`) 指定为可将角色传递到的服务。
> 有关更多信息`iam:PassRole`，请参阅 IAM 用户指南中的授予用户向 AWS 服务传递角色的权限。

如果您想使用静态加密的 AWS 拥有的密钥 来创建和管理 Amazon MWAA 环境，请使用以下策略。

使用 AWS 拥有的密钥

JSON

```
{
    "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::111122223333:policy/service-role/MWAA-
Execution-Policy*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:AttachRolePolicy",
                "iam:CreateRole"
            ],
            "Resource": "arn:aws:iam::111122223333: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/*"
            ]
        }
    ]
}
```

如果您想使用静态加密的由客户托管的密钥来创建和管理 Amazon MWAA 环境，请使用以下策略。要使用客户托管密钥，IAM 委托人必须有权使用存储在您账户中的密钥访问 AWS KMS 资源。

使用由客户托管的密钥。

JSON

```
{
    "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::111122223333:policy/service-role/MWAA-
Execution-Policy*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:AttachRolePolicy",
                "iam:CreateRole"
            ],
            "Resource": "arn:aws:iam::111122223333: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:*:111122223333: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/*"
            ]
        }
    ]
}
```

完整的 API 和控制台访问政策： AmazonMWAAFullApiAccess

如果用户需要访问用于管理环境的所有 Amazon MWAA API，则可能需要访问 `AmazonMWAAFullApiAccess` 权限策略。它不授予访问 Apache Airflow UI 的权限。

> **Note**
>
> 完整 API 访问策略必须包含执行 `iam:PassRole` 的权限。这允许用户将与服务相关的角色和执行角色传递给 Amazon MWAA。Amazon MWAA 扮演每个角色代表您呼叫其他 AWS 服务。以下示例使用 `iam:PassedToService` 条件键将 Amazon MWAA 服务主体 (`airflow.amazonaws.com`) 指定为可将角色传递到的服务。
> 有关更多信息`iam:PassRole`，请参阅 IAM 用户指南中的授予用户向 AWS 服务传递角色的权限。

如果您想使用静态加密来创建和管理您的 Amazon MWAA 环境，请使用以下策略。 AWS 拥有的密钥

使用 AWS 拥有的密钥

JSON

```
{
   "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/*"
         ]
      }
   ]
}
```

如果您想使用静态加密的由客户托管的密钥来创建和管理 Amazon MWAA 环境，请使用以下策略。要使用客户托管密钥，IAM 委托人必须有权使用存储在您账户中的密钥访问 AWS KMS 资源。

使用由客户托管的密钥。

JSON

```
{
    "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:*:111122223333: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/*"
            ]
        }
    ]
```

```
}
```

Read-only 控制台访问策略： AmazonMWAAReadOnlyAccess

如果用户需要访问 Amazon MWAA 控制台环境详情页面上环境所用的资源，则可能需要访问 `AmazonMWAAReadOnlyAccess` 权限策略。它不允许用户创建新环境、编辑现有环境或允许用户访问 Apache Airflow UI。

JSON

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

Apache Airflow 用户界面访问策略： AmazonMWAAWebServerAccess

如果用户需要访问 Apache Airflow UI，则可能需要访问 `AmazonMWAAWebServerAccess` 权限策略。它不允许用户在 Amazon MWAA 控制台上访问环境或使用 Amazon MWAA API 执行任何操作。在 `{airflow-role}` 中指定 `Admin`、`Op`、`User`、`Viewer` 或 `Public` 角色以自定义 Web 令牌用户的访问级别。有关更多信息，请参阅《Apache Airflow 参考指南》中的默认角色。

JSON

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

```
            "Effect": "Allow",
            "Action": "airflow:CreateWebLoginToken",
            "Resource": [
             "arn:aws:airflow:us-east-1:111122223333:role/{your-environment-
name}/{airflow-role}"
            ]
        }
    ]
}
```

Note

- Amazon MWAA 将 IAM 集成到五个默认 Amazon Airflow 基于角色的访问控制 (RBAC) 角色。有关使用自定义 Apache Airflow 角色的更多信息，请参阅。the section called “教程：将用户限制为其中的一个子集 DAGs”
- 此策略中的 `Resource` 字段可用于为 Amazon MWAA 环境指定 Apache Airflow 基于角色的访问控制角色。但不支持在策略的 `Resource` 字段中使用 Amazon MWAA 环境 ARN（Amazon 资源名称）。

Apache Airflow Rest API 访问策略： AmazonMWAARestAPIAccess

要访问 Apache Airflow REST API，您必须在 IAM 策略中授予 `airflow:InvokeRestApi` 权限。在以下策略示例中，在 `{airflow-role}` 中指定 `Admin`、`Op`、`User`、`Viewer` 或 `Public` 角色以自定义用户访问权限级别。有关更多信息，请参阅《Apache Airflow 参考指南》中的默认角色。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowMwaaRestApiAccess",
            "Effect": "Allow",
            "Action": "airflow:InvokeRestApi",
            "Resource": [
```

```
            "arn:aws:airflow:us-east-1:111122223333:role/{your-environment-name}/
{airflow-role}"
            ]
        }
    ]
}
```

> **Note**
>
> - 配置私有 Web 服务器时，无法从虚拟私有云 (VPC) 之外调用 `InvokeRestApi` 操作。您可以使用 `aws:SourceVpc` 键对此操作执行更精细的访问控制。有关更多信息，请参阅 aws：SourceVpc
> - 此策略中的 `Resource` 字段可用于为 Amazon MWAA 环境指定 Apache Airflow 基于角色的访问控制角色。但不支持在策略的 `Resource` 字段中使用 Amazon MWAA 环境 ARN（Amazon 资源名称）。

Apache Airflow CLI 政策： AmazonMWAAAirflowCliAccess

如果用户需要运行 Apache Airflow CLI 命令（例如 `trigger_dag`），则可能需要访问 `AmazonMWAAAirflowCliAccess` 权限策略。它不允许用户在 Amazon MWAA 控制台上访问环境或使用 Amazon MWAA API 执行任何操作。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "airflow:CreateCliToken"
            ],
            "Resource": "arn:aws:airflow:us-east-1:111122223333:environment/
${EnvironmentName}"
        }
    ]
}
```

创建 JSON 策略

您可以创建 JSON 策略，并将策略附加到 IAM 控制台上的用户、角色或群组。以下步骤介绍如何在 IAM 中创建 JSON 策略。

要创建 JSON 策略，请执行以下操作

1. 在 IAM 控制台中打开策略页面。
2. 选择创建策略。
3. 选择 JSON 选项卡。
4. 添加 JSON 策略。
5. 选择查看策略。
6. 在名称和描述（可选）文本字段中各输入一个值。

   例如，您可以将策略命名为 `AmazonMWAAReadOnlyAccess`。
7. 选择创建策略。

将策略附加到开发者群组的示例用例

假设您在 IAM 中使用一个名为 `AirflowDevelopmentGroup` 的群组来向 Apache Airflow 开发团队中的所有开发人员授予权限。这些用户需要访问 `AmazonMWAAFullConsoleAccess`、`AmazonMWAAAirflowCliAccess` 和 `AmazonMWAAWebServerAccess` 权限策略。本节介绍如何在 IAM 中创建群组、创建和附加这些策略以及如何将该群组关联到 IAM 用户。这些步骤假设您使用的是 AWS自有密钥。

创建 AmazonMWAAFullConsoleAccess 策略

1. 下载AmazonMWAAFullConsoleAccess 访问策略。
2. 在 IAM 控制台中打开策略页面。
3. 选择创建策略。
4. 选择 JSON 选项卡。
5. 为 `AmazonMWAAFullConsoleAccess` 粘贴相应的 JSON 策略。
6. 替换以下值：

   a. *123456789012*— 您的 AWS 账户 身份证（例如`0123456789`）

b. *{your-kms-id}*— 客户托管密钥的唯一标识符，仅当您使用客户托管密钥进行静态加密时才适用。

7. 选择查看策略。
8. 在名称中键入 AmazonMWAAFullConsoleAccess。
9. 选择创建策略。

创建 AmazonMWAAWebServerAccess 策略

1. 下载AmazonMWAAWebServerAccess 访问策略。
2. 在 IAM 控制台中打开策略页面。
3. 选择创建策略。
4. 选择 JSON 选项卡。
5. 为 AmazonMWAAWebServerAccess 粘贴相应的 JSON 策略。
6. 替换以下值：

   a. *us-east-1*— 您的亚马逊 MWAA 环境所在的地区（例如）us-east-1

   b. *123456789012*— 你的 AWS 账户 身份证（例如0123456789）

   c. *{your-environment-name}*— 您的亚马逊 MWAA 环境名称（例如）MyAirflowEnvironment

   d. *{airflow-role}*— Admin Apache Airflow 默认角色

7. 选择查看策略。
8. 在名称中键入 AmazonMWAAWebServerAccess。
9. 选择创建策略。

创建 AmazonMWAAAirflowCliAccess 策略

1. 下载AmazonMWAAAirflowCliAccess 访问策略。
2. 在 IAM 控制台中打开策略页面。
3. 选择创建策略。
4. 选择 JSON 选项卡。
5. 为 AmazonMWAAAirflowCliAccess 粘贴相应的 JSON 策略。
6. 选择查看策略。

7. 在名称中键入 AmazonMWAAAirflowCliAccess。
8. 选择创建策略。

要创建群组，执行以下操作

1. 在 IAM 控制台中打开群组页面。
2. 输入 AirflowDevelopmentGroup 的名称。
3. 选择下一步。
4. 在筛选中键入 AmazonMWAA 来筛选结果。
5. 选择您创建的三个策略。
6. 选择下一步。
7. 选择创建组。

要与用户关联，请执行以下操作

1. 在 IAM 控制台中打开用户页面。
2. 选择一个用户。
3. 选择组。
4. 选择将用户添加到各群组。
5. 选择 AirflowDevelopmentGroup。
6. 然后选择添加到组。

接下来做什么？

- 要了解如何生成令牌以访问 Apache Airflow UI，请参阅 访问 Apache Airflow。
- 要详细了解有关创建 IAM 策略，请参阅创建 IAM 策略。

Service-linked 亚马逊 MWAA 的角色

适用于 Apache Airflow 的亚马逊托管工作流程 AWS Identity and Access Management 使用 (IAM) 服务相关角色。服务相关角色是一种独特的 IAM 角色，直接关联到 Amazon MWAA。 Service-linked 角色由 Amazon MWAA 预定义，包括该服务代表您调用其他 AWS 服务所需的所有权限。

服务相关角色可让您更轻松地设置 Amazon MWAA，因为您不必手动添加所需权限。Amazon MWAA 定义其服务相关角色的权限，除非另外定义，否则只有 Amazon MWAA 可以担任该角色。定义的权限包括信任策略和权限策略，以及不能附加到任何其他 IAM 实体的权限策略。

只有在首先删除相关资源后，您才能删除服务关联角色。这将保护 Amazon MWAA 资源，因为您不会无意中删除对资源的访问权限。

有关支持服务相关角色的其他服务的信息，请参阅与 IAM 配合使用的AWS 服务，并在Service-linked 角色列中搜索带有 “是” 的服务。选择是和链接，访问该服务的服务相关角色文档。

Service-linked 亚马逊 MWAA 的角色权限

Amazon MWAA 使用名为的服务相关角色 `AWSServiceRoleForAmazonMWAA` — 在您的账户中创建的服务相关角色授予亚马逊 MWAA 访问以下服务的权限： AWS

- 亚马逊 CloudWatch 日志（ 日CloudWatch 志）-为 Apache Airflow 日志创建日志组。
- Amazon CloudWatch (CloudWatch)-向您的账户发布与您的环境及其底层组件相关的指标。
- Amazon Elastic Compute Cloud（ Amazon EC2）— 创建以下资源：
  - 您的 VPC 中的亚马逊 VPC 终端节点，用于 AWS托管的 Amazon Aurora PostgreSQL 数据库集群，供 Apache Airflow 计划程序和工作人员使用。
  - 如果您为 Apache Airflow Web 服务器选择私有网络选项，则需要一个额外的 Amazon VPC 端点，用于允许对 Web 服务器进行网络访问。
  - 您@@ 的 Amazon VPC 中的弹性网络接口 (ENI)，允许通过网络访问您的亚马逊 VPC 中托管的 AWS 资源。

以下信任策略允许服务主体担任服务相关角色。Amazon MWAA 的服务主体是 `airflow.amazonaws.com`，如策略所示。

JSON

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

```
            "Action": "sts:AssumeRole"
        }
    ]
}
```

名为 AmazonMWAAServiceRolePolicy 的角色权限策略允许 Amazon MWAA 对指定资源完成以下操作：

JSON

```
{
    "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"
                    ]
                }
            }
        }
    ]
}
```

您必须配置权限，允许 IAM 实体（如用户、组或角色）创建、编辑或删除服务关联角色。有关更多信息，请参阅 I AM 用户指南中的Service-linked 角色权限。

为 Amazon MWAA 创建服务相关角色

您无需手动创建服务关联角色。当您使用 AWS 管理控制台、或 AWS API 创建新的 Amazon MWAA 环境时 AWS CLI，Amazon MWAA 会为您创建服务相关角色。

如果您删除该服务关联角色，然后需要再次创建，您可以使用相同流程在账户中重新创建此角色。当您创建另一个环境时，Amazon MWAA 会再次为您创建服务相关角色。

编辑 Amazon MWAA 的服务相关角色

Amazon MWAA 不允许编辑 AWSServiceRoleForAmazonMWAA 服务相关角色。创建服务关联角色后，您将无法更改角色的名称，因为可能有多种实体引用该角色。但是可以使用 IAM 编辑角色描述。有关更多信息，请参阅《IAM 用户指南》 中的编辑服务相关角色。

删除 Amazon MWAA 的服务相关角色

如果不再需要使用某个需要服务关联角色的功能或服务，我们建议您删除该角色。这样就没有未被主动监控或维护的未使用实体。

当您删除 Amazon MWAA 环境时，Amazon MWAA 会删除其作为服务一部分使用的所有关联资源。但是，您必须等到 Amazon MWAA 完成环境删除之前，然后才能尝试删除服务相关角色。如果您在 Amazon MWAA 删除环境之前删除服务相关角色，则 Amazon MWAA 可能无法删除该环境的所有关联资源。

使用 IAM 手动删除服务关联角色

使用 IAM 控制台 AWS CLI、或 AWS API 删除 AWSServiceRoleForAmazonMWAA服务相关角色。有关更多信息，请参阅《IAM 用户指南》中的删除服务相关角色。

Amazon ECS 服务相关角色支持的区域

Amazon ECS 支持在服务可用的所有区域中使用服务相关角色。有关更多信息，请参阅 Amazon MWAA 端点和配额。

策略更新

更改	描述	日期
Amazon MWAA 更新了其服务相关角色权限策略	AmazonMWAAServiceRolePolicy — Amazon MWAA 更新了其服务相关角色的权限策略，授予 Amazon MWAA 向客户账户发布与该服务底层资源相关的其他指标的权限。这些新指标发布在 AWS/MWAA 中	2022 年 11 月 18 日
Amazon MWAA 已开始跟踪更改	Amazon MWAA 开始跟踪其AWS托管服务相关角色权限策略的更改。	2022 年 11 月 18 日

Amazon MWAA 执行角色

执行角色是一个 AWS Identity and Access Management (IAM) 角色，其权限策略授予亚马逊托管工作流程 Apache Airflow 代表您调用 AWS 其他服务资源的权限。这可能包括诸如您的 Amazon S3 存储桶、AWS自有密钥和 CloudWatch 日志之类的资源。Amazon MWAA 环境需要每个环境配备一个执行

角色。本主题介绍如何使用和配置环境的执行角色，以允许 Amazon MWAA 访问您的环境使用的其他 AWS 资源。

目录

执行角色概述

Amazon MWAA 使用您的环境使用的其他 AWS 服务的权限来自执行角色。Amazon MWAA 执行角色需要获得环境使用的以下 AWS 服务的权限：

- 亚马逊 CloudWatch (CloudWatch) — 发送 Apache Airflow 指标和日志。
- Amazon Simple Storage Service（Amazon S3）— 用于解析环境的 DAG 代码和支持文件（例如，`requirements.txt`）。
- Amazon Simple Queue Service（Amazon SQS）— 将环境的 Amazon Airflow 任务在 Amazon MWAA 所拥有的 Amazon SQS 队列中排队。
- AWS Key Management Service (AWS KMS) — 用于环境的数据加密（使用AWS自有密钥或您的密Customer-managed 钥）。

> **Note**
>
> 如果您选择让 Amazon MWAA 使用 AWS 拥有的 KMS 密钥来加密您的数据，则必须在附加到您的 Amazon MWAA 执行角色的策略中定义权限，该策略允许通过 Amazon SQS 访问存储在您账户之外的任意 KMS 密钥。环境的执行角色需要满足以下两个条件才能访问任意 KMS 密钥：
>
> - 第三方账户中的 KMS 密钥需要通过其资源策略允许跨账户访问。
> - DAG 代码需要访问在第三方账户中以 `airflow-celery-` 开头的 Amazon SQS 队列，该队列使用相同的 KMS 密钥进行加密。
>
> 为了降低与跨账户访问资源相关的风险，我们建议您查看 DAG 中的代码，确保工作流程不会访问账户之外的任意 Amazon SQS 队列。此外，您可以使用存储在您自己账户中的由客户托管的 KMS 密钥来管理 Amazon MWAA 上的加密。这将环境的执行角色限制为只能访问您账户中的 KMS 密钥。
>
> 请记住，在选择加密选项后，您无法更改现有环境的选择。

执行角色还需要获得以下 IAM 操作的权限：

- `airflow:PublishMetrics` — 允许 Amazon MWAA 监控环境的运行状况。

默认情况下附加的权限

您可以使用 Amazon MWAA 控制台上的默认选项来创建执行角色和 AWS自有的密钥，然后使用本页上的步骤将权限策略添加到执行角色。

- 当您在控制台上选择创建新角色选项时，Amazon MWAA 会将环境所需的最低权限附加到执行角色。
- 在某些情况下，Amazon MWAA 会附加最高权限。例如，我们建议您在创建环境时在 Amazon MWAA 控制台上选择创建执行角色的选项。Amazon MWAA 通过在执行角色中使用正则表达式模式自动为所有 CloudWatch 日志组添加权限策略。"`arn:aws:logs:`*`us-east-1`*`:`*`111122223333`*`:log-group:`*`airflow-your-environment-name-*`*"

如何添加使用其它的权限 AWS 务

创建环境后，Amazon MWAA 无法向现有执行角色添加或编辑权限策略。您必须使用环境所需的其他权限策略来更新执行角色。例如，如果您的 DAG 需要访问权限 AWS Glue，则 Amazon MWAA 无法自动检测您的环境需要这些权限，也无法将这些权限添加到您的执行角色中。

您可以通过两种方式为执行角色添加权限：

- 通过内联修改执行角色的 JSON 策略。您可以使用本页上的 JSON 策略文档在 IAM 控制台上添加或替换执行角色的 JSON 策略。
- 通过为 AWS 服务创建 JSON 策略并将其附加到您的执行角色。您可以使用此页面上的步骤在 IAM 控制台上将 AWS 服务的新的 JSON 策略文档与您的执行角色关联起来。

假设执行角色已关联到环境，Amazon MWAA 可以立即开始使用添加的权限策略。这也意味着，如果您从执行角色中移除任何必需的权限，则 DAG 可能会失败。

如何关联新的执行角色

您可以随时更改环境的执行角色。如果新的执行角色尚未与环境关联，请使用本页上的步骤创建新的执行角色策略，并将该角色与环境相关联。

创建新角色

默认情况下，Amazon MWAA 会代表您创建用于数据加密的 AWS自有密钥和执行角色。创建环境时，您可以在 Amazon MWAA 控制台上选择默认选项。下图显示了为环境创建执行角色的默认选项。

> **Important**
>
> 创建新的执行角色时，请勿重复使用已删除的执行角色的名称。唯一的名称可以帮助防止冲突并确保正确的资源管理。

访问和更新执行角色策略

您可以在 Amazon MWAA 控制台上访问环境的执行角色，并在 IAM 控制台上更新该角色的 JSON 策略。

要更新执行角色策略，请执行以下操作

1. 在 Amazon MWAA 控制台上打开环境页面。
2. 选择环境。
3. 在权限窗格上选择执行角色以在 IAM 中打开权限页面。
4. 选择执行角色名称以打开权限策略。
5. 选择编辑策略。
6. 选择 JSON 选项卡。
7. 更新 JSON 策略。
8. 选择查看策略。
9. 选择保存更改。

附加 JSON 策略以使用其他 AWS 务

您可以为 AWS 服务创建 JSON 策略并将其附加到您的执行角色。例如，您可以附加以下 JSON 策略，以授予对 AWS Secrets Manager中所有资源的只读访问权限。

JSON

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

要将该策略附加到执行角色，请执行以下操作

1. 在 Amazon MWAA 控制台上打开环境页面。
2. 选择环境。
3. 在权限窗格上选择执行角色。
4. 选择附加策略。
5. 选择创建策略。
6. 选择 JSON。
7. 粘贴 JSON 策略。
8. 选择下一步：标签，然后选择下一步：审核。
9. 输入策略的描述性名称（例如 `SecretsManagerReadPolicy`）和策略的描述。
10. 选择创建策略。

使用账户级公有访问阻止授予对 Amazon S3 存储桶的访问权限

您可能需要使用 `PutPublicAccessBlock` Amazon S3 操作来阻止访问您账户中的所有存储桶。当您阻止访问您账户中的所有存储桶时，环境执行角色必须在权限策略中包含该 `s3:GetAccountPublicAccessBlock` 操作。

以下示例演示了在阻止访问您账户中的所有 Amazon S3 存储桶时必须附加到执行角色的策略。

JSON

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

关于限制对 Amazon S3 存储桶的访问权限的更多信息，请参阅 《Amazon Simple Storage Service 用户指南》中的阻止对 Amazon S3 存储的公有访问。

使用 Apache Airflow 连接

您还可以创建 Apache Airflow 连接，并在 Apache Airflow 连接对象中指定执行角色及其 ARN。要了解更多信息，请参阅 管理与 Apache Airflow 的连接。

执行角色的 JSON 策略示例

您可以使用本节中的两个示例权限策略替换用于现有执行角色的权限策略，或者创建新的执行角色并用于环境。这些策略包含 Apache Airflow 日志组的资源 ARN 占位符、Amazon S3 存储桶和 Amazon MWAA 环境。

我们建议复制示例策略，替换示例 ARN 或占位符，然后使用 JSON 策略创建或更新执行角色。

由客户托管的密钥的示例策略

以下示例显示了可用于Customer-managed 密钥的执行角色策略。

JSON

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Deny",
   "Action": "s3:ListAllMyBuckets",
   "Resource": [
     "arn:aws:s3:::amzn-s3-demo-bucket",
    "arn:aws:s3:::amzn-s3-demo-bucket/*"
   ]
  },
  {
   "Effect": "Allow",
   "Action": [
    "s3:GetObject*",
    "s3:GetBucket*",
    "s3:List*"
   ],
   "Resource": [
     "arn:aws:s3:::amzn-s3-demo-bucket",
    "arn:aws:s3:::amzn-s3-demo-bucket/*"
   ]
  },
```

```
  {
   "Effect": "Allow",
   "Action": [
    "logs:CreateLogStream",
    "logs:CreateLogGroup",
    "logs:PutLogEvents",
    "logs:GetLogEvents",
    "logs:GetLogRecord",
    "logs:GetLogGroupFields",
    "logs:GetQueryResults"
   ],
   "Resource": [
    "arn:aws:logs:us-east-1:111122223333: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:us-east-1:*:airflow-celery-*"
    },
    {
     "Effect": "Allow",
     "Action": [
      "kms:Decrypt",
      "kms:DescribeKey",
      "kms:GenerateDataKey*",
      "kms:Encrypt"
     ],
     "Resource": "arn:aws:kms:us-east-1:111122223333:key/your-kms-cmk-id",
     "Condition": {
      "StringLike": {
       "kms:ViaService": [
        "sqs.us-east-1.amazonaws.com",
        "s3.us-east-1.amazonaws.com"
       ]
      }
     }
    }
   ]
}
```

接下来，您需要允许 Amazon MWAA 担任此角色以代表您执行操作。为此，可以使用 IAM 控制台将 "airflow.amazonaws.com" 和 "airflow-env.amazonaws.com" 服务主体添加到该执行角色的可信实体列表中，或者使用 AWS CLI通过 IAM create-role 命令将这些服务主体放入该执行角色的代入角色策略文档中。请参阅以下代入角色策略文档示例：

JSON

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

```
            "Service": ["airflow.amazonaws.com","airflow-env.amazonaws.com"]
        },
        "Action": "sts:AssumeRole"
      }
   ]
}
```

然后将以下 JSON 策略附加到您的Customer-managed 密钥。此策略使用kms:EncryptionContext条件键前缀来允许访问日志中的 Apache Airflow 日志组。 CloudWatch

```
{
  "Sid": "Allow logs access",
  "Effect": "Allow",
  "Principal": {
    "Service": "logs.us-east-1.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-east-1:111122223333:*"
    }
  }
}
```

的示例政策 AWS-拥有的密钥

以下示例显示了您可用于AWS自有密钥的执行角色策略。

JSON

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

```
            "Effect": "Allow",
            "Action": "airflow:PublishMetrics",
            "Resource": "arn:aws:airflow:us-east-1:111122223333:environment/
{your-environment-name}"
        },
        {
            "Effect": "Deny",
            "Action": "s3:ListAllMyBuckets",
            "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket",
        "arn:aws:s3:::amzn-s3-demo-bucket/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject*",
                "s3:GetBucket*",
                "s3:List*"
            ],
            "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket",
        "arn:aws:s3:::amzn-s3-demo-bucket/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:CreateLogGroup",
                "logs:PutLogEvents",
                "logs:GetLogEvents",
                "logs:GetLogRecord",
                "logs:GetLogGroupFields",
                "logs:GetQueryResults"
            ],
            "Resource": [
            "arn:aws:logs:us-east-1:111122223333: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:us-east-1:*:airflow-celery-*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey",
                "kms:GenerateDataKey*",
                "kms:Encrypt"
            ],
            "NotResource": "arn:aws:kms:*:111122223333:key/*",
            "Condition": {
                "StringLike": {
                    "kms:ViaService": [
```

```
                    "sqs.us-east-1.amazonaws.com"
                    ]
                }
            }
        }
    ]
}
```

接下来做什么？

- 要了解您和 Apache Airflow 用户访问环境所需的权限，请参阅 访问 Amazon MWAA 环境。
- 了解 使用由客户托管的密钥进行加密。
- 浏览更多Customer-managed 政策示例。

Cross-service 混乱的副手预防

混淆代理问题是一个安全性问题，即不具有某操作执行权限的实体可能会迫使具有更高权限的实体执行该操作。在中 AWS，跨服务模仿可能会导致混乱的副手问题。 Cross-service 当一个服务（调用服务）调用另一个服务（被调用的服务）时，可能会发生模仿行为。可以操纵调用服务以使用其权限对另一个客户的资源进行操作，否则该服务没有访问权限。为了防止这种情况，我们 AWS 提供了一些工具，帮助您保护所有服务的数据，这些服务委托人已被授予访问您账户中资源的权限。

我们建议在环境的执行角色中使用 `aws:SourceArn` 和 `aws:SourceAccount` 全局条件上下文键，以限制 Amazon MWAA 为另一个服务访问资源提供的权限。如果您只希望将一个资源与跨服务访问相关联，请使用。`aws:SourceArn`如果您想允许该账户中的任何资源与跨服务使用操作相关联，请使用。`aws:SourceAccount`

防范混淆代理问题最有效的方法是使用 `aws:SourceArn` 全局条件上下文键和资源的完整 ARN。如果不知道资源的完整 ARN，或者正在指定多个资源，请针对 ARN 未知部分使用带有通配符字符（`*`）的 `aws:SourceArn` 全局上下文条件键。例如 `arn:aws:airflow:*:123456789012:environment/*`。

`aws:SourceArn` 的值必须是您正在为其创建执行角色的 Amazon MWAA 环境 ARN。

根据以下示例使用环境的执行角色信任策略中的 `aws:SourceArn` 和 `aws:SourceAccount` 全局条件上下文键来防范混淆代理问题。在创建新的执行角色时，您可以使用以下信任策略。

JSON

```
{
    "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:us-
east-1:123456789012:environment/your-environment-name"
                },
                "StringEquals": {
                    "aws:SourceAccount": "123456789012"
                }
            }
        }
    ]
}
```

Apache Airflow 访问模式

适用于 Apache Airflow 的亚马逊托管工作流程控制台包含内置选项，用于在您的环境中配置到 Apache Airflow 网络服务器的私有路由、公共路由，或者同时配置公共和私有路由。本指南介绍了在 Amazon MWAA 环境中可用的 Apache Airflow Web 服务器的访问模式，以及如果您选择私有网络选项，需要在 Amazon VPC 中配置的其他资源。

目录

Apache Airflow 访问模式

您可以为 Apache Airflow 网络服务器选择私有路由、公共路由，也可以同时选择公共路由和私有路由。要启用私有路由，请选择私有网络。这将用户访问 Apache Airflow Web 服务器的权限限制在 Amazon VPC 内。要启用公有路由，请选择公有网络。这可让用户通过互联网访问 Apache Airflow Web 服务器。要同时启用公用和私有路由，请选择公用和私有网络访问。这允许用户通过互联网访问 Apache Airflow 网络服务器，而工作人员则通过私有 VPC 端点与网络服务器通信。

公有网络

以下架构图描述了带有公有 Web 服务器的 Amazon MWAA 环境。

公有网络访问模式允许拥有环境的 IAM 策略访问权限的用户通过互联网访问 Apache Airflow UI。

> **Important**
>
> 如果您的环境使用带有公共网络访问模式的 Apache Airflow 3 或更高版本，则工作人员必须能够通过互联网访问 Web 服务器才能传达任务状态。如果托管工作人员的子网无法访问互联网（例如，没有 NAT 网关的私有子网），DAG 任务将失败。要解决这个问题，请升级到 Apache Airflow 版本 3.2.1 或更高版本，然后切换到公用和私有网络访问模式，该模式通过私有 VPC 端点路由工作人员的通信。

下图描述了在 Amazon MWAA 控制台上哪里可以找到公有网络选项。

私有网络

以下架构图描述了带有私有 Web 服务器的 Amazon MWAA 环境。

私有网络访问模式将访问 Apache Airflow UI 的权限限制为 Amazon VPC 中已获准访问环境 IAM 策略的用户。

创建具有私有 Web 服务器访问权限的环境时，必须将所有依赖项打包到 Python Wheel 档案 (`.whl`) 中，然后在 `requirements.txt` 中引用 `.whl`。有关使用 Wheel 打包和安装依赖项的说明，请参阅使用 Python Wheel 管理依赖项。

下图描述了在 Amazon MWAA 控制台上哪里可以找到私有网络选项。

公网和私网接入

适用于 Apache Airflow 版本 3.2.1 及更高版本。在 Apache Airflow 版本 3 及更高版本中，工作人员通过任务 API 将任务状态传达给网络服务器。如果您的 Amazon VPC 无法访问互联网，工作人员将无法访问公共 Web 服务器，从而导致 DAG 任务失败。此模式既创建了用于浏览器访问 Apache Airflow UI 的公共网络负载均衡器，又创建了用于工作人员到 Web 服务器通信的私有 VPC 端点，允许工作人员在不访问互联网的情况下访问 Web 服务器。有关每个组件，请参阅上面的公用网络和专用网络架构图。

> **Note**
>
> 在此模式下，浏览器通过公共网址访问 Apache Airflow 用户界面。私有 VPC 终端节点供工作人员用于内部通信，不用于浏览器访问。

访问模式概述

本节介绍当您选择公共网络、私有网络或公有和私有网络访问模式时，在您的 Amazon VPC 中创建的 VPC 终端节点 (AWS PrivateLink)。

公有网络访问模式

如果您为 Apache Airflow Web 服务器选择了公有网络访问模式，则网络流量将通过互联网公开路由。

- Amazon MWAA 为 Amazon Aurora PostgreSQL 元数据数据库创建 VPC 接口端点。端点是在映射到私有子网的可用区中创建的，并且独立于其他 AWS 账户。
- 然后，Amazon MWAA 会将私有子网中的 IP 地址绑定到接口端点。这旨在支持从 Amazon VPC 的每个可用区绑定一个 IP 的最佳实践。

私有网络访问模式

如果您为 Apache Airflow Web 服务器选择了私有网络访问模式，则网络流量将在 Amazon VPC 内私密路由。

- Amazon MWAA 为 Apache Airflow Web 服务器创建一个 VPC 接口端点，为 Amazon Aurora PostgreSQL 元数据数据库创建接口端点。端点是在映射到私有子网的可用区中创建的，并且独立于其他 AWS 账户。
- 然后，Amazon MWAA 会将私有子网中的 IP 地址绑定到接口端点。这旨在支持从 Amazon VPC 的每个可用区绑定一个 IP 的最佳实践。

公网和私网访问模式

如果您为 Apache Airflow 网络服务器选择了公共网络和私有网络访问模式，则流向 Apache Airflow UI 的网络流量将通过互联网公开路由，而工作人员与网络服务器的通信则在您的亚马逊 VPC 内以私密方式路由。

- 亚马逊 MWAA 为您的 Apache Airflow 网络服务器（用于工作人员连接）创建 VPC 接口终端节点，并为您的 Amazon Aurora PostgreSQL 元数据数据库创建接口终端节点。终端节点是在映射到您的私有子网的可用区中创建的，并且独立于其他 AWS 账户终端节点。
- 然后，Amazon MWAA 会将私有子网中的 IP 地址绑定到接口端点。这旨在支持从 Amazon VPC 的每个可用区绑定一个 IP 的最佳实践。
- Apache Airflow 用户界面可通过公共网络负载均衡器通过互联网访问。用户访问用户界面的方式与使用公共网络访问模式相同。

要了解更多信息，请参阅 the section called “Amazon VPC 和 Apache Airflow 访问模式的示例用例”。

访问模式的设置

下一节根据您为环境选择的 Apache Airflow 访问模式，介绍了您需要的其他设置和配置。

公有网络设置

如果您为 Apache Airflow Web 服务器选择公有网络选项，则可以在创建环境后开始使用 Apache Airflow UI。

您需要采取以下步骤来配置用户的访问权限以及您的环境使用其他 AWS 服务的权限。

1. 添加权限。亚马逊 MWAA 需要获得许可才能使用其他 AWS 服务。在您创建环境时，Amazon MWAA 会创建一个服务相关角色，允许其对亚马逊弹性容器注册表 (Amazon ECR)、日志和亚马逊 EC CloudWatch 2 使用某些 IAM 操作。

   您可以添加对这些服务使用其他操作的权限，也可以通过向执行角色添加权限来使用其他 AWS 服务。要了解更多信息，请参阅 Amazon MWAA 执行角色。
2. 创建用户策略。您可能需要为用户创建多个 IAM 策略，以配置对环境和 Apache Airflow UI 的访问权限。要了解更多信息，请参阅 访问 Amazon MWAA 环境。

私有网络的设置

如果您为 Apache Airflow Web 服务器选择 “专用网络” 选项，则需要配置用户的访问权限、您的环境使用 AWS 其他服务的权限，并创建从您的计算机访问 Amazon VPC 中资源的机制。

1. 添加权限。亚马逊 MWAA 需要获得许可才能使用其他 AWS 服务。在您创建环境时，Amazon MWAA 会创建一个服务相关角色，允许其对亚马逊弹性容器注册表 (Amazon ECR)、日志和亚马逊 EC CloudWatch 2 使用某些 IAM 操作。

   您可以添加对这些服务使用其他操作的权限，也可以通过向执行角色添加权限来使用其他 AWS 服务。要了解更多信息，请参阅 Amazon MWAA 执行角色。
2. 创建用户策略。您可能需要为用户创建多个 IAM 策略，以配置对环境和 Apache Airflow UI 的访问权限。要了解更多信息，请参阅 访问 Amazon MWAA 环境。
3. 启用网络访问。您需要在 Amazon VPC 中创建一个机制来连接到 Apache Airflow Web 服务器的 VPC 端点 (AWS PrivateLink)。例如，通过使用 AWS Client VPN从计算机创建 VPN 隧道。

为公用和私有网络访问进行设置

如果您为 Apache Airflow Web 服务器选择公用和私有网络访问选项，则可以在创建环境后开始使用 Apache Airflow 用户界面。浏览器访问不需要 VPN 或 VPC 终端节点访问机制。Apache Airflow 用户界面可通过互联网访问。工作人员通过私有 VPC 终端节点自动连接到 Web 服务器。

您需要采取以下步骤来配置用户的访问权限以及您的环境使用其他 AWS 服务的权限。

1. 添加权限。亚马逊 MWAA 需要获得许可才能使用其他 AWS 服务。在您创建环境时，Amazon MWAA 会创建一个服务相关角色，允许其对亚马逊弹性容器注册表 (Amazon ECR)、日志和亚马逊 EC CloudWatch 2 使用某些 IAM 操作。

   您可以添加对这些服务使用其他操作的权限，也可以通过向执行角色添加权限来使用其他 AWS 服务。要了解更多信息，请参阅 Amazon MWAA 执行角色。
2. 创建用户策略。您可能需要为用户创建多个 IAM 策略，以配置对环境和 Apache Airflow UI 的访问权限。要了解更多信息，请参阅 访问 Amazon MWAA 环境。

访问 Apache Airflow Web 服务器的 VPC 端点（私有网络访问）

如果您选择了私有网络选项，则需要在 Amazon VPC 中创建一个机制来访问 Apache Airflow Web 服务器的 VPC 端点 (AWS PrivateLink)。对于这些资源，我们建议使用与 Amazon MWAA 环境相同的 Amazon VPC、VPC 安全组和私有子网。

如果您选择了同时访问公用和私有网络，则无需创建访问 Apache Airflow 用户界面的机制。它可以通过互联网访问。工作人员会自动使用私有 VPC 终端节点进行内部通信。

要了解更多信息，请参阅管理 VPC 端点的访问。

访问 Apache Airflow

Amazon MWAA 提供了多种访问 Apache Airflow 环境的方法：Apache Airflow 用户界面 (UI) 控制台、Apache Airflow CLI 和 Apache Airflow REST API。你可以使用 Amazon MWAA 控制台在 Apache Airflow 用户界面中访问和调用 DAG，也可以使用亚马逊 APIs MWAA 获取令牌并调用 DAG。本节介绍访问 Apache Airflow UI 所需的权限、如何生成令牌以直接在命令 shell 中调用 Amazon MWAA API，以及 Apache Airflow CLI 中支持的命令。

主题

先决条件

下一节介绍使用本节中的命令和脚本所需的初步步骤。

访问

- AWS 账户 在 AWS Identity and Access Management (IAM) 中访问中的亚马逊 MWAA 权限策略。Apache Airflow 用户界面访问策略： AmazonMWAAWebServerAccess
- AWS 账户 在 AWS Identity and Access Management (IAM) 中访问亚马逊 MWAA 权限策略。完整的 API 和控制台访问政策： AmazonMWAAFullApiAccess

AWS CLI

AWS Command Line Interface (AWS CLI) 是一个开源工具，您可以使用命令行 shell 中的命令与 AWS 服务进行交互。要完成本节中的步骤，您需要以下满足以下条件：

- AWS CLI — 安装版本 2。

- AWS CLI — 使用快速配置`aws configure`。

打开 Apache Airflow UI

下图显示了 Amazon MWAA 控制台上指向 Apache Airflow UI 的链接。

登录 Apache Airflow

你需要你的 AWS 账户 内部 AWS Identity and Access Management (IAM) Apache Airflow 用户界面访问策略： AmazonMWAAWebServerAccess 权限才能访问你的 Apache Airflow 用户界面。

要访问 Apache Airflow UI，请执行以下操作

1. 在 Amazon MWAA 控制台上打开环境页面。
2. 选择环境。
3. 选择打开 Airflow UI。

创建 Apache Airflow Web 服务器访问令牌

您可以使用本页中的命令创建 Web 服务器访问令牌。访问令牌让您能够访问 Amazon MWAA 环境。例如，您可以获取令牌，然后使用 Amazon MWAA API 以编程方式部署 DAG。下一节包括使用 AWS CLI、bash 脚本、POST API 请求或 Python 脚本创建 Apache Airflow Web 登录令牌的步骤。响应中返回的令牌在 60 秒内有效。

> **Important**
>
> 自 2025 年 8 月 19 日起，Amazon MWAA 增加了对 IPv6 端点的支持，现在支持 IPv4 和 IPv6 端点。从该日期起，所有新创建的环境都将使用 `.on.aws` 域作为 Airflow 用户界面 (UI)。对于这些新创建的环境，客户必须将其 Airflow UI 从 `.on.aws` 域迁移到 `.amazonaws.com` 域。用于 Web 服务器和数据库的虚拟私有云 (VPC) 端点服务将保持其当前 `.amazonaws.com` 域，无需进行任何更改。

目录

先决条件

下一节介绍了使用本页上的命令和脚本所需的初步步骤。

访问

- 在 AWS Identity and Access Management (IAM) 中访问 Apache Airflow 用户界面访问策略：AmazonMWAAWebServerAccess 中的 Amazon MWAA 权限策略的 AWS 账户。
- 在 AWS Identity and Access Management (IAM) 中访问 Amazon MWAA 权限策略 完整的 API 和控制台访问政策： AmazonMWAAFullApiAccess 的 AWS 账户。

AWS CLI

AWS Command Line Interface (AWS CLI) 是一种开源工具，您可以用来在命令行 Shell 中使用命令与 AWS 服务进行交互。要完成本节中的步骤，您需要以下满足以下条件：

- AWS CLI – 安装版本 2。
- AWS CLI – 使用 `aws configure` 进行快速配置。

使用 AWS CLI

以下示例使用 AWS CLI 中的 create-web-login-token 命令创建 Apache Airflow Web 登录令牌。

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

使用 bash 脚本

以下示例使用 bash 脚本调用 AWS CLI 中的 create-web-login-token 命令来创建 Apache Airflow Web 登录令牌。

1. 复制以下代码示例的内容，并在本地另存为 get-web-token.sh。

```
#!/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. 用##占位符代替 YOUR_HOST_NAME 和 YOUR_ENVIRONMENT_NAME。例如，公有网络的主机名可能如下所示（没有 https:// ）：

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

3. （可选）macOS 和 Linux 用户可能需要运行以下命令以确保脚本可执行。

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

4. 运行以下脚本可获取 Web 登录令牌。

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

您的命令提示符将显示：

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

使用 Python 脚本

以下示例使用 Python 脚本中的 boto3 create_web_login_token 方法来创建 Apache Airflow Web 登录令牌。您可以在 Amazon MWAA 之外运行此脚本。您只需安装 boto3 库。您可能需要创建一个虚拟环境来安装该库。该环境假设您已经为账户配置了 AWS 身份验证凭证。

1. 复制以下代码示例的内容，并在本地另存为 create-web-login-token.py。

```
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. 用##占位符代替 YOUR_ENVIRONMENT_NAME。
3. 运行以下脚本可获取 Web 登录令牌。

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

接下来做什么？

- 在 CreateWebLoginToken 上浏览用于创建 Web 登录令牌的 Amazon MWAA API 操作。

为 Apache Airflow Web 服务器设置自定义域

使用 Amazon Managed Workflows for Apache Airflow (Amazon MWAA) 时，您可以为托管式 Apache Airflow Web 服务器设置自定义域。通过使用自定义域，您可以使用 Apache Airflow 用户界面、Apache Airflow CLI 或 Apache Airflow Web 服务器访问环境的 Amazon MWAA 托管式 Apache Airflow Web 服务器。

> **Note**
>
> 您只能在无互联网访问权限的私有 Web 服务器上使用自定义域。

Amazon MWAA 上的自定义域应用场景

1. 在 AWS 上的云应用程序中共享 Web 服务器域：通过使用自定义域，您可以定义用于访问 Web 服务器的用户友好型 URL，而不是生成的服务域名。您可以存储此自定义域，并将其作为应用程序中的环境变量共享。
2. 访问私有 Web 服务器：如果您想为无互联网访问权限的 VPC 中的 Web 服务器配置访问权限，使用自定义域可以简化 URL 重定向工作流。

主题

配置自定义域

要配置自定义域功能，您需要在创建或更新 Amazon MWAA 环境时通过 webserver.base_url Apache Airflow 配置来提供自定义域值。以下约束适用于自定义域名：

- 该值应是不含任何协议或路径的完全限定域名 (FQDN)。例如 your-custom-domain.com。
- Amazon MWAA 不允许在 URL 中添加路径。例如，your-custom-domain.com/dags/ 不是有效的自定义域名。
- URL 长度最多为 255 个 ASCII 字符。
- 如果您提供空字符串，则默认情况下将使用 Amazon MWAA 生成的 Web 服务器 URL 创建环境。

根据以下示例使用 AWS CLI 创建带有自定义 Web 服务器域名的环境。

```
aws mwaa create-environment \
--name my-mwaa-env \
--source-bucket-arn arn:aws:s3:::amzn-s3-demo-bucket \
--airflow-configuration-options '{"webserver.base_url":"my-custom-domain.com"}' \
--network-configuration '{"SubnetIds":["subnet-0123456789abcdef","subnet-
fedcba9876543210"]}' \
--execution-role-arn arn:aws:iam::123456789012:role/my-execution-role
```

创建或更新环境后，您需要在 AWS 账户 中设置联网基础设施，以便通过自定义域访问私有 Web 服务器。

要恢复使用默认由服务生成的 URL，请更新您的私有环境并移除 webserver.base_url 配置选项。

设置联网基础设施

执行以下步骤设置所需的联网基础设施，以便与 AWS 账户 中的自定义域一起使用。

1. 获取 Amazon VPC 端点网络接口（ENI）的 IP 地址。要执行此操作，请首先使用 get-environment 查找适合环境的 WebserverVpcEndpointService。

   ```
   aws mwaa get-environment --name your-environment-name
   ```

如果成功，您将收到与以下内容类似的输出。

```
{
  "Environment": {
    "AirflowConfigurationOptions": {},
    "AirflowVersion": "latest-version",
    "Arn": "environment-arn",
    "CreatedAt": "2024-06-01T01:00:00-00:00",
    "DagS3Path": "dags",
    .
    .
    .
    "WebserverVpcEndpointService": "web-server-vpc-endpoint-service",
    "WeeklyMaintenanceWindowStart": "TUE:21:30"
  }
}
```

记下 WebserverVpcEndpointService 值并将其用于以下 Amazon EC2 describe-vpc-endpoints 命令中的 web-server-vpc-endpoint-service。以下命令中的 --filters Name=service-name,Values=*web-server-vpc-endpoint-service-id*。

2. 检索 Amazon VPC 端点详细信息。此命令用于获取与特定服务名称匹配的 Amazon VPC 端点详细信息，并以文本格式返回端点 ID 和关联的网络接口 ID。

```
aws ec2 describe-vpc-endpoints \
 --filters Name=service-name,Values=web-server-vpc-endpoint-service \
 --query 'VpcEndpoints[*].
{EndpointId:VpcEndpointId,NetworkInterfaceIds:NetworkInterfaceIds}' \
 --output text
```

3. 获取网络接口详细信息。此命令用于检索与上一步中所确定 Amazon VPC 端点关联的每个网络接口的私有 IP 地址。

```
for eni_id in $(
  aws ec2 describe-vpc-endpoints \
   --filters Name=service-name,Values=service-id \
   --query 'VpcEndpoints[*].NetworkInterfaceIds' \
   --output text
 ); do
 aws ec2 describe-network-interfaces \
  --network-interface-ids $eni_id \
```

```
  --query 'NetworkInterfaces[*].PrivateIpAddresses[*].PrivateIpAddress' \
  --output text
      done
```

4. 使用 `create-target-group` 创建新的目标组。您将使用此目标组来注册 Web 服务器 Amazon VPC 端点的 IP 地址。

```
aws elbv2 create-target-group \
--name new-target-group-namne \
--protocol HTTPS \
--port 443 \
--vpc-id web-server-vpc-id \
--target-type ip \
--health-check-protocol HTTPS \
--health-check-port 443 \
--health-check-path / \
--health-check-enabled \
--matcher 'HttpCode="200,302"'
```

使用 `register-targets` 命令注册 IP 地址。

```
aws elbv2 register-targets \
--target-group-arn target-group-arn \
--targets Id=ip-address-1 Id=ip-address-2
```

5. 请求 ACM 证书。如果您使用的是现有证书，则跳过此步骤。

```
aws acm request-certificate \
--domain-name my-custom-domain.com \
--validation-method DNS
```

6. 配置 Application Load Balancer。首先创建负载均衡器，然后为该负载均衡器创建侦听器。指定在上一步中创建的 ACM 证书。

```
aws elbv2 create-load-balancer \
--name my-mwaa-lb \
--type application \
--subnets subnet-id-1 subnet-id-2
```

```
aws elbv2 create-listener \
--load-balancer-arn load-balancer-arn \
```

```
--protocol HTTPS \
--port 443 \
--ssl-policy ELBSecurityPolicy-2016-08 \
--certificates CertificateArn=acm-certificate-arn \
--default-actions Type=forward,TargetGroupArn=target-group-arn
```

如果您在私有子网中使用网络负载均衡器，请设置堡垒主机或 Site-to-Site VPN 隧道来访问 Web 服务器。

7. 使用 Route 53 为该域创建托管区。

```
aws route53 create-hosted-zone --name my-custom-domain.com \
--caller-reference 1
```

为该域创建一条 A 记录。要使用 AWS CLI 执行此操作，请使用 `list-hosted-zones-by-name` 获取托管区 ID，然后使用 `change-resource-record-sets` 应用记录。

```
HOSTED_ZONE_ID=$(aws route53 list-hosted-zones-by-name \
--dns-name my-custom-domain.com \
--query 'HostedZones[0].Id' --output text)
```

```
aws route53 change-resource-record-sets \
--hosted-zone-id $HOSTED_ZONE_ID \
--change-batch '{
  "Changes": [
    {
      "Action": "CREATE",
      "ResourceRecordSet": {
        "Name": "my-custom-domain.com",
        "Type": "A",
        "AliasTarget": {
          "HostedZoneId": "load-balancer-hosted-zone-id>",
          "DNSName": "load-balancer-dns-name",
          "EvaluateTargetHealth": true
        }
      }
    }
  ]
}'
```

8. 更新 Web 服务器 Amazon VPC 端点的安全组规则，使其遵循最低权限原则，仅允许来自应用程序负载均衡器所在公有子网的 HTTPS 流量。将以下 JSON 保存到本地。例如，`sg-ingress-ip-permissions.json`。

```
[
  {
    "IpProtocol": "tcp",
    "FromPort": 443,
    "ToPort": 443,
    "UserIdGroupPairs": [
      {
        "GroupId": "load-balancer-security-group-id"
      }
    ],
    "IpRanges": [
      {
        "CidrIp": "public-subnet-1-cidr"
      },
      {
        "CidrIp": "public-subnet-2-cidr"
      }
    ]
  }
]
```

运行以下 Amazon EC2 命令更新入口安全组规则。指定 `--ip-permissions` 的 JSON 文件。

```
aws ec2 authorize-security-group-ingress \
--group-id <security-group-id> \
--ip-permissions file://sg-ingress-ip-permissions.json
```

运行以下 Amazon EC2 命令更新出口规则。

```
aws ec2 authorize-security-group-egress \
--group-id webserver-vpc-endpoint-security-group-id \
--protocol tcp \
--port 443 \
--source-group load-balancer-security-group-id
```

打开 Amazon MWAA 控制台并导航到 Apache Airflow UI。如果是在私有子网中设置网络负载均衡器，而不是此处使用的应用程序负载均衡器，则必须使用以下选项之一访问 Web 服务器。

- the section called “教程：Linux 堡垒主机”
- the section called “教程： AWS Client VPN”

创建 Apache Airflow CLI 令牌

> **Tip**
>
> REST API 比 CLI 更现代，专为与外部系统的编程集成而设计。REST 是与 Apache Airflow 交互的首选方式。

您可以使用本页上的命令生成 CLI 令牌，然后直接在命令 shell 中调用 Amazon MWAA API。例如，您可以获取令牌，然后使用 Amazon MWAA API 以编程方式部署 DAG。下一节包括使用 AWS CLI、curl 脚本、Python 脚本或 bash 脚本创建 Apache Airflow CLI 令牌的步骤。响应中返回的令牌在 60 秒内有效。

AWS CLI 令牌旨在替代同步 shell 操作，而不是异步 API 命令。因此，可用的并发是有限的。为确保 Web 服务器对用户保持响应能力，建议在前一个请求成功完成之前不要打开新的 AWS CLI 请求。

目录

先决条件

下一节介绍了使用本页上的命令和脚本所需的初步步骤。

访问

- 在 AWS Identity and Access Management (IAM) 中访问 Apache Airflow 用户界面访问策略：AmazonMWAAWebServerAccess 中的 Amazon MWAA 权限策略的 AWS 账户。
- 在 AWS Identity and Access Management (IAM) 中访问 Amazon MWAA 权限策略 完整的 API 和控制台访问政策： AmazonMWAAFullApiAccess 的 AWS 账户。

AWS CLI

AWS Command Line Interface (AWS CLI) 是一种开源工具，您可以用来在命令行 Shell 中使用命令与 AWS 服务进行交互。要完成本节中的步骤，您需要以下满足以下条件：

- AWS CLI – 安装版本 2。
- AWS CLI – 使用 `aws configure` 进行快速配置。

使用 AWS CLI

以下示例使用 AWS CLI 中的 create-cli-token 命令创建 Apache Airflow CLI 令牌。

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

使用 curl 脚本

以下示例使用 curl 脚本调用 AWS CLI 中的 create-web-login-token 命令，通过 Apache Airflow Web 服务器上的端点调用 Apache Airflow CLI。

Apache Airflow v3

1. 从文本文件中复制 curl 语句并将其粘贴到命令 shell 中。

> **Note**
>
> 将其复制到剪贴板后，可能需要使用 shell 菜单中的编辑 > 粘贴。

```
CLI_JSON=$(aws mwaa --region us-east-1 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 -L --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 --logical-date $(date -u +"%Y-%m-%dT%H:
%M:%SZ")") \
&& echo "Output:" \
&& echo $CLI_RESULTS | jq -r '.stdout' | base64 --decode \
&& echo "Errors:" \
&& echo $CLI_RESULTS | jq -r '.stderr' | base64 --decode
```

2. 用环境的 AWS 区域、*YOUR_DAG_NAME* 和 *YOUR_ENVIRONMENT_NAME* 替换 *red* 中的占位符。例如，公有网络的主机名可能如下所示（没有 https:// ）：

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

您的命令提示符将显示：

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

Apache Airflow v2

1. 从文本文件中复制 curl 语句并将其粘贴到命令 shell 中。

> **Note**
>
> 将其复制到剪贴板后，可能需要使用 shell 菜单中的编辑 > 粘贴。

```
CLI_JSON=$(aws mwaa --region us-east-1 create-cli-token --
name YOUR_ENVIRONMENT_NAME) \
&& CLI_TOKEN=$(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. 用环境的 AWS 区域、YOUR_DAG_NAME 和 YOUR_ENVIRONMENT_NAME 替换 *red* 中的占位符。例如，公有网络的主机名可能如下所示（没有 https:// ）：

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

您的命令提示符将显示：

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

使用 bash 脚本

以下示例使用 bash 脚本来调用 AWS CLI 中的 create-cli-token 命令创建 Apache Airflow CLI 令牌。

Apache Airflow v3

1. 复制以下代码示例的内容，并在本地另存为 `get-cli-token.sh`。

```
# brew install jq
        aws mwaa create-cli-token --name YOUR_ENVIRONMENT_NAME | export
 CLI_TOKEN=$(jq -r .CliToken) && curl -L --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 --logical-date $(date -u +"%Y-%m-
%dT%H:%M:%SZ")"
```

2. 用##占位符替换 YOUR_ENVIRONMENT_NAME、YOUR_HOST_NAME 和 YOUR_DAG_NAME。例如，公有网络的主机名可能如下所示（没有 https:// ）：

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

3. （可选）macOS 和 Linux 用户可能需要运行以下命令以确保脚本可执行。

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

4. 运行以下脚本可创建 Apache Airflow CLI 令牌。

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

Apache Airflow v2

1. 复制以下代码示例的内容，并在本地另存为 get-cli-token.sh。

```
# 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. 用##占位符替换 YOUR_ENVIRONMENT_NAME、YOUR_HOST_NAME 和 YOUR_DAG_NAME。例如，公有网络的主机名可能如下所示（没有 https://）：

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

3. （可选）macOS 和 Linux 用户可以运行以下命令以确保脚本可执行。

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

4. 运行以下脚本可创建 Apache Airflow CLI 令牌。

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

使用 Python 脚本

以下示例使用 Python 脚本中的 boto3 create_cli_token 方法来创建 Apache Airflow CLI 令牌，并触发 DAG。您可以在 Amazon MWAA 之外运行此脚本。您只需安装 boto3 库。您可能需要创建一个虚拟环境来安装该库。该环境假设您已经为账户配置了 AWS 身份验证凭证。

Apache Airflow v3

1. 复制以下代码示例的内容，并在本地另存为 `create-cli-token.py`。

```
"""
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. 用占位符替换 YOUR_ENVIRONMENT_NAME 和 YOUR_DAG_NAME。
3. 运行以下脚本可创建 Apache Airflow CLI 令牌。

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

Apache Airflow v2

1. 复制以下代码示例的内容，并在本地另存为 create-cli-token.py。

```
"""
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. 用占位符替换 YOUR_ENVIRONMENT_NAME 和 YOUR_DAG_NAME。

3. 运行以下脚本可创建 Apache Airflow CLI 令牌。

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

接下来做什么？

- 在 createClitoken 上浏览用于创建 CLI 令牌的 Amazon MWAA API 操作。

使用 Apache Airflow REST API

对于运行 Apache Airflow v2.4.3 及更高版本的环境，Amazon Managed Workflows for Apache Airflow (Amazon MWAA) 支持使用 Apache Airflow REST API 直接与您的 Apache Airflow 环境进行交互。这使您可以通过编程方式访问和管理您的 Amazon MWAA 环境，从而为调用数据编排工作流程、管理和监控各种 Apache Airflow 组件（例如元数据数据库 DAGs、触发器和调度程序）的状态提供了一种标准化的方式。

为了在使用 Apache Airflow REST API 时支持可扩展性，Amazon MWAA 提供了水平扩缩 Web 服务器容量的选项，以满足增加的需求，无论是来自 REST API 请求、命令行界面 (CLI) 的使用还是并发 Apache Airflow 用户界面 (UI) 用户数量的增加。有关 Amazon MWAA 如何扩展 Web 服务器的更多信息，请参阅 the section called “配置 Web 服务器自动扩缩”。

您可以使用 Apache Airflow REST API 实现环境的以下使用案例：

- 编程访问 – 您现在可以在不依赖 Apache Airflow 用户界面或 CLI 的情况下，启动 Apache Airflow DAG 运行、管理数据集以及检索元数据数据库、触发器和调度器等各种组件的状态。
- 与外部应用程序和微服务集成 – 由于支持 REST API，您可以构建自定义解决方案以将您的 Amazon MWAA 环境与其他系统集成。例如，您可以启动工作流以响应来自外部系统的事件，例如已完成的数据库作业或新用户注册等。
- 集中监控 — 您可以构建监控控制面板，汇总多 DAGs 个 Amazon MWAA 环境中的状态，从而实现集中监控和管理。

有关 Apache Airflow REST API 的更多信息，请参阅 Apache Airflow REST API 参考。

通过使用`InvokeRestApi`，您可以使用凭据访问 Apache Airflow REST API。 AWS 您也可以通过获取 Web 服务器访问令牌，然后使用该令牌进行调用的方式来访问。

如果您在使用 InvokeRestApi 操作时遇到消息 Update your environment to use InvokeRestApi 错误，则表示您需要更新 Amazon MWAA 环境。当您的 Amazon MWAA 环境与 InvokeRestApi 功能相关的最新更改不兼容时，就会发生此错误。要解决此问题，请更新您的 Amazon MWAA 环境以纳入 InvokeRestApi 功能的必要更改。

InvokeRestApi 操作的默认超时时间为 10 秒。如果操作未在这 10 秒的时间范围内完成，则会自动终止并引发错误。确保您的 REST API 调用设计为在此超时时间内完成，以避免出现错误。

为了在使用 Apache Airflow REST API 时支持可扩展性，Amazon MWAA 提供了水平扩缩 Web 服务器容量的选项，以满足增加的需求，无论是来自 REST API 请求、命令行界面 (CLI) 的使用还是并发 Apache Airflow 用户界面 (UI) 用户数量的增加。有关 Amazon MWAA 如何扩展 Web 服务器的更多信息，请参阅 the section called “配置 Web 服务器自动扩缩”。

您可以使用 Apache Airflow REST API 实现环境的以下使用案例：

- 编程访问 – 您现在可以在不依赖 Apache Airflow 用户界面或 CLI 的情况下，启动 Apache Airflow DAG 运行、管理数据集以及检索元数据数据库、触发器和调度器等各种组件的状态。
- 与外部应用程序和微服务集成 – 由于支持 REST API，您可以构建自定义解决方案以将您的 Amazon MWAA 环境与其他系统集成。例如，您可以启动工作流以响应来自外部系统的事件，例如已完成的数据库作业或新用户注册等。
- 集中监控 — 您可以构建监控控制面板，汇总多 DAGs 个 Amazon MWAA 环境中的状态，从而实现集中监控和管理。

有关 Apache Airflow REST API 的更多信息，请参阅 Apache Airflow REST API 参考。

通过使用InvokeRestApi，您可以使用凭据访问 Apache Airflow REST API。 AWS 您也可以通过获取 Web 服务器访问令牌，然后使用该令牌进行调用的方式来访问。

- 如果您在使用 InvokeRestApi 操作时遇到消息 Update your environment to use InvokeRestApi 错误，则表示您需要更新 Amazon MWAA 环境。当您的 Amazon MWAA 环境与 InvokeRestApi 功能相关的最新更改不兼容时，就会发生此错误。要解决此问题，请更新您的 Amazon MWAA 环境以纳入 InvokeRestApi 功能的必要更改。
- InvokeRestApi 操作的默认超时时间为 10 秒。如果操作未在这 10 秒的时间范围内完成，则会自动终止并引发错误。确保您的 REST API 调用设计为在此超时时间内完成，以避免出现错误。

> **Important**
>
> 响应有效载荷大小不能超过 6 MB。如果超出限制，RestApi 将失败。

根据以下示例对 Apache Airflow REST API 进行 API 调用并启动新的 DAG 运行：

主题

- 授予对 Apache Airflow REST API 的访问权限：airflow:InvokeRestApi
- 调用 Apache Airflow REST API
- 创建 Web 服务器会话令牌并调用 Apache Airflow REST API

授予对 Apache Airflow REST API 的访问权限：**airflow:InvokeRestApi**

要 AWS 使用证书访问 Apache Airflow REST API，您必须在 IAM 策略airflow:InvokeRestApi中授予权限。在以下策略示例中，在 {airflow-role} 中指定 Admin、Op、User、Viewer 或 Public 角色以自定义用户访问权限级别。有关更多信息，请参阅《Apache Airflow 参考指南》中的默认角色。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowMwaaRestApiAccess",
            "Effect": "Allow",
            "Action": "airflow:InvokeRestApi",
            "Resource": [
            "arn:aws:airflow:us-east-1:111122223333:role/{your-environment-name}/
{airflow-role}"
            ]
        }
    ]
}
```

> **Note**
>
> 配置私有 Web 服务器时，无法从虚拟私有云 (VPC) 之外调用 `InvokeRestApi` 操作。您可以使用 `aws:SourceVpc` 键对此操作执行更精细的访问控制。有关更多信息，请参阅 a ws:SourceVpc。

调用 Apache Airflow REST API

以下示例脚本介绍了如何使用 Apache Airflow REST API 列出环境中的 DAGs 可用变量以及如何创建 Apache Airflow 变量：

```
import boto3

  env_name = "MyAirflowEnvironment"

  def list_dags(client):
    request_params = {
      "Name": env_name,
      "Path": "/dags",
      "Method": "GET",
      "QueryParameters": {
        "paused": False
      }
    }
  response = client.invoke_rest_api(
    **request_params
  )

  print("Airflow REST API response: ", response['RestApiResponse'])

  def create_variable(client):
    request_params = {
      "Name": env_name,
      "Path": "/variables",
      "Method": "POST",
      "Body": {
        "key": "test-restapi-key",
        "value": "test-restapi-value",
        "description": "Test variable created by MWAA InvokeRestApi API",
      }
    }
```

```
response = client.invoke_rest_api(
  **request_params
)

print("Airflow REST API response: ", response['RestApiResponse'])

if __name__ == "__main__":
  client = boto3.client("mwaa")
  list_dags(client)
  create_variable(client)
```

创建 Web 服务器会话令牌并调用 Apache Airflow REST API

要创建 Web 服务器访问令牌，请使用以下 Python 函数。该函数会首先调用 Amazon MWAA API 来获取 Web 登录令牌。Web 登录令牌将在 60 秒后过期，然后会交换为一个 Web 会话令牌，后者可让您访问 Web 服务器和使用 Apache Airflow REST API。如果您需要每秒 10 个事务（TPS）以上的节流容量，则可以使用此方法访问 Apache Airflow REST API。

会话令牌将在 12 小时后过期。

> **Tip**
>
> 以下代码示例中从 Apache Airflow v2 到 v3 的主要变化是：
>
> - REST API 路径从 `/api/v1` 更改为 `/api/v2`
> - 登录路径从 `/aws_maa/login` 更改为 `/pluginsv2/aws_mwaa/login`
> - 登录 `response.cookies["_token"]` 的响应包含令牌信息，您必须在后续的 API 调用中使用这些信息
> - 对于 REST API 调用，您必须在标头中传递以下 `jwt_token` 信息：
>
> ```
> headers = {
>   "Authorization": f"Bearer {jwt_token}",
>   "Content-Type": "application/json"
> }
> ```

Apache Airflow v3

```
def get_token_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}/pluginsv2/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 successful
  if response.status_code == 200:

  # Return the hostname and the session cookie
  return (
    web_server_host_name,
    response.cookies['_token']
  )
  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 v2

```
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
```

完成身份验证后，您将会获得开始向 API 端点发送请求的凭证。在下一部分的示例中，使用端点 dags/{dag_name}/dagRuns。

Apache Airflow v3

```
def trigger_dag(region, env_name, dag_id):
      """
      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_id (str): ID of the DAG to trigger.
      """

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

      # Retrieve the web server hostname and token for authentication
      try:
      web_server_host_name, jwt_token = get_token_info(region, env_name)
      if not jwt_token:
      logging.error("Authentication failed, no jwt token retrieved.")
      return
      except Exception as e:
      logging.error(f"Error retrieving token info: {str(e)}")
      return

      # Prepare headers and payload for the request
      request_headers = {
      "Authorization": f"Bearer {jwt_token}",
      "Content-Type": "application/json" # Good practice to include, even for GET
      }

      # sample request body input
      json_body = {"logical_date": "2025-09-17T14:15:00Z"}

      # Construct the URL for triggering the DAG
      url = f"https://{web_server_host_name}/api/v2/dags/{dag_id}/dagRuns"
```

```
    # Send the POST request to trigger the DAG
    try:
    response = requests.post(url, headers=request_headers, 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_id}")
    sys.exit(1)

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

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

Apache Airflow v2

```
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 命令参考

本主题介绍了 Amazon MWAA 上支持和不支持的 Apache Airflow CLI 命令。

> **Tip**
>
> REST API 比 CLI 更现代，专为与外部系统的编程集成而设计。REST 是与 Apache Airflow 交互的首选方式。

目录

先决条件

下一节介绍了使用本页上的命令和脚本所需的初步步骤。

访问

- AWS 账户 在 AWS Identity and Access Management (IAM) 中访问中的亚马逊 MWAA 权限策略。Apache Airflow 用户界面访问策略： AmazonMWAAWebServerAccess

- AWS 账户 在 AWS Identity and Access Management (IAM) 中访问亚马逊 MWAA 权限策略。完整的 API 和控制台访问政策： AmazonMWAAFullApiAccess

AWS CLI

AWS Command Line Interface (AWS CLI) 是一个开源工具，您可以使用命令行 shell 中的命令与 AWS 服务进行交互。要完成本节中的步骤，您需要以下满足以下条件：

- AWS CLI — 安装版本 2。
- AWS CLI — 使用快速配置`aws configure`。

更改了哪些内容？

- v3：Airflow 架构。Apache Airflow v3 引入了突破性的架构更改，以提高安全性和可扩展性，并简化维护。要了解更多信息，请参阅升级到 Airflow 3。
- v2：Airflow CLI 命令结构。Apache Airflow v2 CLI 的组织方式是将相关命令分组为子命令，这意味着如果您想升级到 Apache Airflow v2，则需要更新 Apache Airflow v1 脚本。例如，Apache Airflow v1 中的 `unpause` 已更新为 Apache Airflow v2 中的 `dags unpause`。要了解更多信息，请参阅 2.0 中的 Airflow CLI 更改。

支持的 CLI 命令

下一节列出了 Amazon MWAA 上可用的 Apache Airflow CLI 命令。

支持的 命令

Apache Airflow v3

次要版本	命令
v3.0.6、v3.2.1	资产详情
v3.0.6、v3.2.1	资产列表
v3.0.6、v3.2.1	资产实体化

次要版本	命令
v3.0.6、v3.2.1	回填创建
v3.0.6、v3.2.1	备忘单
v3.0.6、v3.2.1	添加连接
v3.0.6、v3.2.1	删除连接
v3.0.6、v3.2.1	删除 dags
v3.0.6、v3.2.1	dags 列表
v3.0.6、v3.2.1	dags 列表-任务
v3.0.6、v3.2.1	dags 列表导入错误
v3.0.6、v3.2.1	dags 列表-运行次数
v3.0.6、v3.2.1	dags 下次-执行
v3.0.6、v3.2.1	暂停 dag
v3.0.6、v3.2.1	报告 dags
v3.0.6、v3.2.1	重新序列化 dags
v3.0.6、v3.2.1	显示 dags

次要版本	命令
v3.0.6、v3.2.1	陈述 dags
v3.0.6、v3.2.1	测试 dags
v3.0.6、v3.2.1	触发 dags
v3.0.6、v3.2.1	取消暂停 dags
v3.0.6、v3.2.1	清理 db
v3.0.6、v3.2.1	提供商的行为
v3.0.6、v3.2.1	获得提供商
v3.0.6、v3.2.1	提供商挂钩
v3.0.6、v3.2.1	提供商链接
v3.0.6、v3.2.1	提供商列表
v3.0.6、v3.2.1	提供程序通知
v3.0.6、v3.2.1	提供商密钥
v3.0.6、v3.2.1	提供商触发器
v3.0.6、v3.2.1	提供商小部件

次要版本	命令
v3.0.6、v3.2.1	角色添加权限
v3.0.6、v3.2.1	角色删除权限
v3.0.6、v3.2.1	角色创建
v3.0.6、v3.2.1	列出角色
v3.0.6、v3.2.1	清除任务
v3.0.6、v3.2.1	任务失败-部署
v3.0.6、v3.2.1	列出任务
v3.0.6、v3.2.1	渲染任务
v3.0.6、v3.2.1	陈述任务
v3.0.6、v3.2.1	dag 运行的任务状态
v3.0.6、v3.2.1	测试任务
v3.0.6、v3.2.1	删除变量
v3.0.6、v3.2.1	获取变量
v3.0.6、v3.2.1	设置变量

次要版本	命令
v3.0.6、v3.2.1	列出变量
v3.0.6、v3.2.1	版本

Apache Airflow v2

次要版本	命令
v2.0+	备忘单
v2.0+	添加连接
v2.0+	删除连接
v2.2+（注意）	回填 dags
v2.0+	删除 dags
v2.2+（注意）	dags 列表
v2.0+	dags 列表-任务
v2.6+	dags 列表导入错误
v2.2+（注意）	dags 列表-运行次数
v2.2+（注意）	dags 下次-执行
v2.0+	暂停 dag

次要版本	命令
v2.0+	报告 dags
v2.4+	重新序列化 dags
v2.0+	显示 dags
v2.0+	陈述 dags
v2.0+	测试 dags
v2.0+	触发 dags
v2.0+	取消暂停 dags
v2.4+	清理 db
v2.0+	提供商的行为
v2.0+	获得提供商
v2.0+	提供商挂钩
v2.0+	提供商链接
v2.0+	提供商列表
v2.8+	提供程序通知

次要版本	命令
v2.6+	提供商密钥
v2.7+	提供商触发器
v2.0+	提供商小部件
v2.6+	角色添加权限
v2.6+	角色删除权限
v2.6+	角色创建
v2.0+	列出角色
v2.0+	清除任务
v2.0+	任务失败-部署
v2.0+	列出任务
v2.0+	渲染任务
v2.0+	运行任务
v2.0+	陈述任务
v2.0+	dag 运行的任务状态

次要版本	命令
v2.0+	测试任务
v2.0+	删除变量
v2.0+	获取变量
v2.0+	设置变量
v2.0+	列出变量
v2.0+	版本

使用解析 DAG 的命令

如果环境运行的是 Apache Airflow v2.0.2，则如果 DAG 使用的插件依赖于通过 `requirements.txt` 安装的程序包，则解析 DAG 的 CLI 命令将会失败：

Apache Airflow v2.0.2

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

如果 DAG 不使用依赖于通过 `requirements.txt` 安装的程序包的插件，则可以使用这些 CLI 命令。

代码示例

下一节包含使用 Apache Airflow CLI 的不同方法的示例。

设置、获取或删除 Apache Airflow v2 变量

您可以使用以下示例代码设置、获取或删除 `<script> <mwaa env name> get | set | delete <variable> <variable value> </variable> </variable>` 格式的变量。

```
[ $# -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
```

触发 DAG 时添加配置

您可以在 Apache Airflow v2 中使用以下示例代码在触发 DAG 时添加配置，例如 `airflow trigger_dag 'dag_name' –conf '{"key":"value"}'`。

```
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)
```

在通往堡垒主机的 SSH 隧道上运行 CLI 命令。

根据以下示例使用连接到 Linux 堡垒主机的 SSH 隧道代理运行 Airflow CLI 命令。

使用 curl

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
```

管理与 Apache Airflow 的连接

本章介绍如何为 Amazon MWAA 环境配置 Apache Airflow 连接。

主题

Apache Airflow 变量和连接概述

在某些情况下，您可能需要为环境（例如 AWS 配置文件）指定其他连接或变量，或者为 Apache Airflow 元存储中的连接对象添加执行角色，然后从 DAG 内部引用该连接。

- 自行管理的 Apache Airflow。在自行管理的 Apache Airflow 安装中，可以在 `airflow.cfg` 中设置 Apache Airflow 配置选项。

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

- Amazon MWAA 上的 Apache Airflow。在 Amazon MWAA 上，您需要将这些配置设置作为 Apache Airflow 配置选项添加到 Amazon MWAA 控制台上。Apache Airflow 配置选项作为环境变量写入环境，并覆盖相同设置的所有其他现有配置。

安装在 Amazon MWAA 环境中的 Apache Airflow 提供程序包

本页列出了 Amazon MWAA 为所有支持的 Apache Airflow 环境安装的 Apache Airflow 提供程序包。有关这些程序包的更多信息，请参阅程序包 Extras 的 Apache Airflow 参考。

> **Note**
>
> 为了确保与 CloudWatch 日志记录的兼容性不会被其他 Python 库安装所覆盖，Amazon MWAA 在执行后会安装 W atchtower 版本 2.0.1。`pip3 install -r requirements.txt`

主题

约束条件文件

从 Apache Airflow v2.7.2 开始，要求文件必须包含一条 `--constraint` 语句。如果您未提供约束条件，Amazon MWAA 将为您指定一个约束条件，以确保您的要求中列出的程序包与您正在使用的 Apache Airflow 版本兼容。

Apache Airflow 约束条件文件指定了 Apache Airflow 发布时可用的提供程序版本。但是，在许多情况下，较新的提供程序与该版本的 Apache Airflow 兼容。由于必须使用约束条件，因此要指定提供程序包的较新版本，因此可以修改特定提供程序版本的约束文件：

1. 例如，从 GitHub下载特定于版本的限制文件 https://raw.githubusercontent.com/apache/airflow/constraints-2.7.2/constraints-3.11.txt（将 “2.7.2” 替换为要使用的版本）。
2. 将修改后的约束条件文件另存到 Amazon MWAA 环境的 Amazon S3 DAG 文件夹，例如 `constraints-3.11-updated.txt`。
3. 如下所示，指定您的要求。

```
--constraint "/usr/local/airflow/dags/constraints-3.11-updated.txt"
apache-airflow-providers-amazon==version-number
```

> **Note**
>
> 如果您使用的是私有 Web 服务器，我们建议您使用 aws-mwaa-docker-images 将所需的库打包为 WHL 文件。

Version-specific 提供者套餐

安装提供程序包允许您在 Apache Airflow UI 中访问连接类型。这也意味着您无需在 `requirements.txt` 文件中将这些程序包指定为 Python 依赖项。本页列出了 Amazon MWAA 为所有支持的 Apache Airflow 环境安装的 Apache Airflow 提供程序包。

> **Note**
>
> 对于 Apache Airflow v2 及更高版本，亚马逊 MWAA 在`pip3 install -r requirements.txt`执行后会安装 Watchtower 版本 2.0.1，以确保与 CloudWatch 日志的兼容性不会被其他 Python 库安装所覆盖。

您可以指定支持的 `apache-airflow-providers-amazon` 的最新版本来升级此提供程序。

支持的 Apache Airflow 版本：

v3.2.1

连接类型	程序包
AWS 连接	apache-airflow-providers-amazon [aiobotocore] ==9.25.0
Postgres 连接	apache-airflow-providers-postgres==6.6.3
FTP 连接	apache-airflow-providers-ftp==3.14.3
Fab 连接	apache-airflow-providers-fab==3.6.1
Celery 连接	apache-airflow-providers-celery==3.18.0
HTTP 连接	apache-airflow-providers-http==6.0.2
IMAP 连接	apache-airflow-providers-imap==3.11.2
常见 SQL	apache-airflow-providers-common-sql==1.34.0
SQLite 连接	apache-airflow-providers-sqlite==4.3.2

v3.0.6

连接类型	程序包
AWS 连接	apache-airflow-providers-amazon [aiobotocore] ==9.0
Postgres 连接	apache-airflow-providers-postgres==6.2.1
FTP 连接	apache-airflow-providers-ftp==3.13.1
Fab 连接	apache-airflow-providers-fab==2.3.0
Celery 连接	apache-airflow-providers-celery==3.12.1
HTTP 连接	apache-airflow-providers-http==5.3.2
IMAP 连接	apache-airflow-providers-imap==3.9.1
常见 SQL	apache-airflow-providers-common-sql==1.27.3
SQLite 连接	apache-airflow-providers-sqlite==4.1.1

v2.11.0

连接类型	程序包
AWS 连接	apache-airflow-providers-amazon [aiobotocore] ==9.8.0
Postgres 连接	apache-airflow-providers-postgres==6.2.0

连接类型	程序包
FTP 连接	apache-airflow-providers-ftp==3.13.0
Fab 连接	apache-airflow-providers-fab==1.5.3
Celery 连接	apache-airflow-providers-celery==3.11.0
HTTP 连接	apache-airflow-providers-http==5.3.0
IMAP 连接	apache-airflow-providers-imap==3.9.0
常见 SQL	apache-airflow-providers-common-sql==1.27.1
SQLite 连接	apache-airflow-providers-sqlite==4.1.0
SMTP 连接	apache-airflow-providers-smtp==2.1.0

v2.10.3

连接类型	程序包
AWS 连接	apache-airflow-providers-amazon [aiobotocore] ==9.0.0
Postgres 连接	apache-airflow-providers-postgres==5.13.1
FTP 连接	apache-airflow-providers-ftp==3.11.1
Fab 连接	apache-airflow-providers-fab==1.5.0

连接类型	程序包
Celery 连接	apache-airflow-providers-celery==3.8.3
HTTP 连接	apache-airflow-providers-http==4.13.2
IMAP 连接	apache-airflow-providers-imap==3.7.0
常见 SQL	apache-airflow-providers-common-sql==1.19.0
SQLite 连接	apache-airflow-providers-sqlite==3.9.0
SMTP 连接	apache-airflow-providers-smtp==1.8.0

v2.10.1

连接类型	程序包
AWS 连接	apache-airflow-providers-amazon[aiobotocore]==8.28.0
Postgres 连接	apache-airflow-providers-postgres==5.12.0
FTP 连接	apache-airflow-providers-ftp==3.11.0
Fab 连接	apache-airflow-providers-fab==1.3.0
Celery 连接	apache-airflow-providers-celery==3.8.1
HTTP 连接	apache-airflow-providers-http==4.13.0

连接类型	程序包
IMAP 连接	apache-airflow-providers-imap==3.7.0
常见 SQL	apache-airflow-providers-common-sql==1.16.0
SQLite 连接	apache-airflow-providers-sqlite==3.9.0
SMTP 连接	apache-airflow-providers-smtp==1.8.0

v2.9.2

连接类型	程序包
AWS 连接	apache-airflow-providers-amazon[aiobotocore]==8.24.0
Postgres 连接	apache-airflow-providers-postgres==5.11.1
FTP 连接	apache-airflow-providers-ftp==3.9.1
Fab 连接	apache-airflow-providers-fab==1.1.1
Celery 连接	apache-airflow-providers-celery==3.7.2
HTTP 连接	apache-airflow-providers-http==4.11.1
IMAP 连接	apache-airflow-providers-imap==3.6.1
常见 SQL	

连接类型	程序包
	apache-airflow-providers-common-sql==1.14.0
SQLite 连接	apache-airflow-providers-sqlite==3.8.1
SMTP 连接	apache-airflow-providers-smtp==1.7.1

v2.8.1

连接类型	程序包
AWS 连接	apache-airflow-providers-amazon[aiobotocore]==8.16.0
Postgres 连接	apache-airflow-providers-postgres==5.10.0
FTP 连接	apache-airflow-providers-ftp==3.7.0
Celery 连接	apache-airflow-providers-celery==3.5.1
HTTP 连接	apache-airflow-providers-http==4.8.0
IMAP 连接	apache-airflow-providers-imap==3.5.0
常见 SQL	apache-airflow-providers-common-sql==1.10.0
SQLite 连接	apache-airflow-providers-sqlite==3.7.0

v2.7.2

连接类型	程序包
AWS 连接	apache-airflow-providers-amazon [aiobotocore] ==8.7.1
Postgres 连接	apache-airflow-providers-postgres==5.6.1
FTP 连接	apache-airflow-providers-ftp==3.5.2
Celery 连接	apache-airflow-providers-celery==3.3.4
HTTP 连接	apache-airflow-providers-http==4.5.2
IMAP 连接	apache-airflow-providers-imap==3.3.2
常见 SQL	apache-airflow-providers-common-sql==1.7.2
SQLite 连接	apache-airflow-providers-sqlite==3.4.3

v2.6.3

连接类型	程序包
AWS 连接	apache-airflow-providers-amazon [aiobotocore] ==8.2.0
Postgres 连接	apache-airflow-providers-postgres==5.5.1
FTP 连接	apache-airflow-providers-ftp==3.4.2

连接类型	程序包
Celery 连接	apache-airflow-providers-celery==3.2.1
HTTP 连接	apache-airflow-providers-http==4.4.2
IMAP 连接	apache-airflow-providers-imap==3.2.2
常见 SQL	apache-airflow-providers-common-sql==1.5.2
SQLite 连接	apache-airflow-providers-sqlite==3.4.2

v2.5.1

连接类型	程序包
AWS 连接	apache-airflow-providers-amazon==7.1.0
Postgres 连接	apache-airflow-providers-postgres==5.4.0
FTP 连接	apache-airflow-providers-ftp==3.3.0
Celery 连接	apache-airflow-providers-celery==3.1.0
HTTP 连接	apache-airflow-providers-http==4.1.1
IMAP 连接	apache-airflow-providers-imap==3.1.1
常见 SQL	apache-airflow-providers-common-sql==1.3.3
SQLite 连接	apache-airflow-providers-sqlite==3.3.1

v2.4.3

连接类型	程序包
AWS 连接	apache-airflow-providers-amazon==6.0.0
Postgres 连接	apache-airflow-providers-postgres==5.2.2
FTP 连接	apache-airflow-providers-ftp==3.1.0
Celery 连接	apache-airflow-providers-celery==3.0.0
HTTP 连接	apache-airflow-providers-http==4.0.0
IMAP 连接	apache-airflow-providers-imap==3.0.0
常见 SQL	apache-airflow-providers-common-sql==1.2.0
SQLite 连接	apache-airflow-providers-sqlite==3.2.1

连接类型概述

Apache Airflow 将各个连接存储为连接 URI 字符串。它在 Apache Airflow UI 中提供了一个连接模板，用于生成连接 URI 字符串，无论连接类型如何。如果 Apache Airflow UI 中没有连接模板，则可以使用备用连接模板来生成此连接 URI 字符串，例如使用 HTTP 连接模板。主要区别在于 URI 前缀，例如 `my-conn-type://`，Apache Airflow 提供程序在连接中通常会忽略该前缀。本页介绍如何交替使用 Apache Airflow UI 中的连接模板来处理不同的连接类型。

> **Warning**
>
> 请勿覆盖 Amazon MWAA 中的 `aws_default` 连接。Amazon MWAA 使用此连接来执行各种关键任务，例如收集任务日志。覆盖此连接可能会导致数据丢失和环境可用性中断。

主题

连接 URI 字符串示例

以下示例介绍了 MySQL 连接类型的连接 URI 字符串。

```
'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'
```

示例连接模板

以下示例描述了 Apache Airflow UI 中的 HTTP 连接模板。

Apache Airflow v3

Apache Airflow v2

使用 HTTP 连接模板进行 Jdbc 连接的示例

根据以下示例在 Apache Airflow UI 中为 Jdbc 连接类型应用 HTTP 连接模板。

Apache Airflow v3

以下示例显示了 Apache Airflow 为本部分中的示例生成的连接 URI 字符串。

```
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
```

根据以下示例在 Apache Airflow UI 中为 Apache Airflow v3 的 Jdbc 连接应用 HTTP 连接模板。

Apache Airflow v2

以下示例显示了 Apache Airflow 为本部分中的示例生成的连接 URI 字符串。

```
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
```

根据以下示例在 Apache Airflow UI 中为 Apache Airflow v2 的 Jdbc 连接应用 HTTP 连接模板。

使用密钥配置 Apache Airflow 连接 AWS Secrets Manager

AWS Secrets Manager 是适用于 Apache Airflow 的亚马逊托管工作流程环境中支持的备用 Apache Airflow 后端。本主题介绍如何使用 AWS Secrets Manager 在 Apache Airflow 的亚马逊托管工作流程上安全地存储 Apache Airflow 变量和 Apache Airflow 连接的机密。

> **Note**
>
> - 您需要为自己创建的密钥付费。有关 Secrets Manager 定价的更多信息，请参阅 AWS 定价。
> - AWS 亚马逊 MWAA 还支持 S@@ ystems Manager 参数存储作为机密后端。有关更多信息，请参阅 Amazon 提供程序包文档。

目录

步骤 1：向 Amazon MWAA 提供访问 Secrets Manager 密钥的权限

您 Amazon MWAA 环境的执行角色需要对 AWS Secrets Manager中的密钥具有读取权限。以下 IAM 策略允许使用 AWS托管SecretsManagerReadWrite策略进行读写访问。

要将该策略附加到执行角色，请执行以下操作

1. 在 Amazon MWAA 控制台上打开环境页面。
2. 选择环境。
3. 在权限窗格上选择执行角色。
4. 选择附加策略。
5. 在筛选策略文本字段中键入 SecretsManagerReadWrite。
6. 选择附加策略。

如果您不想使用 AWS托管权限策略，则可以直接更新环境的执行角色以允许任何级别的访问您的 Secrets Manager 资源。例如，以下策略声明授予您在 Secrets Manager 中的特定密钥中创建的所有密钥 AWS 区域 的读取权限。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetResourcePolicy",
                "secretsmanager:GetSecretValue",
                "secretsmanager:DescribeSecret",
                "secretsmanager:ListSecretVersionIds"
            ],
            "Resource": "arn:aws:secretsmanager:us-east-1:111122223333:secret:*"
        },
        {
            "Effect": "Allow",
            "Action": "secretsmanager:ListSecrets",
            "Resource": "*"
        }
    ]
```

```
}
```

步骤 2： 创建 Secrets Manager 后端作为 Apache Airflow 配置选项

以下部分介绍如何在亚马逊 MWAA 控制台上为后端创建 Apache Airflow 配置选项。 AWS Secrets Manager 如果您在 `airflow.cfg` 中使用同名的配置设置，则您在以下步骤中创建的配置将优先并覆盖配置设置。

1. 在 Amazon MWAA 控制台上打开环境页面。
2. 选择环境。
3. 选择编辑。
4. 选择下一步。
5. 在 Airflow 配置选项窗格中选择添加自定义配置。添加以下键值对：

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

   b. **`secrets.backend_kwargs`**：**`{"connections_prefix" : "airflow/connections", "variables_prefix" : "airflow/variables"}`**这将 Apache Airflow 配置为在 `airflow/connections/*` 和 `airflow/variables/*` 路径中搜索连接字符串和变量。

      您可以使用查找模式来减少 Amazon MWAA 代表您向 Secrets Manager 调用 API 的次数。如果您未指定查找模式，Apache Airflow 会在已配置的后端中搜索所有连接和变量。通过指定模式，可以收窄 Apache Airflow 搜索的可能路径。这可以降低您在 Amazon MWAA 中使用 Secrets Manager 时的成本。

      要指定查找模式，请指定 `connections_lookup_pattern` 和 `variables_lookup_pattern` 参数。这些参数接受 RegEx 字符串作为输入。例如，要搜索以 `test` 开头的密钥，请输入 `secrets.backend_kwargs` 的以下内容：

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

> **Note**
>
> 要使用 `connections_lookup_pattern` 和 `variables_lookup_pattern`，必须安装 `apache-airflow-providers-amazon` 的 7.3.0 或更高版本。有关将提供程序包更新到新版本的更多信息，请参阅 约束条件文件。

6. 选择保存。

第三步：生成 Apache A AWS irflow 连接 URI 字符串

要创建连接字符串，请使用键盘上的“Tab”键缩进连接对象中的键值对。我们还建议在 shell 会话中为该 `extra` 对象创建一个变量。下一节将引导您完成使用 Apache Airflow 或 Python 脚本为 Amazon MWAA 环境生成 Apache Airflow 连接 URI 字符串的步骤。

Apache Airflow CLI

以下 shell 会话使用本地 Airflow CLI 生成连接字符串。如果您没有安装 CLI，我们建议您使用 Python 脚本。

1. 打开 Python shell 会话：

```
python3
```

2. 输入以下命令：

```
>>> import json
```

3. 输入以下命令：

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

4. 在 shell 会话中为该 `extra` 对象创建一个变量。将中的*YOUR_EXECUTION_ROLE_ARN*样本值替换为执行角色 ARN 和中的区域*us-east-1*（例如us-east-1）。

```
>>> extra=json.dumps({'role_arn': 'YOUR_EXECUTION_ROLE_ARN', 'region_name': 'us-east-1'})
```

5. 创建连接对象。用 Apache Airflow 连接的名称替换 `myconn` 中的示例值。

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

6. 使用键盘上的“Tab”键缩进连接对象中的以下每个键值对。替换中的样本值*red*。

   a. 指定 AWS 连接类型：

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

   b. 指定 Apache Airflow 数据库选项：

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

   c. 在 Amazon MWAA 上指定 Apache Airflow UI 网址：

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

   d. 指定用于登录 Amazon MWAA 的 AWS 访问密钥 ID（用户名）：

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

   e. 指定用于登录 Amazon MWAA 的私有访问 AWS 密钥（密码）：

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

   f. 指定 `extra` shell 会话变量：

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

   g. 关闭连接对象。

   ```
   ... )
   ```

7. 打印连接 URI 字符串：

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

   请参阅响应中的连接 URI 字符串：

```
'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

以下 Python 脚本不需要 Apache Airflow CLI。

1. 复制以下代码示例的内容，并在本地另存为 mwaa_connection.py。

```
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 = 'us-east-1'

        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. 将占位符替换为。*red*
3. 运行以下脚本可生成连接字符串。

```
python3 mwaa_connection.py
```

步骤 4：在 Secrets Manager 中添加变量

下一节介绍如何在 Secrets Manager 中为变量创建密钥。

要创建密钥，请执行以下操作

1. 打开 AWS Secrets Manager 控制台。
2. 选择存储新密钥。

3. 选择其他密钥类型。
4. 在 “指定要存储在此密钥中的 key/value 配对” 窗格中，选择 Plaintex t。
5. 按以下格式将变量值添加为纯文本。

```
"YOUR_VARIABLE_VALUE"
```

例如，要指定一个整数，请执行以下操作：

```
14
```

例如，要指定一个字符串，请执行以下操作：

```
"mystring"
```

6. 对于加密密钥，请从下拉列表中选择一个 AWS KMS 密钥选项。
7. 按以下格式在密钥名称文本字段中输入名称。

```
airflow/variables/YOUR_VARIABLE_NAME
```

例如：

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

8. 选择下一步。
9. 在配置密钥页面的密钥名称和描述窗格上，执行以下操作。

   a. 在密钥名称中，输入密钥名称。

   b. （可选）在描述中，输入密钥名称的描述。

   选择下一步。
10. 在配置轮换-可选上，保留默认选项，然后选择下一步。
11. 对于要添加的任何其他变量，在 Secrets Manager 中重复这些步骤。
12. 在查看 页上，查看您密钥的详细信息，然后选择存储。

步骤 5：在 Secrets Manager 中添加连接

下一节介绍如何在 Secrets Manager 中为连接字符串 URI 创建密钥。

要创建密钥，请执行以下操作

1. 打开 AWS Secrets Manager 控制台。
2. 选择存储新密钥。
3. 选择其他密钥类型。
4. 在 “指定要存储在此密钥中的 key/value 配对” 窗格中，选择 Plaintex t。
5. 按以下格式将连接 URI 字符串添加为纯文本。

```
YOUR_CONNECTION_URI_STRING
```

例如：

```
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 会解析连接字符串中的每个值。不得使用单引号或双引号，否则它会将连接解析为单个字符串。

6. 对于加密密钥，请从下拉列表中选择一个 AWS KMS 密钥选项。
7. 按以下格式在密钥名称文本字段中输入名称。

```
airflow/connections/YOUR_CONNECTION_NAME
```

例如：

```
airflow/connections/myconn
```

8. 选择下一步。
9. 在配置密钥页面的密钥名称和描述窗格上，执行以下操作。

a. 在密钥名称中，输入密钥名称。

b. （可选）在描述中，输入密钥名称的描述。

选择下一步。

10. 在配置轮换-可选上，保留默认选项，然后选择下一步。
11. 对于要添加的任何其他变量，在 Secrets Manager 中重复这些步骤。
12. 在查看 页上，查看您密钥的详细信息，然后选择存储。

代码示例

- 要了解在使用以下示例代码的本页上如何使用 Apache Airflow 连接（`myconn`）的密钥，请参阅 使用 AWS Secrets Manager 中的密钥进行 Apache Airflow 连接。
- 要了解在使用以下示例代码的本页上如何使用 Apache Airflow 变量（`test-variable`）的密钥，请参阅 为 Apache Airflow 变量使用 AWS Secrets Manager 中的密钥。

资源

- 有关使用控制台和配置 Secrets Manager 密钥的更多信息 AWS CLI，请参阅AWS Secrets Manager 用户指南中的创建密钥。
- 在将 Apache Airflow 连接和变量移动至 AWS Secrets Manager中，使用 Python 脚本将大量 Apache Airflow 变量和连接迁移到 Secrets Manager。

接下来做什么？

- 要了解如何生成令牌以访问 Apache Airflow UI，请参阅 访问 Apache Airflow。

管理 Amazon MWAA 环境

Amazon MWAA 控制台包含内置选项，用于配置对 Apache Airflow UI 的私有或公开访问权限。该控制台还包含内置选项（以配置环境大小、何时扩展工作线程）以及 Apache Airflow 配置选项（可用来覆盖通常只能在 `airflow.cfg` 中访问的 Apache Airflow 配置）。本章介绍了如何在 Amazon MWAA 控制台上使用这些配置。

主题

配置 Amazon MWAA 环境类

你为亚马逊 MWAA 环境选择的环境类决定了运行 Celery Executo r 的 AWS托管 AWS Fargate 容器的大小，以及 Apache Airflow 计划程序在其中创建任务实例的托管的 Amazon AWS Aurora PostgreSQL 元数据数据库的大小。本主题描述了每个 Amazon MWAA 环境类，以及如何在 Amazon MWAA 控制台上更新环境类。

Sections

环境功能

下一节包含每个环境类的默认并发 Apache Airflow 任务、随机存取存储器（RAM）和虚拟集中处理单元（vCPU）。列出的并发任务假设任务并发性不超过环境中的 Apache Airflow 工作线程容量。

在下表中，DAG 容量是指 DAG 定义数，而不是执行数，并且假设您的 DAG 在单个 Python 文件中是动态的，是遵循 Apache Airflow 最佳实践编写的。

任务执行取决于同时安排了多少任务，并假设设置为同时启动的 DAG 运行次数不超过默认值 `max_dagruns_per_loop_to_schedule`，以及本主题中详细介绍的工作线程的大小和数量。

mw1.micro

- 高达 25 个 DAG 的容量
- 3 个并发任务（默认）
- 组件：
  - Web 服务器：1 个 vCPU，3 GB RAM 内存
  - 工作线程和计划程序：1 个 vCPU，3 GB RAM 内存
  - 数据库：2 个 vCPU，4 GB RAM 内存

> **Note**
>
> mw1.micro 不支持自动扩缩。

mw1.small

- 高达 50 DAG 的容量
- 5 个并发任务（默认）
- 组件：
  - Web 服务器：每个服务器 1 个 vCPU，2 GB RAM 内存
  - Worker 节点：每个节点 1 个 vCPU，2 GB RAM 内存
  - 调度器：每个调度器 1 个 vCPU，2 GB RAM 内存
  - 数据库：2 个 vCPU，4 GB RAM 内存

mw1.medium

- 高达 250 DAG 的容量
- 10 个并发任务（默认）
- 组件：
  - Web 服务器：每个服务器 1 个 vCPU，2 GB RAM 内存
  - Worker 节点：每个节点 2 个 vCPU，4 GB RAM 内存
  - 调度器：每个调度器 2 个 vCPU，4 GB RAM 内存

- 数据库：2 个 vCPU，8 GB RAM 内存

mw1.large

- 高达 1000 DAG 的容量
- 20 个并发任务（默认）
- 组件：
  - Web 服务器：每个服务器 2 个 vCPU，4 GB RAM 内存
  - Worker 节点：每个节点 4 个 vCPU，8 GB RAM 内存
  - 调度器：每个调度器 4 个 vCPU，8 GB RAM 内存
  - 数据库：2 个 vCPU，8 GB RAM 内存

mw1.xlarge

- 高达 2000 DAG 的容量
- 40 个并发任务（默认）
- 组件：
  - Web 服务器：每个服务器 4 个 vCPU，12 GB RAM 内存
  - Worker 节点：每个节点 8 个 vCPU，24 GB RAM 内存
  - 调度器：每个调度器 8 个 vCPU，24 GB RAM 内存
  - 数据库：4 个 vCPU，32 GB RAM 内存

mw1.2xlarge

- 高达 4000 DAG 的容量
- 80 个并发任务（默认）
- 组件：
  - Web 服务器：每个服务器 8 个 vCPU，24 GB RAM 内存
  - Worker 节点：每个节点 16 个 vCPU，48 GB RAM 内存
  - 调度器：每个调度器 16 个 vCPU，48 GB RAM 内存
  - 数据库：8 个 vCPU，64 GB RAM 内存

您可以使用 `celery.worker_autoscale` 来增加每个工作线程的任务数。有关更多信息，请参阅 the section called “高性能用例示例”。

Apache Airflow 计划程序

下一节包含 Amazon MWAA 上可用的 Apache Airflow 计划程序选项，以及计划程序数如何影响触发器数。

在 Apache Airflow 中，触发器负责管理由其延迟的任务，直到满足使用触发器指定的特定条件时为止。在 Amazon MWAA 中，触发器与计划程序一起运行相同的 Fargate 任务。增加计划程序计数会相应地增加可用触发器的数量，从而优化环境管理延迟任务的方式。这样可以确保高效处理任务，在条件满足时及时安排任务运行。

Apache Airflow v3

- v3 - 对于大于 mw1.micro 的环境，接受从 2 到 5 的值。除了 mw1.micro 之外，所有环境大小的默认值均为 2，而 mw1.micro 默认为 1。

Apache Airflow v2

- v2 - 对于大于 mw1.micro 的环境，接受从 2 到 5 的值。除了 mw1.micro 之外，所有环境大小的默认值均为 2，而 mw1.micro 默认为 1。

配置 Amazon MWAA Worker 节点自动扩缩

自动扩缩机制会根据 Amazon Managed Workflows for Apache Airflow 环境中正在运行和排队的任务数量，自动增加 Apache Airflow Worker 节点数量，并在没有其他任务排队或正在执行时停止多余的 Worker 节点。本主题介绍如何使用 Amazon MWAA 控制台指定在环境中运行的最大 Apache Airflow Worker 工作线程数，从而配置自动扩缩。

> **Note**
>
> Amazon MWAA 使用 Apache Airflow 指标来确定何时需要额外的 Celery 执行程序工作线程，并根据需要将 Fargate 工作线程数增加到 `max-workers` 指定的值。随着额外 Worker 节点完成工作和工作负载的减少，Amazon MWAA 会将其移除，从而缩减到 `min-workers` 设定的值。
>
> 如果 Worker 节点在缩减的同时接到新任务，Amazon MWAA 会保留 Fargate 资源并且不会移除该 Worker 节点。有关更多信息，请参阅 Amazon MWAA 自动扩缩的工作原理。

Sections

Worker 节点扩缩的工作原理

Amazon MWAA 使用 `RunningTasks` 和 `QueuedTasks` 指标，其中（正在运行的任务 + 排队的任务）/（每个工作线程的任务数）=（所需工作线程）。如果所需的工作线程数大于当前工作线程数，Amazon MWAA 将在该值中添加 Fargate 工作线程容器，但不得超过 `max-workers` 指定的最大值。

随着工作负载减少以及 `RunningTasks` 与 `QueuedTasks` 指标之和下降，Amazon MWAA 会请求 Fargate 缩减环境的 Worker 节点数。任何在缩减期间仍在完成工作的 Worker 节点继续受到保护，直到完成工作为止。根据具体的工作负载，工作线程缩减时任务可能会排队等候。

使用 Amazon MWAA 控制台

您可以在 Amazon MWAA 控制台上选择可在环境中同时运行的最大工作线程数。默认情况下，您可以指定最大值，最大值为 25。

要配置工作线程数，请执行以下操作

1. 在 Amazon MWAA 控制台上打开环境页面。
2. 选择环境。
3. 选择编辑。
4. 选择下一步。
5. 在环境类窗格上，在最大工作线程计数中输入一个值。
6. 选择保存。

> **Note**
>
> 更改可能需要几分钟才能生效。

高性能用例示例

下一节介绍可用于在环境中实现高性能和并行性的配置类型。

On-premise Apache Airflow

通常，在本地 Apache Airflow 平台中，您在 `airflow.cfg` 文件中配置任务并行度、自动扩缩和并发设置：

- `core.parallelism`— 每个计划程序可以同时运行的最大任务实例数。
- `core.dag_concurrency`— DAG（非工作线程）的最大并发度。
- `celery.worker_autoscale`— 可在任何工作线程上同时运行的最大和最小任务数。

例如，如果 `core.parallelism` 设置为 `100` 且 `core.dag_concurrency` 设置为 `7`，则只有在拥有 2 个 DAG 的情况下，您才能够同时运行总共 14 个任务。假设，即使总体并行度设置为 `100`（在 `core.parallelism` 中），每个 DAG 也只能同时运行七个任务（在 `core.dag_concurrency` 中）。

> **Note**
>
> `core.dag_concurrency` 在 Apache Airflow v3 中不可用。

在Amazon MWAA 环境中

在 Amazon MWAA 环境中，您可以直接在 Amazon MWAA 控制台上使用 在 Amazon MWAA 上使用 Apache Airflow 配置选项、配置 Amazon MWAA 环境类 和 最大 Worker 节点数自动扩缩机制来配置这些设置。虽然在 Amazon MWAA 控制台中，`core.dag_concurrency` 并未作为一个 Apache Airflow 配置选项在下拉列表中出现，但您可以将其作为自定义 Apache Airflow 配置选项添加。

比方说，当您创建环境时，您选择了以下设置：

1. mw1.small 环境类，用于控制默认情况下每个工作线程可以运行的最大并发任务数和容器的 vCPU。
2. 最大工作线程计数中的 `10` 工作线程的默认设置。
3. Apache Airflow 配置选项，适用于每个工作线程的 `5,5` 个任务的 `celery.worker_autoscale`。

这意味着您可以在环境中运行 50 个并发任务。任何超过 50 的任务都将排队，并等待正在运行的任务完成。

运行更多并发任务。您可以使用以下配置修改环境以同时运行更多任务：

1. 通过选择 `mw1.medium`（默认为 10 个并发任务）环境类，增加每个工作线程默认可以运行的最大并发任务数和各个容器的 vCPU。
2. 添加 `celery.worker_autoscale` 作为 Apache Airflow 配置选项。
3. 增加最大工作线程计数。在此示例中，将最大工作线程数从 `10` 增加到 `20` 会使环境可以运行的并发任务数增加一倍。

指定最低工作线程数。您还可以使用 () 指定在您的环境中运行的 Apache Airflow 工作程序的最小和最大数量。 AWS Command Line Interface AWS CLI例如：

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

要了解更多信息，请参阅 AWS CLI中的 update-environment 命令。

对停留在运行状态的任务进行故障排除

在极少数情况下，Apache Airflow 可能会认为还有任务在运行。要解决此问题，您需要清除 Apache Airflow UI 中的滞留任务。有关更多信息，请参阅 Amazon MWAA 故障排除 故障排除主题。

接下来做什么？

- 要详细了解我们推荐的调整环境性能的最佳实践，请参阅 Amazon MWAA 上的 Apache Airflow 的性能调整。

配置 Amazon MWAA Web 服务器自动扩缩

对于运行 Apache Airflow v2.2.2 及更高版本的环境，Amazon MWAA 会动态扩展 Web 服务器以处理持续波动的工作负载，这反过来又可以防止在峰值负载期间出现性能问题。通过根据 CPU 利用率和活动连接数自动扩缩 Web 服务器的数量，Amazon MWAA 可确保您的 Apache Airflow 环境能够无缝满足增加的需求，无论是来自 REST API 请求、CLI 使用还是并发 Apache Airflow 用户界面用户数量的增加。

各个部分

Web 服务器扩缩的工作原理

Amazon MWAA 使用容器指标 `CPUUtilization` 和负载均衡器指标 `ActiveConnectionCount`，从而根据流量大小确定是否需要扩缩 Web 服务器。如果 `CPUUtilization` 大于 70 或者 `ActiveConnectionCount` 大于 15，Amazon MWAA 将添加额外的 Fargate Web 服务器容器，最高不超过 `MaxWebservers` 指定的最大值。

随着流量减少以及 `CPUUtilization` 和 `ActiveConnectionCount` 值的下降，Amazon MWAA 会请求 Fargate 将环境的 Web 服务器容器数缩减至 `MinimumWebservers` 设定的最小值。

使用 Amazon MWAA 控制台

您可以在 Amazon MWAA 控制台上选择可在环境中同时运行的 Web 服务器数。默认情况下，Web 服务器数最小为两个，最大为五个。

配置 Web 服务器数

1. 在 Amazon MWAA 控制台上打开环境页面。
2. 选择环境。
3. 选择编辑。
4. 选择下一步。
5. 在环境类窗格的最大 Web 服务器数中输入一个值。
6. 然后在最小 Web 服务器数中输入一个值。
7. 选择保存。

> **Note**
>
> 更改可能需要几分钟才能生效。

在 Amazon MWAA 上使用 Apache Airflow 配置选项

Apache Airflow 配置选项可以作为环境变量附加到 Amazon MWAA 环境中。您可以从建议的下拉列表中进行选择，也可以在 Amazon MWAA 控制台上为 Apache Airflow 版本指定自定义配置选项。本主题介绍可用的 Apache Airflow 配置选项，以及如何使用这些选项来覆盖环境中的 Apache Airflow 配置设置。

目录

先决条件

在完成本页上的步骤之前，您需要具备以下条件。

- 权限-您的管理员 AWS 账户 必须已授予您访问您环境的AmazonMWAAFullConsoleAccess访问控制策略的权限。此外，您的执行角色必须允许您的 Amazon MWAA 环境访问您的环境所使用的 AWS 资源。
- 访问权限 — 如果您需要访问公共存储库以便直接在 Web 服务器上安装依赖项，则必须将环境配置为具有公共网络 Web 服务器访问权限。有关更多信息，请参阅the section called “Apache Airflow 访问模式”。
- Amazon S3 配置 — 用于存储 DAG 的 Amazon S3 存储桶、在 `plugins.zip` 中的自定义插件和在 `requirements.txt` 中的 Python 依赖项必须配置为已阻止公共访问和已启用版本控制。

工作原理

创建环境时，Amazon MWAA 会将您在亚马逊 MWAA 控制台的 A irflow 配置选项中指定的配置设置作为环境变量附加到环境容器中 AWS Fargate 。如果您在 `airflow.cfg` 中使用同名设置，则您在 Amazon MWAA 控制台上指定的选项将覆盖 `airflow.cfg` 中的值。

虽然默认情况下我们不会在 Amazon MWAA 环境的 Apache Airflow UI 中公开 `airflow.cfg`，但您可以直接在 Amazon MWAA 控制台上更改 Apache Airflow 配置选项，然后通过设置 `webserver.expose_config` 来公开配置。

使用配置选项加载插件

默认情况下，在 Apache Airflow v2 和更高版本中，使用 `core.lazy_load_plugins : True` 设置将插件配置为“延迟”加载。如果您使用自定义插件，则必须添加 `core.lazy_load_plugins : False` 为 Apache Airflow 配置选项，以便在每个 Airflow 流程开始时加载插件，从而覆盖默认设置。

配置选项概述

当您在 Amazon MWAA 控制台上添加配置时，Amazon MWAA 会将该配置作为环境变量写入。

- 列出的选项。您可以在下拉列表中从适用于 Apache Airflow 的版本中选择一项配置设置。例如 `dag_concurrency : 16`。配置设置将环境的 Fargate 容器转化为 `AIRFLOW__CORE__DAG_CONCURRENCY : 16`
- 自定义选项。您还可以指定未在下拉列表中列出的 Apache Airflow 版本的 Airflow 配置选项。例如 `foo.user : YOUR_USER_NAME`。配置设置将环境的 Fargate 容器转化为 `AIRFLOW__FOO__USER : YOUR_USER_NAME`

Apache Airflow 配置选项

下图显示了您可以在 Amazon MWAA 控制台上自定义 Apache Airflow 配置选项的位置。

Apache Airflow 参考

有关 Apache Airflow 支持的配置选项列表，请参阅《Apache Airflow 参考指南》中的配置参考。要访问您在 Amazon MWAA 上运行的 Apache Airflow 版本的选项，请从下拉列表中选择版本。

使用 Amazon MWAA 控制台

以下过程将指导您完成将 Airflow 配置选项添加到环境中的步骤。

1. 在 Amazon MWAA 控制台上打开环境页面。
2. 选择环境。
3. 选择编辑。
4. 选择下一步。
5. 在 Airflow 配置选项窗格中选择添加自定义配置。
6. 从下拉列表中选择配置并输入值，或者输入自定义配置并输入值。
7. 为每个您想要添加的配置选择添加自定义配置选项。
8. 选择保存。

配置参考

下一节包含 Amazon MWAA 控制台下拉列表中可用的 Apache Airflow 配置列表。

电子邮件配置

以下列表显示了 Amazon MWAA 上可用于 Apache Airflow v2 和 v3 的 Airflow 电子邮件通知配置选项。

我们建议对 SMTP 流量使用端口 587。默认情况下，会 AWS 阻止所有 Amazon EC2 实例的端口 25 上的出站 SMTP 流量。要在端口 25 上发送出站流量，可请求移除此限制。

Airflow 配置选项	说明	示例值

Airflow 配置选项	说明	示例值
email.email_backend	Apache Airflow 实用工具用于在 email_backend 中发送电子邮件通知。	airflow.utils.email.send_email_smtp
smtp.smtp_host	在 smtp_host 中用作电子邮件地址的出站服务器的名称。	localhost
smtp.smtp_starttls	在 smtp_startttls 中，传输层安全性协议 (TLS) 用于加密互联网上的电子邮件。	False
smtp.smtp_sll	安全套接字层（SSL）用于连接 smtp_ssl 中的服务器和电子邮件客户端。	True
smtp.smtp_port	在 smtp_port 中为服务器指定的传输控制协议（TCP）端口。	587
smtp.smtp_mail_from	smtp_mail_from 中的出站电子邮件地址。	myemail@domain.com

任务配置数

以下列表显示了 Amazon MWAA 上可用于 Apache Airflow v2 和 v3 的 Airflow 任务下拉列表中可用的配置。

Airflow 配置选项	说明	示例值
core.default_task_ret	在 default_task_retries 中重试 Apache Airflow 任务的次数。	3

Airflow 配置选项	说明	示例值
core.parallelism	可以在整个环境中并行运行的最大任务实例数（并行）。	40

计划程序配置数

以下列表显示了 Amazon MWAA 上可用于 Apache Airflow v2 和 v3 的下拉列表中可用的 Apache Airflow 计划程序配置。

Airflow 配置选项	说明	示例值
scheduler.catchup_by_default	告诉计划程序创建 DAG 运行以“赶上”在 catchup_by_default 中的特定时间间隔。	False
scheduler.scheduler_zombie_task_task_ Note 在 Apache Airflow v3 中不可用。	告诉计划程序是否将任务实例标记为失败并在 scheduler_zombie_task_treshold 中重新安排任务。	300

工作线程配置数

以下列表显示了 Amazon MWAA 上可用于 Apache Airflow v2 和 v3 的下拉列表中可用的 Airflow 工作线程配置。

Airflow 配置选项	说明	示例值
celery.worker_autoscale	在 worker_autoscale 中使用 Celery Executor 在任何工作线程上同时运行的最大和最	16,12

Airflow 配置选项	说明	示例值
	小任务数。值必须按以下顺序以逗号分隔：`max_concurrency,min_concurrency` 。	

Web 服务器配置

以下列表显示了 Amazon MWAA 上可用于 Apache Airflow v2 和 v3 的下拉列表中可用的 Apache Airflow Web 服务器配置。

Airflow 配置选项	说明	示例值
网络服务器.default_ui_timezone **Note** 在 Apache Airflow v3 中不可用。	在 default_ui_timezone 中的默认 Apache Airflow UI 的日期时间设置。 **Note** 设置 `default_ui_timezone` 选项不会更改 DAG 计划运行的时区。要更改 DAG 的时区，您可以使用自定义插件。有关更多信息，请参阅the section called “更改 DAG 的时区”。	America/New_York

触发器配置

以下列表显示了在 Amazon MWAA 上可用于 Apache Airflow v2 和 v3 的 Apache Airflow 触发器配置。

Airflow 配置选项	说明	示例值
mwaa.triggerer_enabled	用于在 Amazon MWAA 上激活和停用触发器。默认情况下，该值设置为 `True`。如果设置为 `False`，Amazon MWAA 将不会在计划程序上启动任何触发器进程。	True
triggerer.default_capacity（在 v2 中） trigger.capacity（在 v3 中）	定义每个触发器可以并行运行的触发器数量。在 Amazon MWAA 上，此容量是按每个触发器和每个计划程序设置的，因为这两个组件并排运行。对于 small、medium、large、xlarge 和 2xlarge实例，每个调度器的默认值分别设置为 `60`、`125`、`250`、`500` 和 `1000`。	125

不支持的配置

以下 Apache Airflow 配置选项在亚马逊 MWAA 中不可用。您无法使用 Amazon MWAA 控制台或 API 设置或覆盖这些选项。

Airflow 配置选项	Apache Airflow 版本	默认 值	Reason
核心多人组	3.2	False	Amazon MWAA 目前不支持多团队模式。启用此功能与 Amazon MWAA 身份验证 CeleryExecutor、和环境级密钥管理不兼容。

Airflow 配置选项	Apache Airflow 版本	默认 值	Reason
触发器.queues_enabled	3.2	False	Amazon MWAA 不支持触发器队列分配。启用此选项会导致延迟的任务无限期挂起。

示例和示例代码

示例 DAG

您可以使用以下 DAG 来打印 `email_backend` Apache Airflow 配置选项。要运行以响应 Amazon MWAA 事件，请将代码复制到 Amazon S3 存储桶上环境的 DAG 文件夹。

```
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()
```

示例电子邮件通知设置

以下 Apache Airflow 配置选项可用于使用应用程序 Gmail.com 密码的电子邮件帐户。有关更多信息，请参阅《Gmail 帮助参考指南》中的使用应用程序密码登录。

接下来做什么？

- 要了解如何将 DAG 文件夹上传到 Amazon S3 存储桶，请参阅 添加或更新 DAG。

更新 Amazon MWAA 环境

> **Note**
>
> 在加拿大西部（卡尔加里）和亚太地区（马来西亚）区域尚不支持 Amazon MWAA 正常更新。

Amazon MWAA 环境更新会应用最新更改和安全补丁。您还可以编辑现有配置并升级 Apache Airflow 版本。本指南介绍更新 Amazon MWAA 环境的步骤。

目录

开始前的准备工作

- 环境创建后，将无法修改为环境指定的 VPC 网络。
- 您需要将 Amazon S3 存储桶配置为阻止所有公开访问并启用存储桶版本控制。

- 您需要 AWS 账户 具有使用 Amazon MWAA 的权限，以及 AWS Identity and Access Management (IAM) 中的权限才能创建 IAM 角色。如果您为 Apache Airflow Web 服务器选择了私有网络访问模式，由于该模式会限制 Amazon VPC 内的 Apache Airflow 访问权限，因此您需要 IAM 中的权限才能创建 Amazon VPC 端点。
- 要启用正常环境更新，需要升级到 Apache Airflow 版本 2.4.3 或更高版本。要升级 Airflow 版本，请参阅 更改 Apache Airflow 版本。

工作线程替换策略

您可以选择工作线程替换策略来控制 Amazon MWAA 在环境更新期间如何处理活跃的工作线程。您可以选择下列策略之一：

强制更新

强制更新是默认的工作线程替换策略。强制更新会立即停止所有活跃的工作进程，从而导致正在运行的任务在更新期间失败。

正常更新

正常更新支持工作线程在关闭前继续运行任务长达 12 小时。只要任务在 12 小时内完成，它就可以防止任务因更新中断而失败。新任务将路由到更新后的工作线程。

要在现有环境上启用正常更新，必须完成一次强制更新，并确保环境在 Apache Airflow 版本 2.4.3 或更高版本上。

> **Note**
>
> 如果您在环境处于 `MAINTENANCE` 状态时执行更新，则任何正在进行的环境更新的工作线程替换策略将从 `GRACEFUL` 切换到 `FORCED`。在维护完成后，将执行更新。

更新环境资源

默认情况下，Amazon MWAA 环境更新使用现有环境配置。要更新环境但不更改当前配置，请执行以下操作：

1. 在 Amazon MWAA 控制台上打开环境页面。
2. 从环境列表中，选择要更新的环境。

3. 在环境页面上，选择编辑以编辑环境。
4. 选择下一步，直到进入查看并保存页面。
5. 在查看并保存页面上，查看更改，然后选择保存。

更新环境

下一节介绍更新 Amazon MWAA 环境的步骤。

步骤 1：指定详细信息

要指定环境的详细信息，请执行以下操作

1. 在 Amazon MWAA 控制台上打开环境页面。
2. 从环境列表中，选择要更新的环境。
3. 在环境页面上，选择编辑以编辑环境。
4. 在环境详细信息部分，对于 Airflow 版本，从下拉列表中选择要将环境升级到的新 Apache Airflow 版本号。

> **Note**
>
> 在升级之前，请确保 DAG 和其他工作流程资源与新 Apache Airflow 版本兼容。有关更多信息，请参阅更改 Apache Airflow 版本。

5. 在 Amazon S3 的 DAG 代码下指定以下内容：
   a. S3 Bucket。选择浏览 S3 并选择 Amazon S3 存储桶，或者输入 Amazon S3 URI。
   b. DAG 文件夹。选择浏览 S3，然后选择 Amazon S3 存储桶中的 `dags` 文件夹，或者输入 Amazon S3 URI。
   c. 插件文件-可选。选择浏览 S3，然后选择 Amazon S3 存储桶上的 `plugins.zip` 文件，或者输入 Amazon S3 URI。
   d. 要求文件-可选。选择浏览 S3，然后选择 Amazon S3 存储桶上的 `requirements.txt` 文件，或者输入 Amazon S3 URI。
   e. 启动脚本文件-可选，选择浏览 S3并选择 Amazon S3 存储桶上的脚本文件，或者输入 Amazon S3 URI。
6. 选择下一步。

步骤 2：配置高级设置

配置高级设置

1. 在 Web 服务器访问下，选择您首选的 Apache Airflow 访问模式：

   a. 私有网络。这限制了对 Apache Airflow UI 的访问，只有在 Amazon VPC 中已获得 环境的 IAM 策略访问权限的用户才能访问。您需要获得权限才能为此步骤创建 Amazon VPC 端点。

   > **Note**
   >
   > 如果 Apache Airflow UI 只能在公司网络中访问，并且不需要访问公共存储库即可进行 Web 服务器要求安装，请选择私有网络选项。如果您选择此访问模式选项，则需要创建一种机制来访问 Amazon VPC 中的 Apache Airflow Web 服务器。有关更多信息，请参阅访问 Apache Airflow Web 服务器的 VPC 端点（私有网络访问）。

   b. 公有网络。这使获得环境的 IAM 策略访问权限的用户可以通过互联网访问 Apache Airflow UI。

   c. 公网和私网均可访问。适用于 Apache Airflow 版本 3.2.1 及更高版本。这允许通过互联网访问 Apache Airflow 用户界面，而工作人员则通过私有 VPC 端点与网络服务器通信。如果托管您的环境的 Amazon VPC 无法访问互联网，请选择此选项。

2. 在安全组下，选择用于保护 Amazon VPC 的安全组：

   a. 默认情况下，Amazon MWAA 会在 Amazon VPC 中创建一个安全组，并在创建新安全组中使用特定的入站和出站规则。

   b. 可选。取消选中创建新安全组中的复选框可选择最多 5 个安全组。

   > **Note**
   >
   > 现有 Amazon VPC 安全组必须配置特定的入站和出站规则，才能允许网络流量。要了解更多信息，请参阅 Amazon MWAA 上的 VPC 安全。

3. 在环境类下，选择一个环境类。

   我们建议选择支持您的工作负载所需的最小尺寸。您可以随时更改环境类。

4. 对于最大工作线程计数，请指定要在环境中运行的 Apache Airflow 工作线程的最大数量。

   有关更多信息，请参阅高性能用例示例。

5. 指定最大 Web 服务器数和最小 Web 服务器数，以配置 Amazon MWAA 如何在环境中扩展 Apache Airflow Web 服务器。

   有关 Web 服务器自动扩展的更多信息，请参阅 the section called “配置 Web 服务器自动扩缩”。

6. 在加密下，选择一个数据加密选项：

   a. 默认情况下，Amazon MWAA 使用 AWS拥有的密钥来加密您的数据。

   b. 可选。选择 “自定义加密设置（高级）” 以选择其他 AWS KMS 密钥。如果您选择在此步骤中指定Customer-managed 密钥，则必须指定 AWS KMS 密钥 ID 或 ARN。AWS KMS A@@mazon MWAA 不支持别名和多区域密钥。如果您在 Amazon S3 存储桶上指定了用于服务器端加密的 Amazon S3 密钥，则必须为 Amazon MWAA 环境指定相同的密钥。

   > **Note**
   >
   > 您必须拥有该密钥的权限才能在 Amazon MWAA 控制台上选择该密钥。您还必须通过 附加密钥政策 中所述的附加策略授予 Amazon MWAA 使用密钥的权限。

7. 推荐。在 “监控” 下，为 Airflo w 日志配置选择一个或多个日志类别，将 Apache Airflow 日志发送到日志： CloudWatch

   a. Airflow 任务日志。选择要发送到 “登录日志” 级别的 Apache Airflow 任务日志 CloudWatch 的类型。

   b. Airflow Web 服务器日志。选择要发送到 “登录日志” 级别的 Apache Airflow 网络服务器日志的类型 CloudWatch 。

   c. Airflow 计划程序日志。选择要发送到 “登录日志” 级别的 Apache Airflow 调度程序日志的类型 CloudWatch 。

   d. Airflow 工作线程日志。选择要发送到 “登录日志” 级别的 Apache Airflow 工作日志 CloudWatch 的类型。

   e. Airflow DAG 处理日志。选择要发送到 “登录日志” 级别的 Apache Airflow DAG 处理日志 CloudWatch 的类型。

8. 可选。对于 Airflow 配置选项，选择添加自定义配置选项。

   您可以从 Apache Airflow 版本的 Apache Airflow 配置选项的建议下拉列表中进行选择，也可以指定自定义配置选项。例如 `core.default_task_retries`：3。

9. 在权限下，选择一个执行角色。

a. 默认情况下，Amazon MWAA 会在创建新角色中创建一个执行角色。您必须具有创建 IAM 角色的权限，才能使用此选项。

b. 可选。选择输入角色 ARN 输入现有执行角色的 Amazon 资源名称（ARN）。

10. 在更新规范下，选择 a 工作线程替换策略 以控制在更新期间如何处理活跃工作线程。

11. 选择下一步。

步骤 3：审核并更新

查看环境摘要

- 查看环境摘要，选择保存。

> **Note**
>
> 使用强制更新方法更新环境大约需要二十到三十分钟。正常环境更新可能需要长达十二个小时才能完成，因为它需要等待您正在进行的任务完成。

更改 Apache Airflow 版本

Amazon MWAA 支持次要版本升级和降级。这意味着您可以将环境从版本 `x.4.z` 更新到 `x.5.z`，或从 `x.5.z` 更新到 `x.4.z`。要执行主要版本升级（例如从版本 `1.y.z` 升级到 `2.y.z`），必须创建新环境并迁移资源。有关升级到 Apache Airflow 的新主要版本的更多信息，请参阅《Amazon MWAA 迁移指南》中的迁移到新的 Amazon MWAA 环境。

在升级或降级过程中，Amazon MWAA 会捕获环境的元数据快照，将工作线程、计划程序、Web 服务器升级或降级到新的 Apache Airflow 版本，最后使用快照恢复元数据数据库。

在升级或降级之前，请确保 DAG 和其他工作流程资源与您要升级到的新 Apache Airflow 版本兼容。如果您使用 `requirements.txt` 来管理依赖项，则还必须确保您在要求中指定的依赖项与新版本兼容。

主题

- 升级或降级工作流程资源
- 指定新版本

升级或降级工作流程资源

每当您更改 Apache Airflow 版本时，请确保在 `requirements.txt` 中引用正确的 `--constraint` URL。

> **Warning**
>
> 在升级或降级期间指定与目标 Apache Airflow 版本不兼容的要求可能会导致回滚到具有先前要求版本的 Apache Airflow 先前版本的过程很长。

迁移工作流程资源

1. 创建 aws-mwaa-docker-images 存储库的分支，然后克隆 Amazon MWAA 本地运行器的副本。
2. 查看与您要升级或降级到的版本匹配的 aws-mwaa-docker-images 存储库的分支。
3. 要更新 `requirements.txt`，请按照《Amazon MWAA 用户指南》中管理 Python 依赖项中推荐的最佳实践进行操作。
4. （可选）要加快升级或降级过程，请清理环境的元数据数据库。具有大量元数据的环境可能需要更长的时间才能升级。
5. 成功测试工作流程资源后，将 DAG、`requirements.txt` 和插件复制到环境的 Amazon S3 存储桶。

现在，您可以编辑环境、指定新的 Apache Airflow 版本并开始更新过程了。

指定新版本

更新工作流程资源以确保与新 Apache Airflow 版本兼容后，请执行以下操作来编辑环境详细信息并指定要升级到的 Apache Airflow 版本。

> **Note**
>
> 执行升级或降级时，当前在环境中运行的所有任务都将在这个过程中终止。更新过程最多可能需要两个小时，在此期间，环境将不可用。

使用控制台指定新版本

1. 在 Amazon MWAA 控制台上打开环境页面。

2. 从环境列表中，选择要升级或降级的环境。
3. 在环境页面上，选择编辑以编辑环境。
4. 在环境详细信息部分中，对于 Airflow 版本，从下拉列表中选择要将环境升级或降级到的 Apache Airflow 版本号。
5. 选择下一步，直到进入查看并保存页面。
6. 在查看并保存页面上，查看更改，然后选择保存。

当您应用更改时，环境将开始升级或降级过程。在此期间，环境状态表明 Amazon MWAA 正在采取哪些操作以及该过程是否成功。

在成功升级或降级的情况下，状态将为 UPDATING，然后在 Amazon MWAA 捕获元数据备份时变为 CREATING_SNAPSHOT。最后，状态将首先返回到 UPDATING，然后过程完成时，返回到 AVAILABLE。

如果环境升级或降级失败，则环境状态是 ROLLING_BACK。如果回滚成功，则状态将首先显示 UPDATE_FAILED，表示更新失败但环境可用。如果回滚失败，则状态是 UNAVAILABLE，表示您无法访问环境。

在 Amazon MWAA 中使用启动脚本

启动脚本是您托管在环境的 Amazon S3 存储桶中的 Shell (.sh) 脚本，类似于 DAG、要求和插件。在安装要求和初始化 Apache Airflow 流程之前，Amazon MWAA 会在每个单独的 Apache Airflow 组件（工作线程、计划程序和 Web 服务器）上启动时运行此脚本。使用启动脚本执行以下操作：

- 安装运行时 - 安装工作流程和连接所需的 Linux 运行时。
- 配置环境变量 - 为每个 Apache Airflow 组件设置环境变量。覆盖常用变量，例如 PATH、PYTHONPATH 和 LD_LIBRARY_PATH。
- 管理密钥和令牌 - 将自定义存储库的访问令牌传递给 requirements.txt 并配置安全密钥。

以下主题介绍如何配置启动脚本以安装 Linux 运行时、设置环境变量以及使用 CloudWatch Logs 排查相关问题。

主题

- 配置启动脚本
- 使用启动脚本安装 Linux 运行时
- 使用启动脚本设置环境变量

配置启动脚本

要在现有 Amazon MWAA 环境中使用启动脚本，请将 `.sh` 文件上传到环境的 Amazon S3 存储桶。然后，要将脚本与环境关联，请在环境详细信息中指定以下内容：

- 脚本的 Amazon S3 URL 路径 — 存储桶中托管的脚本的相对路径，例如 `s3://mwaa-environment/startup.sh`
- 脚本的 Amazon S3 版本 ID — Amazon S3 存储桶中启动 shell 脚本的版本。每次更新脚本时，您都必须指定 Amazon S3 为文件分配的版本 ID。版本 ID 是 Unicode、以 UTF-8 编码、URL 就绪、不透明的字符串，长度不超过 1,024 字节，例如 `3sL4kqtJlcpXroDTDmJ+rmSpXd3dIbrHY+MTRCxf3vjVBH40Nr8X8gdRQBpUMLUo`。

要完成本节中的步骤，请使用以下示例脚本。该脚本输出分配给 `MWAA_AIRFLOW_COMPONENT` 的值。此环境变量标识在其上运行脚本的每个 Apache Airflow 组件。

复制代码并将其本地另存为 `startup.sh`。

```
#!/bin/sh

echo "Printing Apache Airflow component"
echo $MWAA_AIRFLOW_COMPONENT
```

接下来，将脚本上传到 Amazon S3 存储桶。

AWS 管理控制台

要上传 shell 脚本（控制台），请执行以下操作

1. 登录到 AWS 管理控制台，然后通过以下网址打开 Amazon S3 控制台：https://console.aws.amazon.com/s3/。
2. 从存储桶列表中，选择与环境关联的存储桶的名称。
3. 在 Objects（对象）选项卡上，选择 Upload（上载）。
4. 在上传页面上，拖放您创建的 shell 脚本。
5. 选择上传。

该脚本在对象列表中列出。Amazon S3 将为文件创建新的版本 ID。如果您更新脚本并使用相同的文件名将其再次上传，则会为该文件分配一个新的版本 ID。

AWS CLI

要创建和上传 shell 脚本 (CLI)，请执行以下操作

1. 打开新的命令提示符，然后运行 Amazon S3 `ls` 命令以列出和识别与环境关联的存储桶。

```
aws s3 ls
```

2. 导航到保存 shell 脚本的文件夹。在新提示窗口中使用 cp 将脚本上传到存储桶。用您的信息替换 *amzn-s3-demo-bucket*。

```
aws s3 cp startup.sh s3://amzn-s3-demo-bucket/startup.sh
```

如果成功，Amazon S3 会输出该对象的 URL 路径：

```
upload: ./startup.sh to s3://amzn-s3-demo-bucket/startup.sh
```

3. 使用以下命令检索脚本的最新版本 ID。

```
aws s3api list-object-versions --bucket amzn-s3-demo-bucket --prefix startup --
query 'Versions[?IsLatest].[VersionId]' --output text
```

```
BbdVMmBRjtestta1EsVnbybZp1Wqh1J4
```

当您将脚本与环境关联时，可以指定此版本 ID。

现在，将脚本与环境相关联。

AWS 管理控制台

要将脚本与环境（控制台）关联，请执行以下操作

1. 在 Amazon MWAA 控制台上打开环境页面。
2. 选择要更新的环境所在的行，然后选择编辑。
3. 在指定详情页面上，在启动脚本文件（可选），输入脚本的 Amazon S3 URL，例如：s3://*amzn-s3-demo-bucket*/startup-sh.。
4. 从下拉列表中选择最新版本，或者浏览 S3 以查找脚本。

5. 选择下一步，继续进入查看页面。
6. 查看更改，然后选择保存。

环境更新可能需要 10 到 30 分钟。当环境中的每个组件重新启动时，Amazon MWAA 会运行启动脚本。

AWS CLI

要将脚本与环境 (CLI) 关联，请执行以下操作

- 打开命令提示符并使用 `update-environment` 为脚本指定 Amazon S3 URL 和版本 ID。

```
aws mwaa update-environment \
--name your-mwaa-environment \
--startup-script-s3-path startup.sh \
--startup-script-s3-object-version BbdVMmBRjtestta1EsVnbybZp1Wqh1J4
```

如果成功，Amazon MWAA 将返回环境的 Amazon 资源名称（ARN）：

```
arn:aws:airflow:us-west-2:123456789012:environment/your-mwaa-environment
```

环境更新可能需要 10 到 30 分钟。当环境中的每个组件重新启动时，Amazon MWAA 会运行启动脚本。

最后，检索日志事件以验证脚本是否按预期运行。当您为每个 Apache Airflow 组件激活日志记录时，Amazon MWAA 会创建新的日志组和日志流。有关更多信息，请参阅 Apache Airflow 日志类型。

AWS 管理控制台

获得 Apache Airflow 日志流（控制台）

1. 在 Amazon MWAA 控制台上打开环境页面。
2. 选择环境。
3. 在监控窗格中，选择要访问其日志的日志组，例如 Airflow 计划程序日志组。
4. 在 CloudWatch 控制台中，从日志流列表中选择一个带有前缀 `startup_script_execution_ip` 的流。

5. 在日志事件窗格上，您将看到用于打印 MWAA_AIRFLOW_COMPONENT 的值的命令输出。例如，对于计划程序日志，您将执行以下操作：

```
Printing Apache Airflow component
        scheduler
        Finished running startup script. Execution time: 0.004s.
        Running verification
        Verification completed
```

您可以重复前述步骤来访问工作线程和 Web 服务器日志。

使用启动脚本安装 Linux 运行时

使用启动脚本更新 Apache Airflow 组件的操作系统，并安装其他运行时库以与工作流程一起使用。例如，以下脚本运行 `yum update` 来更新操作系统。

在启动脚本中运行 `yum update` 时，必须使用 `--exclude=python*` 排除 Python，如示例所列。为了让环境运行，Amazon MWAA 会安装与环境兼容的特定版本的 Python。因此，您无法使用启动脚本更新环境的 Python 版本。

```
#!/bin/sh

echo "Updating operating system"
sudo yum update -y --exclude=python*
```

要在特定的 Apache Airflow 组件上安装运行时，请使用 `MWAA_AIRFLOW_COMPONENT` 和 `if` 和 `fi` 条件语句。此示例运行单个命令将 `libaio` 库安装在计划程序和工作线程上，但不安装在 Web 服务器上。

> **Important**
>
> - 如果您配置了私有 Web 服务器，则必须使用以下条件或在本地提供所有安装文件，以避免安装超时。
> - 使用 `sudo` 运行需要管理权限的操作。

```
#!/bin/sh
```

```
if [[ "${MWAA_AIRFLOW_COMPONENT}" != "webserver" ]]
  then
    sudo yum -y install libaio

fi
```

您可以使用启动脚本来获得 Python 版本。

```
#!/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 不支持覆盖默认 Python 版本，因为这可能会导致与已安装的 Apache Airflow 库不兼容。

使用启动脚本设置环境变量

使用启动脚本来设置环境变量和修改 Apache Airflow 配置。以下示例定义了新变量 `ENVIRONMENT_STAGE`。您可以在 DAG 或自定义模块中引用此变量。

```
#!/bin/sh

export ENVIRONMENT_STAGE="development"
echo "$ENVIRONMENT_STAGE"
```

使用启动脚本覆盖常见的 Apache Airflow 或系统变量。例如，您设置 `LD_LIBRARY_PATH` 来指示 Python 在您指定的路径中搜索二进制文件。这允许您使用插件为工作流程提供自定义二进制文件：

```
#!/bin/sh

export LD_LIBRARY_PATH=/usr/local/airflow/plugins/your-custom-binary
```

预留环境变量

Amazon MWAA 保留了一组关键的环境变量。如果您覆盖保留变量，Amazon MWAA 会将其恢复为默认值。以下列出了保留变量：

- MWAA__AIRFLOW__COMPONENT— 用于使用以下值之一标识 Apache Airflow 组件：scheduler、worker 或 webserver。
- AIRFLOW__WEBSERVER__SECRET_KEY— 用于在 Apache Airflow Web 服务器中安全签署会话 cookie 的密钥。
- AIRFLOW__CORE__FERNET_KEY— 用于加密和解密存储在元数据数据库中的敏感数据的密钥，例如连接密码。
- AIRFLOW_HOME— Apache Airflow 主目录的路径，配置文件和 DAG 文件存储在本地。
- AIRFLOW__CELERY__BROKER_URL— 用于 Apache Airflow 计划程序和 Celery Worker 节点之间通信的消息代理的 URL。
- AIRFLOW__CELERY__RESULT_BACKEND— 用于存储 Celery 任务结果的数据库的 URL。
- AIRFLOW__CORE__EXECUTOR— Apache Airflow 必须使用的执行程序类。在 Amazon MWAA 中，这是 CeleryExecutor
- AIRFLOW__CORE__LOAD_EXAMPLES— 用于激活或停用示例 DAG 的加载。
- AIRFLOW__METRICS__METRICS_BLOCK_LIST— 用于管理 Amazon MWAA 在 CloudWatch 中发出和捕获的 Apache Airflow 指标。
- SQL_ALCHEMY_CONN— 用于在 Amazon MWAA 中存储 Apache Airflow 元数据的 RDS for PostgreSQL 数据库的连接字符串。
- AIRFLOW__CORE__SQL_ALCHEMY_CONN— 用于与 SQL_ALCHEMY_CONN 相同的目的，但遵循新的 Apache Airflow 命名约定。
- AIRFLOW__CELERY__DEFAULT_QUEUE— Apache Airflow 中 Celery 任务的默认队列。
- AIRFLOW__OPERATORS__DEFAULT_QUEUE— 使用特定 Apache Airflow 运算符执行任务的默认队列。
- AIRFLOW_VERSION— 安装在 Amazon MWAA 环境中的 Apache Airflow 版本。
- AIRFLOW_CONN_AWS_DEFAULT— 用于与其他 AWS 服务集成的默认 AWS 凭证。
- AWS_DEFAULT_REGION— 设置默认 AWS 区域，与默认凭证一起用于与其他 AWS 服务集成。
- AWS_REGION — 如果已定义此环境变量，它将覆盖环境变量 AWS_DEFAULT_REGION 和配置文件设置区域中的值。
- PYTHONUNBUFFERED— 用于向容器日志发送 stdout 和 stderr 流。
- AIRFLOW__METRICS__STATSD_ALLOW_LIST— 用于配置以逗号分隔前缀的允许列表，以发送以列表元素开头的指标。
- AIRFLOW__METRICS__STATSD_ON— 激活向 StatsD 发送指标。
- AIRFLOW__METRICS__STATSD_HOST— 用于连接到 StatSD 进程守护程序。

- AIRFLOW__METRICS__STATSD_PORT— 用于连接到 StatSD 进程守护程序。
- AIRFLOW__METRICS__STATSD_PREFIX— 用于连接到 StatSD 进程守护程序。
- AIRFLOW__CELERY__WORKER_AUTOSCALE— 设置最大和最小并发度。
- AIRFLOW__CORE__DAG_CONCURRENCY— 设置计划程序可以在一个 DAG 中并行运行的任务实例数。
- AIRFLOW__CORE__MAX_ACTIVE_TASKS_PER_DAG— 设置每个 DAG 的最大活动任务数。
- AIRFLOW__CORE__PARALLELISM— 定义可以同时运行的最大任务实例数。
- AIRFLOW__SCHEDULER__PARSING_PROCESSES— 设置计划程序为调度 DAG 而解析的最大进程数。
- AIRFLOW__CELERY_BROKER_TRANSPORT_OPTIONS__VISIBILITY_TIMEOUT— 定义在将消息重新传送给其他工作线程之前，工作线程等待确认任务的秒数。
- AIRFLOW__CELERY_BROKER_TRANSPORT_OPTIONS__REGION— 为底层 Celery 传输设置 AWS 区域。
- AIRFLOW__CELERY_BROKER_TRANSPORT_OPTIONS__PREDEFINED_QUEUES— 为底层 Celery 传输设置队列。
- AIRFLOW_SCHEDULER_ALLOWED_RUN_ID_PATTERN— 用于在触发 DAG 时验证 run_id 参数输入的有效性。
- AIRFLOW__WEBSERVER__BASE_URL— 用于托管 Apache Airflow UI 的 Web 服务器的 URL。
- PYTHONPATH（仅适用于 Apache Airflow v2.9 及更高版本）— 由 Amazon MWAA 保留，以确保所有基本环境功能都能正常运行。

> **Note**
>
> 对于 v2.9 之前的 Apache Airflow 版本，PYTHONPATH 是一个未保留的环境变量。

非预留环境变量

您可以使用启动脚本来覆盖未保留的环境变量。以下列表提供了一些常见变量：

- PATH— 指定操作系统在其中搜索可执行文件和脚本的目录列表。在命令行中运行命令时，系统会检查 PATH 中的目录以查找并执行该命令。在 Apache Airflow 中创建自定义运算符或任务时，可能需要依赖外部脚本或可执行文件。如果包含这些文件的目录不在 PATH 变量中指定的目录中，则当系统无法找到它们时，任务将无法运行。通过将相应的目录添加到 PATH，Apache Airflow 任务可以找到并运行所需的可执行文件。

- PYTHONPATH— Python 解释器使用它来确定要在哪些目录中搜索导入的模块和程序包。它是您可以添加到默认搜索路径的目录列表。这使解释器可以查找和加载未包含在标准库中或未安装在系统目录中的 Python 库。使用此变量添加模块和自定义 Python 程序包，并将其与 DAG 一起使用。

  > **Note**
  >
  > 对于 Apache Airflow v2.9 及更高版本，PYTHONPATH 是一个保留的环境变量。

- LD_LIBRARY_PATH— Linux 中的动态链接器和加载器使用的环境变量，用于查找和加载共享库。它指定了包含共享库的目录列表，这些共享库在默认系统库目录之前接受搜索。使用此变量来指定自定义二进制文件。
- CLASSPATH— 由 Java 运行时环境（JRE）和 Java 开发套件（JDK）用于在运行时查找和加载 Java 类、库和资源。它是一个包含已编译的 Java 代码的目录、JAR 文件和 ZIP 档案的列表。

在 Amazon MWAA 上使用 DAG

要在 Amazon MWAA 环境中运行有向无环图（DAG），请将文件复制到与环境相连的 Amazon S3 存储桶中，然后让 Amazon MWAA 知道您的 DAG 和支持文件在 Amazon MWAA 控制台上的位置。Amazon MWAA 负责在工作线程、计划程序和 Web 服务器之间同步 DAG。本指南介绍如何在 Amazon MWAA 环境中添加或更新 DAG，以及如何安装自定义插件和 Python 依赖项。

主题

Amazon S3 存储桶概述

适用于 Amazon MWAA 环境的 Amazon S3 存储桶必须已阻止公共访问权限。默认情况下，所有 Amazon S3 资源都是私有的，包括桶、对象和相关子资源（例如，生命周期配置）。

- 只有资源拥有者，即创建该存储桶的 AWS 账户 可以访问该资源。资源拥有者（例如管理员）可以写入访问控制策略来授予他人访问权限。
- 您设置的访问策略必须有权将 DAG、`plugins.zip` 中的自定义插件和 `requirements.txt` 中的 Python 依赖项添加到 Amazon S3 存储桶中。有关包含所需权限的策略示例，请参阅 AmazonMWAAFullConsoleAccess。

Amazon MWAA 环境的 Amazon S3 存储桶必须启用版本控制。启用 Amazon S3 存储桶版本控制后，每当创建新版本时，都会创建一个新副本。

- 在 Amazon S3 存储桶上，为 `plugins.zip` 中的自定义插件和 `requirements.txt` 中的 Python 依赖项启用了版本控制。
- 每次在 Amazon S3 存储桶上更新文件时，都必须在 Amazon MWAA 控制台上指定 `plugins.zip` 和 `requirements.txt` 的版本。

添加或更新 DAG

有向无环图（DAG）在 Python 文件中定义，该文件将 DAG 的结构定义为代码。您可以使用 AWS CLI 或 Amazon S3 控制台将 DAG 上传到环境。本主题介绍使用 Amazon S3 存储桶中的 dags 文件夹，在 Amazon MWAA 环境中添加或更新 Apache Airflow DAG 的步骤。

各个部分

先决条件

在完成本页上的步骤之前，您需要具备以下条件。

- 权限 — 您的 AWS 账户 必须已获得管理员授权，访问适用于环境的 AmazonMWAAFullConsoleAccess 访问控制策略。此外，执行角色必须允许 Amazon MWAA 环境访问环境所使用的 AWS 资源。
- 访问权限 — 如果您需要访问公共存储库以便直接在 Web 服务器上安装依赖项，则必须将环境配置为具有公共网络 Web 服务器访问权限。有关更多信息，请参阅the section called “Apache Airflow 访问模式”。
- Amazon S3 配置 — 用于存储 DAG 的 Amazon S3 存储桶、在 `plugins.zip` 中的自定义插件和在 `requirements.txt` 中的 Python 依赖项必须配置为已阻止公共访问和已启用版本控制。

工作方式

有向无环图（DAG）在单个 Python 文件中定义，该文件将 DAG 的结构定义为代码。它包含以下各项：

- DAG 定义。

- 描述如何运行 DAG 和要运行的任务的运算符。
- 描述任务运行顺序的运算符关系。

要在 Amazon MWAA 环境中运行 Apache Airflow 平台，您需要将 DAG 定义复制到存储桶中的 dags 文件夹。例如，存储桶中的 DAG 文件夹应如下所示：

Example DAG 文件夹

```
dags/
# dag_def.py
```

Amazon MWAA 每 30 秒自动将新建和更改的对象从 Amazon S3 存储桶同步到 Amazon MWAA 计划程序和工作线程容器的 /usr/local/airflow/dags 文件夹，从而保留 Amazon S3 源的文件层次结构，无论文件类型如何。新 DAG 在 Apache Airflow UI 中列出所需的时间由 scheduler.dag_dir_list_interval 控制。对现有 DAG 的更改将在下一个 DAG 处理循环中获取。

> **Note**
>
> 您不需要在 DAG 文件夹中包含 airflow.cfg 配置文件。您可以从Amazon MWAA 控制台覆盖默认 Apache Airflow 配置。有关更多信息，请参阅在 Amazon MWAA 上使用 Apache Airflow 配置选项。

更改了哪些内容？

要查看特定 Apache Airflow 版本的更改，请参阅发布说明页面。

- Apache Airflow v3 配置：配置参考
- Apache Airflow v2 公共接口信息：Airflow 的公共接口

使用 Amazon MWAA CLI 实用工具测试 DAG

- 命令行界面 (CLI) 实用工具可在本地复制 Amazon MWAA 环境。
- CLI 在本地构建 Docker 容器镜像，类似于 Amazon MWAA 生产镜像。您可以使用它运行本地 Apache Airflow 环境来开发和测试 DAG、自定义插件和依赖项，然后部署到 Amazon MWAA。
- 要运行 CLI，请参阅 GitHub 上的 aws-mwaa-docker-images。

将 DAG 代码上传到 Amazon S3

您可以使用 Amazon S3 控制台或 AWS Command Line Interface (AWS CLI) 将 DAG 代码上传到 Amazon S3 存储桶中。以下步骤假设您正在将代码 (.py) 上传到 Amazon S3 存储桶中名为 dags 的文件夹。

使用 AWS CLI

AWS Command Line Interface (AWS CLI) 是一种开源工具，您可以用来在命令行 Shell 中使用命令与 AWS 服务进行交互。要完成本节中的步骤，您需要以下满足以下条件：

- AWS CLI – 安装版本 2。
- AWS CLI – 使用 aws configure 进行快速配置。

要使用 AWS CLI 上传，请执行以下操作

1. 以下示例列出所有 Amazon S3 存储桶。

```
aws s3 ls
```

2. 使用以下命令列出 Amazon S3 存储桶中适合环境的文件和文件夹。

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

3. 以下命令将 dag_def.py 文件上传到 dags 文件夹。

```
aws s3 cp dag_def.py s3://amzn-s3-demo-bucket/dags/
```

如果 Amazon S3 存储桶中尚不存在名为 dags 的文件夹，则此命令会创建 dags 文件夹，并将名为 dag_def.py 的文件上传到新文件夹。

使用 Amazon S3 控制台

Amazon S3 控制台是一个基于 Web 的用户界面，可用来创建和管理 Amazon S3 存储桶中的资源。以下步骤假设您有一个名为 dags 的 DAG 文件夹。

要使用 Amazon S3 控制台上传，请执行以下操作

1. 在 Amazon MWAA 控制台上打开环境页面。

2. 选择环境。
3. 在 S3 中的 DAG 代码窗格中选择 S3 存储桶链接，在控制台上打开存储桶。
4. 选择 dags 文件夹。
5. 选择上传。
6. 选择 添加文件。
7. 选择 dag_def.py 的本地副本，选择上传。

在 Amazon MWAA 控制台上指定 DAG 文件夹的路径（第一次）

以下步骤假设您要在 Amazon S3 桶中指定名为 dags 文件夹的路径。

1. 在 Amazon MWAA 控制台上打开环境页面。
2. 选择要在其中运行 DAG 的环境。
3. 选择编辑。
4. 在 Amazon S3 中的 DAG 代码窗格上，选择 DAG 文件夹字段旁边的浏览 S3。
5. 选择 dags 文件夹。
6. 选择选择。
7. 选择下一步、更新环境。

在 Apache Airflow UI 上访问更改

您需要在 AWS Identity and Access Management (IAM) 中拥有 AWS 账户 的 Apache Airflow 用户界面访问策略： AmazonMWAAWebServerAccess 权限才能访问 Apache Airflow UI。

要访问 Apache Airflow UI，请执行以下操作

1. 在 Amazon MWAA 控制台上打开环境页面。
2. 选择环境。
3. 选择打开 Airflow UI。

接下来做什么？

使用 GitHub 上的 aws-mwaa-docker-images 在本地测试 DAG、自定义插件和 Python 依赖项。

安装自定义插件

Amazon MWAA 支持 Apache Airflow 的内置插件管理器，允许您使用自定义 Apache Airflow 运算符、挂钩、传感器或接口。本页介绍使用 `plugins.zip` 文件在 Amazon MWAA 环境中安装 Apache Airflow 自定义插件的步骤。

目录

先决条件

在完成本页上的步骤之前，您需要具备以下条件。

- 权限 — 您的 AWS 账户 必须已获得管理员授权，访问适用于环境的 AmazonMWAAFullConsoleAccess 访问控制策略。此外，执行角色必须允许 Amazon MWAA 环境访问环境所使用的 AWS 资源。
- 访问权限 — 如果您需要访问公共存储库以便直接在 Web 服务器上安装依赖项，则必须将环境配置为具有公共网络 Web 服务器访问权限。有关更多信息，请参阅the section called “Apache Airflow 访问模式”。
- Amazon S3 配置 — 用于存储 DAG 的 Amazon S3 存储桶、在 `plugins.zip` 中的自定义插件和在 `requirements.txt` 中的 Python 依赖项必须配置为已阻止公共访问和已启用版本控制。

工作方式

要在环境中运行自定义插件，您必须做三件事：

1. 在本地创建 `plugins.zip` 文件。
2. 将 `plugins.zip` 文件上传到 Amazon S3 中的存储桶。
3. 在 Amazon MWAA 控制台的插件文件字段中指定此文件的版本。

> **Note**
>
> 如果这是您首次将 `plugins.zip` 上传到 Amazon S3 存储桶，则还需要在 Amazon MWAA 控制台上指定文件路径。您只需要完成此步骤一次。

何时使用插件

正如 Apache Airflow 文档中所述，仅在扩展 Apache Airflow 用户界面时才需要插件。可以直接将自定义操作符与 DAG 代码一起放入 `/dags` 文件夹中。

如果需要自行创建与外部系统的集成，请将其放在 `/dags` 文件夹或其中的子文件夹中，而不是放在 `plugins.zip` 文件夹中。在 Apache Airflow 2.x 中，插件主要用于扩展 UI。

同样，不能将其他依赖项放入 `plugins.zip` 中。而可以将其存储在 Amazon S3 `/dags` 文件夹中的某个位置，在 Apache Airflow 启动之前，这些依赖项将在该位置同步到每个 Amazon MWAA 容器。

> **Note**
>
> /dags 文件夹或 plugins.zip 中任何未显式定义 Apache Airflow DAG 对象的文件都必须在 .airflowignore 文件中列出。

自定义插件概述

Apache Airflow 的内置插件管理器只需将文件拖放到 $AIRFLOW_HOME/plugins 文件夹中即可将外部功能集成到其核心中。它允许您使用自定义 Apache Airflow 运算符、钩子、传感器或接口。下一节提供了本地开发环境中平面和嵌套目录结构的示例，以及生成的 import 语句，这些语句决定了 plugins.zip 中的目录结构。

自定义插件目录和大小限制

在启动期间，Apache Airflow 计划程序和工作线程在 /usr/local/airflow/plugins/* 中环境 AWS 托管的 Fargate 容器中搜索自定义插件。

- 目录结构。目录结构（在 /* 中）基于您 plugins.zip 文件的内容。例如，如果 plugins.zip 包含 operators 目录作为主级目录，则该目录将被解压缩到环境的 /usr/local/airflow/plugins/*operators* 中。
- 大小限制。我们建议使用小于 1 GB 的 plugins.zip 文件。plugins.zip 文件大小越大，环境的启动时间就越长。尽管 Amazon MWAA 没有明确限制 plugins.zip 文件的大小，但如果无法在十分钟内安装依赖项，Fargate 服务将超时并尝试将环境回滚到稳定状态。

> **Note**
>
> 对于使用 Apache Airflow v2.0.2 的环境，Amazon MWAA 会限制 Apache Airflow Web 服务器上的出站流量，并且不允许直接在 Web 服务器上安装插件或 Python 依赖项。从 Apache Airflow v2.2.2 开始，Amazon MWAA 可以直接在 Web 服务器上安装插件和依赖项。

自定义插件示例

下一节使用《Apache Airflow 参考指南》中的示例代码来说明如何构建本地开发环境。

在 plugins.zip 中使用平面目录结构的示例

Apache Airflow v3

以下示例显示了 Apache Airflow v3 中一个采用扁平目录结构的 `plugins.zip` 文件。

Example带 PythonVirtualenvOperator plugins.zip 的扁平目录

以下示例显示 为 Apache Airflow PythonVirtualenvOperator 创建自定义插件 中 `PythonVirtualenvOperator` 自定义插件的 plugins.zip 文件的主级树。

```
### virtual_python_plugin.py
```

Example plugins/virtual_python_plugin.py

以下示例显示 `PythonVirtualenvOperator` 自定义插件。

```
"""
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 v2

以下示例显示了 Apache Airflow v2 中一个采用扁平目录结构的 `plugins.zip` 文件。

Example带 PythonVirtualenvOperator plugins.zip 的扁平目录

以下示例显示 为 Apache Airflow PythonVirtualenvOperator 创建自定义插件 中 `PythonVirtualenvOperator` 自定义插件的 plugins.zip 文件的主级树。

```
### virtual_python_plugin.py
```

Example plugins/virtual_python_plugin.py

以下示例显示 `PythonVirtualenvOperator` 自定义插件。

```
"""
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'
```

在 plugins.zip 中使用平面目录结构的示例

Apache Airflow v3

以下示例显示一个 `plugins.zip` 文件，其中包含 `hooks`、`operators` 的单独目录和 `sensors` 目录。

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
```

以下示例显示使用自定义插件的 DAG（DAG 文件夹）中的导入语句。

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]
```

以下示例显示自定义插件文件中所需的每条导入语句。

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
```

按照使用 Amazon MWAA CLI 实用工具测试自定义插件中的步骤进行操作，然后创建 plugins.zip 文件来将内容压缩到 `plugins` 目录之内。例如 `cd plugins`。

Apache Airflow v2

以下示例显示一个 plugins.zip 文件，其中包含 hooks、operators 的单独目录和 sensors 目录。

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
```

以下示例显示使用自定义插件的 DAG（DAG 文件夹）中的导入语句。

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]
```

以下示例显示自定义插件文件中所需的每条导入语句。

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
```

按照使用 Amazon MWAA CLI 实用工具测试自定义插件中的步骤进行操作，然后创建 plugins.zip 文件来将内容压缩到 `plugins` 目录之内。例如 `cd plugins`。

创建 plugins.zip 文件

以下步骤描述了我们建议在本地创建 plugins.zip 文件的步骤。

步骤 1：使用 Amazon MWAA CLI 实用工具测试自定义插件

- 命令行界面 (CLI) 实用工具可在本地复制 Amazon MWAA 环境。
- CLI 在本地构建 Docker 容器镜像，类似于 Amazon MWAA 生产镜像。您可以使用它运行本地 Apache Airflow 环境来开发和测试 DAG、自定义插件和依赖项，然后部署到 Amazon MWAA。
- 要运行 CLI，请参阅 GitHub 上的 aws-mwaa-docker-images。

步骤 2：创建 plugins.zip 文件

您可以使用内置的 ZIP 存档实用工具或任何其他 ZIP 实用工具（例如 7zip）来创建.zip 文件。

> **Note**
>
> 当您创建 .zip 文件时，Windows 操作系统的内置 ZIP 实用工具可能会添加子文件夹。我们建议您验证 plugins.zip 文件的内容，然后再上传到 Amazon S3 存储桶，以确保没有添加其他目录。

1. 将目录更改为本地 Airflow 插件目录。例如：

```
myproject$ cd plugins
```

2. 运行以下命令以确保内容具有可执行权限（仅限 macOS 和 Linux）。

```
plugins$ chmod -R 755 .
```

3. 将内容压缩到 plugins 文件夹中。

```
plugins$ zip -r plugins.zip .
```

上传 **plugins.zip** 到 Amazon S3

您可以使用 Amazon S3 控制台或 AWS Command Line Interface (AWS CLI) 将 plugins.zip 文件上传到 Amazon S3 存储桶中。

使用 AWS CLI

AWS Command Line Interface (AWS CLI) 是一种开源工具，您可以用来在命令行 Shell 中使用命令与 AWS 服务进行交互。要完成本节中的步骤，您需要以下满足以下条件：

- AWS CLI – 安装版本 2。
- AWS CLI – 使用 aws configure 进行快速配置。

要使用 AWS CLI 上传，请执行以下操作

1. 在命令提示符下，导航到存储 plugins.zip 文件的目录。例如：

```
cd plugins
```

2. 以下示例列出所有 Amazon S3 存储桶。

```
aws s3 ls
```

3. 使用以下命令列出 Amazon S3 存储桶中适合环境的文件和文件夹。

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

4. 使用以下命令将 plugins.zip 文件上传到环境的 Amazon S3 存储桶。

```
aws s3 cp plugins.zip s3://amzn-s3-demo-bucket/plugins.zip
```

使用 Amazon S3 控制台

Amazon S3 控制台是一个基于 Web 的用户界面，可用来创建和管理 Amazon S3 存储桶中的资源。

要使用 Amazon S3 控制台上传，请执行以下操作

1. 在 Amazon MWAA 控制台上打开环境页面。
2. 选择环境。
3. 在 S3 中的 DAG 代码窗格中选择 S3 存储桶链接，在控制台上打开存储桶。
4. 选择上传。
5. 选择 添加文件。
6. 选择 plugins.zip 的本地副本，选择上传。

在环境中安装自定义插件

本节介绍如何安装您上传到 Amazon S3 存储桶的自定义插件，方法是指定 plugins.zip 文件的路径，并在每次更新 zip 文件时指定 plugins.zip 文件的版本。

在 Amazon MWAA 控制台上指定 **plugins.zip** 的路径（第一次）

如果这是您首次将 plugins.zip 上传到 Amazon S3 存储桶，则还需要在 Amazon MWAA 控制台上指定文件路径。您只需要完成此步骤一次。

1. 在 Amazon MWAA 控制台上打开环境页面。
2. 选择环境。
3. 选择编辑。
4. 在 Amazon S3 中的 DAG 代码窗格上，选择要求文件 - 可选字段旁边的浏览 S3。
5. 选择 Amazon S3 存储桶中的 plugins.zip 文件。
6. 选择选择。
7. 选择下一步、更新环境。

在 Amazon MWAA 控制台上指定 **plugins.zip** 的版本

每次在 Amazon S3 存储桶中上传 plugins.zip 的新版本时，都需要在 Amazon MWAA 控制台上指定 plugins.zip 文件的版本。

1. 在 Amazon MWAA 控制台上打开环境页面。
2. 选择环境。
3. 选择编辑。
4. 在 Amazon S3 中的 DAG 代码窗格中，从下拉列表中选择 plugins.zip 的版本。
5. 选择下一步。

plugins.zip 的用例示例

- 要了解如何创建自定义插件，请参阅 使用 Apache Hive 和 Hadoop 的自定义插件。
- 要了解如何创建自定义插件，请参阅 修补 PythonVirtualenvOperator 的自定义插件 。
- 要了解如何创建自定义插件，请参阅 使用 Oracle 定制插件。
- 要了解如何创建自定义插件，请参阅 the section called “更改 DAG 的时区”。

接下来做什么？

使用 GitHub 上的 aws-mwaa-docker-images 在本地测试 DAG、自定义插件和 Python 依赖项。

安装 Python 依赖项

Python 依赖项是指在 Amazon MWAA 环境中的 Apache Airflow 版本的 Apache Airflow 基础版安装中未包含的任何程序包或发行版。本主题介绍使用 Amazon S3 存储桶中的 requirements.txt 文件在 Amazon MWAA 环境中安装 Apache Airflow Python 依赖项的步骤。

目录

先决条件

在完成本页上的步骤之前，您需要具备以下条件。

- 权限 — 您的管理员 AWS 账户 必须已授予您访问您环境的 Amazon MWAAFull ConsoleAccess 访问控制策略的权限。此外，您的执行角色必须允许您的 Amazon MWAA 环境访问您的环境所使用的 AWS 资源。
- 访问权限 — 如果您需要访问公共存储库以便直接在 Web 服务器上安装依赖项，则必须将环境配置为具有公共网络 Web 服务器访问权限。有关更多信息，请参阅the section called “Apache Airflow 访问模式”。
- Amazon S3 配置 — 用于存储您的 DAGs自定义插件和 Python 依赖项的 Amazon S3 存储桶`requirements.txt`必须配置为已阻止公共访问和启用版本控制。`plugins.zip`

工作原理

在 Amazon MWAA 上，您可以安装所有 Python 依赖项，方法是将 `requirements.txt` 文件上传到 Amazon S3 存储桶，然后在每次更新文件时在 Amazon MWAA 控制台上指定该文件的版本。Amazon MWAA 运行 `pip3 install -r requirements.txt`，以在 Apache Airflow 计划程序和每个工作线程上安装 Python 依赖项。

要在环境中运行 Python 依赖项，您必须做三件事：

1. 在本地创建 `requirements.txt` 文件。
2. 将本地 `requirements.txt` 上传到 Amazon S3 中的存储桶。

3. 在 Amazon MWAA 控制台上的要求文件字段中指定此文件的版本。

> **Note**
>
> 如果这是您首次创建 `requirements.txt` 并将其上传到 Amazon S3 存储桶，则还需要在 Amazon MWAA 控制台上指定文件路径。您只需要完成此步骤一次。

Python 依赖项概述

你可以从 Python 包索引 PyPi (.org)、Python wheels () 或 Python whe `.whl` els () 或环境中兼容私有的 /Pep-503 存储库上 PyPi托管的 Python 依赖项中安装 Apache Airflow 额外内容和其他 Python 依赖项。

Python 依赖项位置和大小限制

Apache Airflow 计划程序和工作线程会在 `requirements.txt` 文件中搜索软件包，然后在环境中将该软件包安装到 `/usr/local/airflow/.local/bin` 位置。

- 大小限制。我们建议使用 `requirements.txt` 文件，以引用组合大小小于 1 GB 的库。Amazon MWAA 需要安装的库越多，环境上的启动时间就越长。尽管 Amazon MWAA 没有明确限制安装的库的大小，但如果无法在十分钟内安装依赖项，Fargate 服务将超时并尝试将环境回滚到稳定状态。

创建 requirements.txt 文件

以下步骤描述了在本地创建 plugins.zip 文件时我们建议的步骤。

步骤 1：使用 Amazon MWAA CLI 实用工具测试 Python 依赖项

- 命令行界面 (CLI) 实用工具可在本地复制 Amazon MWAA 环境。
- CLI 在本地构建 Docker 容器镜像，类似于 Amazon MWAA 生产镜像。在部署到 Amazon MWAA 之前，您可以使用它来运行本地 Apache Airflow 环境来开发和 DAGs测试自定义插件和依赖项。
- 要运行 CLI，请参阅aws-mwaa-docker-images上的 GitHub。

步骤 2：创建 **`requirements.txt`**

下一节介绍如何在 `requirements.txt` 文件中指定 Python 程序包索引中的 Python 依赖项。

Apache Airflow v3

1. 本地测试。在创建 `requirements.txt` 文件之前，以迭代方式添加其他库以找到程序包及其版本的正确组合。要运行 Amazon MWAA CLI 实用程序，请参阅上的。aws-mwaa-docker-images GitHub
2. 查看 Apache Airflow 程序包的 Extras。要访问亚马逊 MWAA 上为 Apache Airflow v3 安装的软件包列表，请参阅网站。aws-mwaa-docker-images `requirements.txt` GitHub
3. 添加约束语句。在文件顶部添加 Apache Airflow v3 环境的约束文件。`requirements.txt`Apache Airflow 约束文件指定了 Apache Airflow 发布时可用的提供程序版本。

   在以下示例中，用环境的版本号替换 *`{environment-version}`* 以及用与环境兼容的 Python 版本替换 *`{Python-version}`*。

   有关与 Apache Airflow 环境兼容的 Python 版本的信息，请参阅 Apache Airflow 版本。

   ```
   --constraint "https://raw.githubusercontent.com/apache/airflow/
   constraints-{Airflow-version}/constraints-{Python-version}.txt"
   ```

   如果约束文件确定该 `xyz==1.0` 程序包与环境中的其他程序包不兼容，则 `pip3 install` 将无法阻止不兼容的库安装到环境中。如果任何软件包的安装失败，则可以在日志的相应日志流中访问每个 Apache Airflow 组件（调度程序、工作程序和 Web 服务器）的错误日志。CloudWatch 有关日志类型的更多信息，请参阅 the section called “访问 Airflow 日志”。
4. Apache Airflow 程序包。添加程序包 Extras及其版本 (==)。这有助于防止在环境中安装同名但版本不同的程序包。

   ```
   apache-airflow[package-extra]==2.5.1
   ```
5. Python 库。在 `requirements.txt` 文件中添加程序包名称和版本 (==)。这有助于防止自动应用来自 PyPi.org 的 future 更新。

   ```
   library == version
   ```

   Example boto3 和 psycopg2-binary

   此示例仅用于演示目的。boto 和 psycopg2-binary 库包含在 Apache Airflow v3 基础版安装中，无需在 `requirements.txt` 文件中指定。

```
boto3==1.17.54
boto==2.49.0
botocore==1.20.54
psycopg2-binary==2.8.6
```

如果指定的包没有版本，则 Amazon MWAA 会安装.org 中最新版本的PyPi软件包。此版本可能与您 `requirements.txt` 中的其他程序包冲突。

Apache Airflow v2

1. 本地测试。在创建 `requirements.txt` 文件之前，以迭代方式添加其他库以找到程序包及其版本的正确组合。要运行 Amazon MWAA CLI 实用程序，请参阅上的。aws-mwaa-docker-images GitHub
2. 查看 Apache Airflow 程序包的 Extras。要访问亚马逊 MWAA 上为 Apache Airflow v2 安装的软件包列表，请访问网站。aws-mwaa-docker-images `requirements.txt` GitHub
3. 添加约束语句。在 `requirements.txt` 文件顶部添加 Apache Airflow v2 环境的约束文件。Apache Airflow 约束文件指定了 Apache Airflow 发布时可用的提供程序版本。

   从 Apache Airflow v2.7.2 开始，要求文件必须包含一条 `--constraint` 语句。如果您未提供约束条件，Amazon MWAA 将为您指定一个约束条件，以确保您的要求中列出的程序包与您正在使用的 Apache Airflow 版本兼容。

   在以下示例中，用环境的版本号替换 *{environment-version}* 以及用与环境兼容的 Python 版本替换 *{Python-version}*。

   有关与 Apache Airflow 环境兼容的 Python 版本的信息，请参阅 Apache Airflow 版本。

   ```
   --constraint "https://raw.githubusercontent.com/apache/airflow/
   constraints-{Airflow-version}/constraints-{Python-version}.txt"
   ```

   如果约束文件确定该 `xyz==1.0` 程序包与环境中的其他程序包不兼容，则 `pip3 install` 将无法阻止不兼容的库安装到环境中。如果任何软件包的安装失败，则可以在日志的相应日志流中访问每个 Apache Airflow 组件（调度程序、工作程序和 Web 服务器）的错误日志。CloudWatch 有关日志类型的更多信息，请参阅 the section called “访问 Airflow 日志”。
4. Apache Airflow 程序包。添加程序包 Extras及其版本 (==)。这有助于防止在环境中安装同名但版本不同的程序包。

```
apache-airflow[package-extra]==2.5.1
```

5. Python 库。在 `requirements.txt` 文件中添加程序包名称和版本 (==)。这有助于防止自动应用来自 PyPi.org 的 future 更新。

```
library == version
```

Example boto3 和 psycopg2-binary

此示例仅用于演示目的。boto 和 psycopg2-binary 库包含在 Apache Airflow v2 基础版安装中，无需在 `requirements.txt` 文件中指定。

```
boto3==1.17.54
boto==2.49.0
botocore==1.20.54
psycopg2-binary==2.8.6
```

如果指定的包没有版本，则 Amazon MWAA 会安装.org 中最新版本的PyPi软件包。此版本可能与您 `requirements.txt` 中的其他程序包冲突。

上传 `requirements.txt` 到 Amazon S3

您可以使用 Amazon S3 控制台或 AWS Command Line Interface (AWS CLI) 将`requirements.txt`文件上传到您的 Amazon S3 存储桶。

使用 AWS CLI

AWS Command Line Interface (AWS CLI) 是一个开源工具，您可以使用命令行 shell 中的命令与 AWS 服务进行交互。要完成本节中的步骤，您需要以下满足以下条件：

- AWS CLI — 安装版本 2。
- AWS CLI — 使用快速配置`aws configure`。

要使用上传 AWS CLI

1. 以下示例列出所有 Amazon S3 存储桶。

```
aws s3 ls
```

2. 使用以下命令列出 Amazon S3 存储桶中适合环境的文件和文件夹。

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

3. 以下命令将 `requirements.txt` 文件上传到 Amazon S3 存储桶。

```
aws s3 cp requirements.txt s3://amzn-s3-demo-bucket/requirements.txt
```

使用 Amazon S3 控制台

Amazon S3 控制台是一个基于 Web 的用户界面，可用来创建和管理 Amazon S3 存储桶中的资源。

要使用 Amazon S3 控制台上传，请执行以下操作

1. 在 Amazon MWAA 控制台上打开环境页面。
2. 选择环境。
3. 在 S3 中的 DAG 代码窗格中选择 S3 存储桶链接，在控制台上打开存储桶。
4. 选择上传。
5. 选择 添加文件。
6. 选择 `requirements.txt` 的本地副本，选择上传。

在环境中安装 Python 依赖项

本节介绍如何安装您上传到 Amazon S3 存储桶的依赖项，方法是指定 requirements.txt 文件的路径，并在每次更新时指定 requirements.txt 文件的版本。

在 Amazon MWAA 控制台上指定 **`requirements.txt`** 的路径（第一次）

如果这是您首次创建 `requirements.txt` 并将其上传到 Amazon S3 存储桶，则还需要在 Amazon MWAA 控制台上指定文件路径。您只需要完成此步骤一次。

1. 在 Amazon MWAA 控制台上打开环境页面。
2. 选择环境。

3. 选择编辑。
4. 在 Amazon S3 中的 DAG 代码窗格上，选择要求文件 - 可选字段旁边的浏览 S3。
5. 选择 Amazon S3 存储桶中的 `requirements.txt` 文件。
6. 选择选择。
7. 选择下一步、更新环境。

您可以在环境完成更新后立即开始使用新程序包。

在 Amazon MWAA 控制台上指定 **`requirements.txt`** 的版本

每次在 Amazon S3 存储桶中上传 `requirements.txt` 的新版本时，都需要在 Amazon MWAA 控制台上指定 `requirements.txt` 文件的版本。

1. 在 Amazon MWAA 控制台上打开环境页面。
2. 选择环境。
3. 选择编辑。
4. 在 Amazon S3 中的 DAG 代码窗格中，从下拉列表中选择 `requirements.txt` 的版本。
5. 选择下一步、更新环境。

您可以在环境完成更新后立即开始使用新程序包。

访问 **`requirements.txt`** 的日志

您可以查看调度工作流程并解析 `dags` 文件夹的计划程序的 Apache Airflow 日志。以下步骤介绍如何在 Amazon MWAA 控制台上打开计划程序的日志组，以及如何在日志控制台上访问 Apache Airflow 日志。 CloudWatch

访问 **`requirements.txt`** 的日志

1. 在 Amazon MWAA 控制台上打开环境页面。
2. 选择环境。
3. 在监控窗格上选择 Airflow 计划程序日志组。
4. 在日志流中选择 `requirements_install_ip` 日志。
5. 请参阅 `/usr/local/airflow/.local/bin` 上环境中安装的程序包列表。例如：

```
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. 查看程序包列表以及其中任何程序包在安装过程中是否遇到错误。如果出现问题，您可能会看到类似以下的错误：

```
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))
```

接下来做什么？

使用aws-mwaa-docker-images在本地测试你的 DAGs自定义插件和 Python 依赖关系 GitHub。

删除 Amazon S3 上的文件

本页介绍如何在 Amazon MWAA 环境的 Amazon S3 存储桶中进行版本控制，以及删除 DAG、`plugins.zip` 或 `requirements.txt` 文件的步骤。

目录

先决条件

在完成本页上的步骤之前，您需要具备以下条件。

- 权限 — 您的 AWS 账户 必须已获得管理员授权，访问适用于环境的 AmazonMWAAFullConsoleAccess 访问控制策略。此外，执行角色必须允许 Amazon MWAA 环境访问环境所使用的 AWS 资源。
- 访问权限 — 如果您需要访问公共存储库以便直接在 Web 服务器上安装依赖项，则必须将环境配置为具有公共网络 Web 服务器访问权限。有关更多信息，请参阅the section called “Apache Airflow 访问模式”。
- Amazon S3 配置 — 用于存储 DAG 的 Amazon S3 存储桶、在 `plugins.zip` 中的自定义插件和在 `requirements.txt` 中的 Python 依赖项必须配置为已阻止公共访问和已启用版本控制。

版本控制概述

您 Amazon S3 存储桶中的 `plugins.zip` 和 `requirements.txt` 已进行版本控制。当对象启用 Amazon S3 存储桶版本控制并且从 Amazon S3 存储桶中删除构件（例如 plugins.zip）时，该文件不会被完全删除。每当在 Amazon S3 上删除构件时，都会创建该文件的新副本，该副本是 404（未找到对象）错误/“`I'm not here`”的 0k 文件。Amazon S3 称此为删除标记。与任何其他对象一样，删除标记是带有键名（或键）和版本 ID 的“空”文本。

我们建议定期删除文件版本并删除标记，以降低 Amazon S3 存储桶的存储成本。要完全删除“非当前”（以前的）文件版本，必须删除（这些）文件的所有版本，然后删除该版本的删除标记。

工作方式

Amazon MWAA 每三十秒钟对 Amazon S3 存储桶运行一次同步操作。这会导致 Amazon S3 存储桶中删除的任何 DAG 同步到 Fargate 容器的 Airflow 镜像。

只有当 Amazon MWAA 使用自定义插件和 Python 依赖项为 Fargate 容器构建新的 Airflow 映像时，在环境更新之后，`plugins.zip` 和 `requirements.txt` 文件才会发生更改。如果您删除了 `requirements.txt` 或 `plugins.zip` 文件的当前版本，然后更新环境但不为已删除文件提供新版本，则更新将失败，并显示一条错误消息，例如“`Unable to read version {version number} of file {file name}`”。

删除 Amazon S3 上的 DAG

DAG 文件 (.py) 没有版本控制，可以直接在 Amazon S3 控制台上删除。以下步骤描述如何删除 Amazon S3 桶中的 DAG。

要删除 DAG，请执行以下操作

1. 在 Amazon MWAA 控制台上打开环境页面。
2. 选择环境。
3. 在 S3 中的 DAG 代码窗格中选择 S3 存储桶链接，在控制台上打开存储桶。
4. 选择 dags 文件夹。
5. 选择 DAG，点击删除。
6. 在删除对象？下 ，键入 delete。
7. 选择删除对象。

> **Note**
>
> Apache Airflow 保留了历史上的 DAG 运行。在 Apache Airflow 中运行 DAG 后，无论文件状态如何，它都会保留在 Apache Airflow 列表中，直到您在 Apache Airflow 中将其删除。要删除 Apache Airflow 中的 DAG，请选择链接列中的红色“删除”按钮。

从环境中移除 “当前”的 requirements.txt 或 plugins.zip

目前，没有办法在添加 plugins.zip 或 requirements.txt 后将其从环境中删除，但我们正在努力解决这个问题。在此期间，变通方法是分别指向空文本或 zip 文件。

删除“非当前”（之前）的 requirements.txt 或 plugins.zip 版本

在 Amazon MWAA 上，Amazon S3 存储桶中的 requirements.txt 和 plugins.zip 文件受到版本控制。如果您想完全删除 Amazon S3 存储桶中的这些文件，则必须检索对象（例如 plugins.zip）的当前版本 (121212)，删除该版本，然后移除文件版本的删除标记。

您也可以在 Amazon S3 控制台上删除所有“非当前”（先前）文件版本；但是，您仍需要使用以下选项之一删除删除标记。

- 要检索对象版本，请参阅《Amazon S3 指南》中的从启用版本控制的存储桶检索对象版本。

- 要删除对象版本，请参阅《Amazon S3 指南》中的从启用版本控制的存储桶删除对象版本。
- 要移除删除标记，请参阅《Amazon S3 指南》中的管理删除标记。

使用生命周期删除所有“非当前”（先前）版本并自动删除标记

您可以为 Amazon S3 存储桶配置生命周期策略，以便在一定天数后删除 Amazon S3 存储桶中的 plugins.zip 和 requirements.txt 文件的所有“非当前”（先前）版本，或者移除过期对象的删除标记。

1. 在 Amazon MWAA 控制台上打开环境页面。
2. 选择环境。
3. 在 Amazon S3 的 DAG 代码下，选择 Amazon S3 存储桶。
4. 选择创建生命周期规则。

删除 requirements.txt 所有“非当前”版本并自动删除标记的生命周期策略示例

使用以下示例创建生命周期规则，该规则将在三十天后永久删除 requirements.txt 文件的“非当前”版本及其删除标记。

1. 在 Amazon MWAA 控制台上打开环境页面。
2. 选择环境。
3. 在 Amazon S3 的 DAG 代码下，选择 Amazon S3 存储桶。
4. 选择创建生命周期规则。
5. 在生命周期规则名称中，键入 `Delete previous requirements.txt versions and delete markers after thirty days`。
6. 在前缀中，选择要求。
7. 在生命周期规则操作中，选择永久删除对象的所有先前版本和删除过期的删除标记或未完成的分段上传。
8. 在对象变为先前版本后的天数中，键入 `30`。
9. 在过期对象删除标记中，选择删除过期对象删除标记，对象将在 30 天后永久删除。

接下来做什么？

- 要详细了解 Amazon S3 删除标记，请参阅管理删除标记。
- 要了解有关 Amazon S3 生命周期的更多信息，请参阅过期对象。

网络连接

本指南介绍了您在 Amazon MWAA 环境中所需的 Amazon VPC 网络设置。

各个部分

关于在 Amazon MWAA 上联网

Amazon VPC 是链接到您的虚拟网络 AWS 账户。它通过提供对虚拟基础设施和网络流量分段的精细控制，为您提供云安全性和动态扩展的能力。本页介绍支持适用于 Apache Airflow 的亚马逊托管工作流环境所需的具有公共路由、私有路由或公有和私有路由的 Amazon VPC 基础设施。

目录

术语

公有路由

可以访问互联网的 Amazon VPC 网络。

私有路由

无法访问互联网的 Amazon VPC 网络。

支持什么？

下表描述 Amazon MWAA 支持的 Amazon VPC 类型。

Amazon VPC 类型	支持	
Amazon VPC 属于尝试创建环境的账户。	是	
一个共享的 Amazon VPC，多个 AWS 账户 用户可以在其中创建自己的 AWS 资源。	是	

VPC 基础设施概述

当您创建 Amazon MWAA 环境时，Amazon MWAA 会根据您为环境选择的 Apache Airflow 访问模式为环境创建一到两个 VPC 端点。这些端点显示为弹性网络接口 (ENI)，私有 IP 位于 Amazon VPC 中。创建这些端点后，任何发往这些 IP 的流量都将私密或公开路由到您的环境使用的相应 AWS 服务。

以下部分介绍通过互联网公共地路由流量或在 Amazon VPC 内私密地路由流量所需的 Amazon VPC 基础设施。

通过互联网进行公共路由

本节介绍具有公有路由的环境的 Amazon VPC 基础设施。您需要以下 VPC 基础设施：

- 一个 VPC 安全组 VPC 安全组 充当虚拟防火墙，以控制实例上的入口（入站）和出口（出站）流量。
  - 最多可以指定 5 个安全组。
  - 安全组必须为自己指定自引用的入站规则。
  - 安全组必须为所有流量（`0.0.0.0/0`；对于 IPv6，使用 `::/0`）指定出站规则。
  - 安全组必须允许自引用规则中的所有流量。例如 （推荐）所有访问自引用安全组示例。

  - 安全组可以选择通过指定 HTTPS 端口范围 443 和 TCP 端口范围 5432 来进一步限制流量。例如，（可选）限制入站访问端口 5432 的安全组示例 和 （可选）限制入站访问端口 443 的安全组示例。
- 两个公有子网。公有子网是指与包含指向互联网网关的路由的路由表关联的子网。
  - 需要两个公有子网。这样，如果一个容器出现故障，Amazon MWAA 就可以为另一个可用区中的环境构建新的容器镜像。
  - 这些子网必须位于不同的可用区中。例如 `us-east-1a`、`us-east-1b`。
  - 子网必须路由到具有弹性 IP 地址（EIP）的 NAT 网关（或 NAT 实例）。
  - 子网必须具有将面向互联网的流量定向到互联网网关的路由表。
- 两个私有子网。公有子网是指与包含指向互联网网关的路由的路由表不关联的子网。
  - 需要两个私有子网。这样，如果一个容器出现故障，Amazon MWAA 就可以为另一个可用区中的环境构建新的容器镜像。
  - 这些子网必须位于不同的可用区中。例如 `us-east-1a`、`us-east-1b`。
  - 这些子网必须有通往 NAT 设备（网关或实例）的路由表。
  - 这些子网不得路由到互联网网关。
  - 对于 IPv6 子网，设置为 `assignIpV6AddressOnCreation` 到 `true`。
  - 对于 IPv6 私有子网，必须连接到仅出口互联网网关 (EIGW)。
- 网络访问控制列表（ACL） NACL 管理（通过允许或拒绝规则）子网级别的入站和出站流量。
  - NACL 必须有允许所有流量的入站规则（`0.0.0.0/0`；对于 IPv6，使用 `::/0`）。
  - NACL 必须有允许所有流量的出站规则（`0.0.0.0/0`；对于 IPv6，使用 `::/0`）。
  - 例如 （推荐）示例 ACLs。
- 两个 NAT 网关（或 NAT 实例）。NAT 设备将流量从私有子网中的实例转发到 Internet 或其他 AWS 服务，然后将响应路由回实例。
  - NAT 设备必须连接公有子网。（每个公有子网一个 NAT 设备。）
  - NAT 设备必须将弹性 IPv4 地址（EIP）附加到每个公有子网。
- 互联网网关。互联网网关将 Amazon VPC 连接到互联网和其他 AWS 服务。
  - 互联网网关必须连接到 Amazon VPC。

无法访问互联网的私有路由

本节介绍具有私有路由的环境的 Amazon VPC 基础设施。您需要以下 VPC 基础设施：

- 一个 VPC 安全组 VPC 安全组 充当虚拟防火墙，以控制实例上的入口（入站）和出口（出站）流量。
  - 最多可以指定 5 个安全组。
  - 安全组必须为自己指定自引用的入站规则。
  - 安全组必须为所有流量（`0.0.0.0/0`；对于 IPv6，使用 `::/0`）指定出站规则。
  - 安全组必须允许自引用规则中的所有流量。例如（推荐）所有访问自引用安全组示例。
  - 安全组可以选择通过指定 HTTPS 端口范围 443 和 TCP 端口范围 5432 来进一步限制流量。例如，（可选）限制入站访问端口 5432 的安全组示例 和 （可选）限制入站访问端口 443 的安全组示例。
- 两个私有子网。公有子网是指与包含指向互联网网关的路由的路由表不关联的子网。
  - 需要两个私有子网。这样，如果一个容器出现故障，Amazon MWAA 就可以为另一个可用区中的环境构建新的容器镜像。
  - 这些子网必须位于不同的可用区中。例如 `us-east-1a`、`us-east-1b`。
  - 这些子网必须有通往 VPC 端点的路由表。
  - 子网必须具有指向 EIGW 的路由表，才能作为 DAG 的一部分从互联网下载。
  - 这些子网不得有通往 NAT 设备（网关或实例）的路由表，也不能有互联网网关。
- 网络访问控制列表（ACL） NACL 管理（通过允许或拒绝规则）子网级别的入站和出站流量。
  - NACL 必须有允许所有流量的入站规则（`0.0.0.0/0`；对于 IPv6，使用 `::/0`）。
  - NACL 必须有拒绝所有流量的出站规则（`0.0.0.0/0`；对于 IPv6，使用 `::/0`）。
  - 例如（推荐）示例 ACLs。
- 本地路由表。本地路由表是指用于 VPC 内通信的默认路由。
  - 本地路由表必须与私有子网关联。
  - 本地路由表必须启用 VPC 中的实例与您自己的网络进行通信。例如，如果您使用访问您的 Apache Airflow 网络服务器的 VPC 接口终端节点，则路由表必须路由到 VPC 终端节点。 AWS Client VPN
- 您的环境使用的每项 AWS 服务的 VPC 终端节点，Apache Airflow VPC 终端节点与您的亚马逊 MWAA 环境位于相同且 AWS 区域 亚马逊 VPC 中。
  - 环境使用的每项 AWS 服务的 VPC 终端节点和 Apache Airflow 的 VPC 终端节点。例如（必需）VPC 端点。
  - VPC 端点必须有启用的私有 DNS。
  - VPC 端点必须与您环境的两个私有子网关联。
  - VPC 端点必须与您环境的安全组相关联。

- 必须将每个终端节点的 VPC 终端节点策略配置为允许访问环境使用的 AWS 服务。例如 （推荐）允许所有人访问的 VPC 端点策略示例。
- Amazon S3 的 VPC 端点策略必须配置为允许访问存储桶。例如 （推荐）允许访问存储桶的 Amazon S3 网关端点策略示例。

Amazon VPC 和 Apache Airflow 访问模式的示例用例

本部分介绍 Amazon VPC 中网络访问的不同使用案例，以及在 Amazon MWAA 控制台上选择的 Apache Airflow Web 服务器访问模式。

允许访问互联网-新的 Amazon VPC 网络

如果贵组织允许在 VPC 中访问互联网，并且您希望用户通过互联网访问 Apache Airflow Web 服务器，请执行以下操作：

1. 创建可访问互联网的 Amazon VPC 网络。
2. 为 Apache Airflow Web 服务器创建具有公有网络访问模式的环境。
3. 我们的建议：我们建议使用 CloudFormation 快速入门模板来同时创建 Amazon VPC 基础设施、Amazon S3 存储桶和 Amazon MWAA 环境。要了解更多信息，请参阅 Amazon MWAA 的快速入门教程。

如果贵组织允许在 VPC 中访问互联网，并且您希望将 Apache Airflow Web 服务器访问权限限于 VPC 内的用户，请执行以下操作：

1. 创建可访问互联网的 Amazon VPC 网络。
2. 创建一种机制，用于从计算机访问 Apache Airflow Web 服务器的 VPC 接口端点。
3. 为 Apache Airflow Web 服务器创建具有私有网络访问模式的环境。
4. 我们的建议：
    a. 我们建议使用中的 Amazon MWAA 控制台或中的选项一：在 Amazon MWAA 控制台上创建 VPC 网络 CloudFormation 模板。选项二：创建可访问互联网的 Amazon VPC 网络
    b. 我们建议在中使用配置 AWS Client VPN 对您的 Apache Airflow 网络服务器的访问权限。教程：使用配置私有网络访问权限 AWS Client VPN

不允许访问互联网-新的 Amazon VPC 网络

如果您的组织不允许在您的 VPC 中访问互联网，并且您希望 Apache Airflow 网络服务器仍可通过互联网访问：

1. 创建没有互联网访问权限的 Amazon VPC 网络。
2. 为您的 Apache Airflow 网络服务器创建一个同时使用公共网络和私有网络访问模式的环境。此选项适用于 Apache Airflow 版本 3.2.1 及更高版本。
3. 我们的建议：我们建议使用该 CloudFormation 模板创建没有互联网访问权限的 Amazon VPC，并为 Amazon MWAA 在中使用的每项 AWS 服务创建 VPC 终端节点。选项三：创建不可访问互联网的 Amazon VPC 网络

如果您的组织不允许在您的 VPC 中访问互联网，并且您想将 Apache Airflow 网络服务器限制为仅允许您的 VPC 内的用户访问：

1. 创建没有互联网访问权限的 Amazon VPC 网络。
2. 创建一种机制，用于从计算机访问 Apache Airflow Web 服务器的 VPC 接口端点。
3. 为您的环境使用的每项 AWS 服务创建 VPC 终端节点。
4. 为 Apache Airflow Web 服务器创建具有私有网络访问模式的环境。
5. 我们的建议：
    a. 我们建议使用该 CloudFormation 模板创建没有互联网访问权限的 Amazon VPC，并为 Amazon MWAA 在中使用的每项 AWS 服务创建 VPC 终端节点。选项三：创建不可访问互联网的 Amazon VPC 网络
    b. 我们建议在中使用配置 AWS Client VPN 对您的 Apache Airflow 网络服务器的访问权限。教程：使用配置私有网络访问权限 AWS Client VPN

不允许访问互联网-现有 Amazon VPC 网络

如果贵组织不允许在 VPC 中访问互联网，并且您已经拥有所需的无法访问互联网的 Amazon VPC 网络：

1. 为您的环境使用的每项 AWS 服务创建 VPC 终端节点。
2. 为 Apache Airflow 创建 VPC 端点。
3. 创建一种机制，用于从计算机访问 Apache Airflow Web 服务器的 VPC 接口端点。

4. 为 Apache Airflow Web 服务器创建具有私有网络访问模式的环境。
5. 我们的建议：
   a. 我们建议创建并连接亚马逊 MWAA 使用的每项 AWS 服务所需的 VPC 终端节点，以及 Apache Airflow 所需的 VPC 终端节点。使用私有路由在 Amazon VPC 中创建所需的 VPC 服务端点
   b. 我们建议在中使用配置 AWS Client VPN 对您的 Apache Airflow 网络服务器的访问权限。教程：使用配置私有网络访问权限 AWS Client VPN

Amazon MWAA 上的 VPC 安全

本页介绍用于保护 Amazon MWAA 环境的 Amazon VPC 组件以及这些组件所需的配置。

目录

术语

公有路由

可以访问互联网的 Amazon VPC 网络。

私有路由

无法访问互联网的 Amazon VPC 网络。

安全性概述

安全组和访问控制列表 (ACLs) 提供了使用您指定的规则控制跨子网和您的 Amazon VPC 实例的网络流量的方法。

- 进出子网的网络流量可以通过访问控制列表 (ACLs) 进行控制。您只需要一个 ACL，并且可以在多个环境中使用相同的 ACL。
- 进出实例的网络流量可以由 Amazon VPC 安全组控制。您可以在每个环境中使用一到五个安全组。
- 进出实例的网络流量也可以通过 VPC 端点策略进行控制。如果贵组织不允许在 Amazon VPC 内访问互联网，并且您使用的是带有私有路由的 Amazon VPC 网络，则需要为 AWS VPC 端点和 Apache Airflow VPC 端点制定一个 VPC 端点策略。

网络访问控制列表 (ACLs)

网络访问控制列表（ACL）可以管理（通过允许或拒绝规则）子网级别的入站和出站流量。ACL 是无状态的，这意味着必须单独、明确指定入站和出站规则。它用于指定 VPC 网络中允许从实例进出的网络流量类型。

每个 Amazon VPC 都有允许所有入站和出站流量的默认 ACL。您可以编辑默认 ACL 规则，也可以创建自定义 ACL 并将其附加到子网。一个子网在任何时候只能连接一个 ACL，但一个 ACL 可以连接到多个子网。

（推荐）示例 ACLs

以下示例说明了入站和出站 ACL 规则，可用于使用公有路由或私有路由的 Amazon VPC。

规则编号	Type	协议	端口范围	源	允许/拒绝
100	所有 IPv4 流量	全部	全部	0.0.0.0/0	允许
*	所有 IPv4 流量	全部	全部	0.0.0.0/0	拒绝

VPC 安全组

VPC 安全组可以作为虚拟防火墙，用于控制一个或多个实例级别的流量。安全组是有状态的，这意味着当允许入站连接时，它可以进行回复。它用于指定 VPC 网络中允许从实例进入的网络流量类型。

每个 Amazon VPC 都有一个默认安全组。默认情况下，它没有入站规则。它有一条允许所有出站流量的出站规则。您可以编辑默认安全组规则，也可以创建自定义安全组并将其附加到 Amazon VPC。在 Amazon MWAA 上，您需要配置入站和出站规则，以便在 NAT 网关上引导流量。

（推荐）所有访问自引用安全组示例

以下示例说明了入站安全组规则，将允许使用公有路由或私有路由的 Amazon VPC 的所有流量。本例中的安全组必须为自己指定自引用规则。

Type	协议	源类型	来源		
所有流量	All	全部	sg-0909e8e81919/-group my-mwaa-vpc-security		

以下示例说明了出站安全组规则。

Type	协议	源类型	来源		
所有流量	All	全部	0.0.0.0/0		

（可选）限制入站访问端口 5432 的安全组示例

以下示例说明了入站安全组规则，这些规则允许环境的 Amazon Aurora PostgreSQL 元数据数据库（由 Amazon MWAA 拥有）在端口 5432 上的所有 HTTPS 流量。

> **Note**
>
> 如果您选择使用此规则限制流量，则需要添加另一条规则以允许端口 443 上的 TCP 流量。

Type	协议	端口范围	Source type（源类型）	来源	
自定义 TCP	TCP	5432	自定义	sg-0909e8e81919/-group my-mwaa-vpc-security	

（可选）限制入站访问端口 443 的安全组示例

以下示例说明了允许 Apache Airflow Web 服务器端口 443 上所有 TCP 流量的入站安全组规则。

Type	协议	端口范围	Source type（源类型）	来源	
HTTPS	TCP	443	自定义	sg-0909e8e81919/-group my-mwaa-vpc-security	

VPC 端点策略（仅限私有路由）

VPC 终端节点 (AWS PrivateLink) 策略控制从您的私有子网访问 AWS 服务。VPC 端点策略是一种 IAM 资源策略，您可以将其附加到 VPC 网关或接口端点。本节介绍每个 VPC 端点的 VPC 端点策略所需的权限。

我们建议对您创建的每个 VPC 终端节点使用允许完全访问所有 AWS 服务的 VPC 接口终端节点策略，并仅使用您的执行角色来获得 AWS 权限。

（推荐）允许所有人访问的 VPC 端点策略示例

对于使用私有路由的 Amazon VPC，以下示例介绍了 VPC 接口端点策略。

```
{
  "Statement": [
    {
      "Action": "*",
      "Effect": "Allow",
      "Resource": "*",
      "Principal": "*"
    }
  ]
}
```

（推荐）允许访问存储桶的 Amazon S3 网关端点策略示例

以下示例介绍了一个 VPC 网关端点策略，对于使用私有路由的 Amazon VPC，它提供了对 Amazon ECR 操作所需的 Amazon S3 存储桶的访问权限。除了存储您的文件 DAGs 和支持文件的存储桶外，还需要这样才能检索您的 Amazon ECR 映像。

```
{
  "Statement": [
    {
      "Sid": "Access-to-specific-bucket-only",
      "Principal": "*",
      "Action": [
        "s3:GetObject"
      ],
      "Effect": "Allow",
      "Resource": ["arn:aws:s3:::prod-us-east-1-starport-layer-bucket/*"]
    }
  ]
}
```

在 Amazon MWAA 上管理对服务特定 Amazon VPC 端点的访问

一个 VPC 终端节点 (AWS PrivateLink)，可用于将您的 VPC 私密连接到托管的服务， AWS 而无需互联网网关、NAT 设备、VPN 或防火墙代理。这些终端节点是水平可扩展且高度可用的虚拟设备，允许在您的 VPC 中的实例与 AWS 服务之间进行通信。本页介绍由 Amazon MWAA 创建的 VPC 端点，

以及如果您在 Amazon MWAA 上选择了私有网络访问模式，如何访问 Apache Airflow Web 服务器的 VPC 端点。

> **Note**
>
> 如果您选择公共网络和私有网络访问，则网络服务器的 VPC 终端节点将由 Amazon MWAA 自动创建和管理，以实现工作人员连接。无需额外设置 VPC 终端节点，也无需配置对 VPC 终端节点的访问权限即可通过浏览器访问 Apache Airflow 用户界面。

目录

定价

- AWS PrivateLink 定价

VPC 端点概述

当您创建 Amazon MWAA 环境时，Amazon MWAA 会为环境创建一到两个 VPC 端点。这些端点显示为弹性网络接口 (ENI)，私有 IP 位于 Amazon VPC 中。创建这些端点后，任何发往这些 IP 的流量都将私下或公开路由到您的环境使用的相应 AWS 服务。

公有网络访问模式

如果您为 Apache Airflow Web 服务器选择了公有网络访问模式，则网络流量将通过互联网公开路由。

- Amazon MWAA 为 Amazon Aurora PostgreSQL 元数据数据库创建 VPC 接口端点。端点是在映射到私有子网的可用区中创建的，并且独立于其他 AWS 账户。
- 然后，Amazon MWAA 会将私有子网中的 IP 地址绑定到接口端点。这旨在支持从 Amazon VPC 的每个可用区绑定一个 IP 的最佳实践。

私有网络访问模式

如果您为 Apache Airflow Web 服务器选择了私有网络访问模式，则网络流量将在 Amazon VPC 内私密路由。

- Amazon MWAA 为 Apache Airflow Web 服务器创建一个 VPC 接口端点，为 Amazon Aurora PostgreSQL 元数据数据库创建接口端点。端点是在映射到私有子网的可用区中创建的，并且独立于其他 AWS 账户。
- 然后，Amazon MWAA 会将私有子网中的 IP 地址绑定到接口端点。这旨在支持从 Amazon VPC 的每个可用区绑定一个 IP 的最佳实践。

允许使用其他 AWS 务

接口终端节点在 AWS Identity and Access Management (IAM) 中使用您的环境的执行角色来管理您的环境所用 AWS 资源的权限。随着为环境开启的 AWS 服务越来越多，每项服务都要求您使用环境的执行角色配置权限。要添加权限，请参阅 Amazon MWAA 执行角色。

如果您为 Apache Airflow Web 服务器选择了私有网络访问模式，则还必须在 VPC 端点策略中允许每个端点的权限。要了解更多信息，请参阅 the section called “VPC 端点策略（仅限私有路由）”。

访问 VPC 端点

本节介绍如何访问 Amazon MWAA 创建的 VPC 端点，以及如何识别 Apache Airflow VPC 端点的私有 IP 地址。

在 Amazon VPC 控制台中访问 VPC 端点

下一节显示了如何访问 Amazon MWAA 创建的 VPC 端点，以及可能已创建的任何 VPC 端点（如果您在 Amazon VPC 中使用私有路由）。

访问 VPC 端点

1. 打开 Amazon VPC 控制台的 端点页面。
2. 选择你的 AWS 区域。
3. 请参阅由 Amazon MWAA 创建的 VPC 接口端点，以及您可能已创建的任何 VPC 端点（如果您在 Amazon VPC 中使用私有路由）。

要详细了解使用私有路由的 Amazon VPC 所需的 VPC 服务端点，请参阅 使用私有路由在 Amazon VPC 中创建所需的 VPC 服务端点。

识别 Apache Airflow Web 服务器及其 VPC 端点的私有 IP 地址

以下步骤介绍如何检索 Apache Airflow Web 服务器的主机名、其 VPC 接口端点及其私有 IP 地址。

1. 使用以下 AWS Command Line Interface (AWS CLI) 命令检索 Apache Airflow 网络服务器的主机名。

```
aws mwaa get-environment --name YOUR_ENVIRONMENT_NAME --query
 'Environment.WebserverUrl'
```

   您会收到类似如下响应所示的内容：

```
"99aa99aa-55aa-44a1-a91f-f4552cf4e2f5-vpce.c10.us-west-2.airflow.amazonaws.com"
```

2. 对上一个命令的响应中返回的主机名运行 dig 命令。例如：

```
dig CNAME +short 99aa99aa-55aa-44a1-a91f-f4552cf4e2f5-vpce.c10.us-
west-2.airflow.amazonaws.com
```

   您会收到类似如下响应所示的内容：

```
vpce-0699aa333a0a0a0-bf90xjtr.vpce-svc-00bb7c2ca2213bc37.us-
west-2.vpce.amazonaws.com.
```

3. 使用以下 AWS Command Line Interface (AWS CLI) 命令检索上一个命令的响应中返回的 VPC 终端节点 DNS 名称。例如：

```
aws ec2 describe-vpc-endpoints | grep vpce-0699aa333a0a0a0-bf90xjtr.vpce-
svc-00bb7c2ca2213bc37.us-west-2.vpce.amazonaws.com.
```

您会收到类似如下响应所示的内容：

```
"DnsName": "vpce-066777a0a0a0-bf90xjtr.vpce-svc-00bb7c2ca2213bc37.us-
west-2.vpce.amazonaws.com",
```

4. 对 Apache Airflow 主机名及其 VPC 端点 DNS 名称运行 nslookup 或 dig 命令以检索 IP 地址。例如：

```
dig +short YOUR_AIRFLOW_HOST_NAME YOUR_AIRFLOW_VPC_ENDPOINT_DNS
```

您会收到类似如下响应所示的内容：

```
192.0.5.1
192.0.6.1
```

访问 Apache Airflow Web 服务器的 VPC 端点（私有网络访问）

如果您为 Apache Airflow Web 服务器选择了私有网络访问模式，则需要创建一种机制来访问 Apache Airflow Web 服务器的 VPC 接口端点。对于这些资源，我们建议使用与 Amazon MWAA 环境相同的 Amazon VPC、VPC 安全组和私有子网。

使用 AWS Client VPN

AWS Client VPN 是一项基于客户端的托管 VPN 服务，可用于安全地访问本地网络中的 AWS 资源和资源。它使用 OpenVPN 客户端从任何位置提供安全的 TLS 连接。

我们建议按照 Amazon MWAA 教程来配置客户端 VPN：教程：使用配置私有网络访问权限 AWS Client VPN。

使用 Linux 堡垒主机

堡垒主机是一种服务器，其目的是提供从外部网络（例如从计算机通过互联网）访问私有网络的权限。Linux 实例位于公有子网中，这些实例设有一个安全组，该安全组允许从连接到运行堡垒主机的底层 Amazon EC2 实例的安全组进行 SSH 访问。

我们建议按照 Amazon MWAA 教程来配置 Linux 堡垒主机：教程：使用 Linux 堡垒主机配置私有网络访问权限。

使用负载均衡器（高级）

下一节显示了应用于应用程序负载均衡器所需的配置。

1. 目标组。您需要使用指向 Apache Airflow Web 服务器及其 VPC 接口端点的私有 IP 地址的目标组。我们建议将两个私有 IP 地址都指定为注册目标，因为只使用一个会降低可用性。有关如何识别私有 IP 地址的更多信息，请参阅 the section called “识别 Apache Airflow Web 服务器及其 VPC 端点的私有 IP 地址”。
2. 状态代码。我们建议在目标群组设置中使用 `200` 和 `302` 状态码。否则，如果 Apache Airflow Web 服务器的 VPC 端点响应为 `302 Redirect` 错误，则目标可能会被标记为运行状况不佳。
3. HTTPS 侦听器。您需要为 Apache Airflow Web 服务器指定目标端口。例如：

协议	端口：
HTTPS	443

4. ACM 新域名。如果要在中关联 SSL/TLS 证书 AWS Certificate Manager，则需要为负载均衡器的 HTTPS 侦听器创建一个新域。
5. ACM 证书区域。如果要在中关联 SSL/TLS 证书 AWS Certificate Manager，则需要将证书上传到与您的环境 AWS 区域 相同的证书。例如：

    - Example要上传证书的区域

```
aws acm import-certificate --certificate fileb://Certificate.pem --certificate-
chain fileb://CertificateChain.pem --private-key fileb://PrivateKey.pem --
region us-west-2
```

使用私有路由在 Amazon VPC 中创建所需的 VPC 服务端点

无法访问互联网的现有 Amazon VPC 网络需要额外的 VPC 服务端点 (AWS PrivateLink) 才能在 Amazon MWAA 上使用 Apache Airflow。本页介绍亚马逊 MWAA 使用的 AWS 服务所需的 VPC 终端节点、Apache Airflow 所需的 VPC 终端节点，以及如何使用私有路由创建 VPC 终端节点并将其连接到现有亚马逊 VPC。

> **Note**
>
> 如果您选择同时访问公有网络和私有网络，则网络服务器的 VPC 终端节点将由 Amazon MWAA 自动创建和管理。您无需为 Apache Airflow 网络服务器连接创建 VPC 终端节点。但是，如果您的 Amazon VPC 无法访问互联网，则仍需要使用本页所述的其他 AWS 服务（例如 Amazon S3、 CloudWatch 日志、SQS、KMS 和 Amazon ECR）的 VPC 终端节点。

目录

定价

- AWS PrivateLink 定价

私有网络和私有路由

私有网络访问模式将访问 Apache Airflow UI 的权限限制为 Amazon VPC 中已获准访问环境 IAM 策略的用户。

创建具有私有网络访问权限的环境时，必须将所有依赖项打包到 Python 轮档案 (`.whl`) 中，然后在`.whl`中引用`requirements.txt`。有关使用 Wheel 打包和安装依赖项的说明，请参阅使用 Python Wheel 管理依赖项。

下图描述了在 Amazon MWAA 控制台上哪里可以找到私有网络选项。

- 私有路由。无法访问互联网的 Amazon VPC 会限制 VPC 内的网络流量。本页假设您的亚马逊 VPC 无法访问互联网，并且您的环境使用的每项 AWS 服务都需要 VPC 终端节点，Apache Airflow 需要与您的亚马逊 MWAA 环境相同且 AWS 区域 亚马逊 VPC 的 VPC 终端节点。

（必需）VPC 端点

下一节显示了无法访问互联网的 Amazon VPC 所需的 VPC 端点。它列出了亚马逊 MWAA 使用的每项 AWS 服务的 VPC 终端节点，包括 Apache Airflow 所需的 VPC 终端节点。

```
com.amazonaws.us-east-1.s3
com.amazonaws.us-east-1.monitoring
com.amazonaws.us-east-1.logs
com.amazonaws.us-east-1.sqs
com.amazonaws.us-east-1.kms
```

> **Note**
>
> 在使用 Transit Gateway 或任何其他不直接连接到 AWS API 终端节点的路由时，我们建议您为以下服务添加 AWS PrivateLink 到 Amazon MWAA 私有子网：
>
> - Amazon S3
> - Amazon SQS
> - CloudWatch 日志
> - CloudWatch 指标
> - AWS KMS （如果适用）
>
> 这可确保您的 Amazon MWAA 环境能够安全高效地与这些服务进行通信，无需通过公共互联网路由流量，从而提高安全性和性能。

连接所需的 VPC 端点

本节介绍为使用私有路由的 Amazon VPC 连接所需的 VPC 端点的步骤。

需要 VPC 终端节点 AWS 务

以下部分显示了将环境所用 AWS 服务的 VPC 终端节点连接到现有 Amazon VPC 的步骤。

将 VPC 端点连接到私有子网

1. 打开 Amazon VPC 控制台的 端点页面。
2. 选择你的 AWS 区域。
3. 创建 Amazon S3 网关端点：
   a. 选择创建端点。
   b. 在按属性筛选或按关键字搜索文本字段中，键入:**.s3**，然后按键盘上的 Enter。
   c. 我们建议为网关类型选择列出的服务端点。

      例如，**com.amazonaws.us-west-2.s3 amazon Gateway**
   d. 在 VPC 中选择环境的 Amazon VPC。
   e. 确保选择了位于不同可用区的两个私有子网，并通过选择启用 DNS 名称来启用该私有 DNS。
   f. 选择环境的 Amazon VPC 安全组。
   g. 在策略中选择完全访问权限。
   h. 选择创建端点。
4. 为 CloudWatch 日志创建终端节点：
   a. 选择创建端点。
   b. 在按属性筛选或按关键字搜索文本字段中，键入:**.logs**，然后按键盘上的 Enter。
   c. 选择服务端点。
   d. 在 VPC 中选择环境的 Amazon VPC。
   e. 确保选择了位于不同可用区的两个私有子网，并启用启用 DNS 名称。
   f. 选择环境的 Amazon VPC 安全组。
   g. 在策略中选择完全访问权限。
   h. 选择创建端点。
5. 创建用于 CloudWatch 监控的终端节点：
   a. 选择创建端点。
   b. 在按属性筛选或按关键字搜索文本字段中，键入:**.monitoring**，然后按键盘上的 Enter。
   c. 选择服务端点。
   d. 在 VPC 中选择环境的 Amazon VPC。
   e. 确保选择了位于不同可用区的两个私有子网，并启用启用 DNS 名称。

f. 选择环境的 Amazon VPC 安全组。

g. 在策略中选择完全访问权限。

h. 选择创建端点。

6. 为 Amazon SQS 创建端点：

a. 选择创建端点。

b. 在按属性筛选或按关键字搜索文本字段中，键入:**.sqs**，然后按键盘上的 Enter。

c. 选择服务端点。

d. 在 VPC 中选择环境的 Amazon VPC。

e. 确保选择了位于不同可用区的两个私有子网，并启用启用 DNS 名称。

f. 选择环境的 Amazon VPC 安全组。

g. 在策略中选择完全访问权限。

h. 选择创建端点。

7. 为以下对象创建终端节点 AWS KMS：

a. 选择创建端点。

b. 在按属性筛选或按关键字搜索文本字段中，键入:**.kms**，然后按键盘上的 Enter。

c. 选择服务端点。

d. 在 VPC 中选择环境的 Amazon VPC。

e. 确保选择了位于不同可用区的两个私有子网，并启用启用 DNS 名称。

f. 选择环境的 Amazon VPC 安全组。

g. 在策略中选择完全访问权限。

h. 选择创建端点。

Apache Airflow 所需的 VPC 端点

下一节显示了将 Apache Airflow 的 VPC 端点连接到现有 Amazon VPC 的步骤。

将 VPC 端点连接到私有子网

1. 打开 Amazon VPC 控制台的 端点页面。

2. 选择你的 AWS 区域。

3. 为 Apache Airflow API 创建端点：

a. 选择创建端点。

b. 在按属性筛选或按关键字搜索文本字段中，键入:**.airflow.api**，然后按键盘上的 Enter。

c. 选择服务端点。

d. 在 VPC 中选择环境的 Amazon VPC。

e. 确保选择了位于不同可用区的两个私有子网，并启用启用 DNS 名称。

f. 选择环境的 Amazon VPC 安全组。

g. 在策略中选择完全访问权限。

h. 选择创建端点。

4. 为 Apache Airflow 环境创建第一个端点：

a. 选择创建端点。

b. 在按属性筛选或按关键字搜索文本字段中，键入:**.airflow.env**，然后按键盘上的 Enter。

c. 选择服务端点。

d. 在 VPC 中选择环境的 Amazon VPC。

e. 确保选择了位于不同可用区的两个私有子网，并启用启用 DNS 名称。

f. 选择环境的 Amazon VPC 安全组。

g. 在策略中选择完全访问权限。

h. 选择创建端点。

5. 为 Apache Airflow 操作创建第二个端点：

a. 选择创建端点。

b. 在按属性筛选或按关键字搜索文本字段中，键入:**.airflow.ops**，然后按键盘上的 Enter。

c. 选择服务端点。

d. 在 VPC 中选择环境的 Amazon VPC。

e. 确保选择了位于不同可用区的两个私有子网，并启用启用 DNS 名称。

f. 选择环境的 Amazon VPC 安全组。

g. 在策略中选择完全访问权限。

h. 选择创建端点。

（可选）为 Amazon S3 VPC 接口端点启用私有 IP 地址

Amazon S3 接口端点不支持私有 DNS。S3 端点请求仍会解析为公有 IP 地址。要将 S3 地址解析为私有 IP 地址，您需要在 Route 53 中为 S3 区域端点添加私有托管区。

使用 Route 53

本节介绍使用 Route 53 为 S3 接口端点启用私有 IP 地址的步骤。

1. 为 Amazon S3 VPC 接口端点（例如 s3.eu-west-1.amazonaws.com）创建私有托管区并将其与 Amazon VPC 关联。
2. 为 Amazon S3 VPC 接口端点（例如 s3.eu-west-1.amazonaws.com）创建别名 A 记录，该记录可解析为 VPC 接口端点 DNS 名称。
3. 为 Amazon S3 接口端点（例如，*. s3.eu-west-1.amazonaws.com）创建一个别名 A 通配符记录，该记录可解析为 VPC 接口端点 DNS 名称。

带有自定义 DNS 解析

如果 Amazon VPC 使用自定义 DNS 路由，则需要通过创建别名记录在 DNS 解析器（不是 Route 53，通常是运行 DNS 服务器的 EC2 实例）中进行更改。例如：

```
Name: s3.us-west-2.amazonaws.com
Type: CNAME
Value:  *.vpce-0f67d23e37648915c-e2q2e2j3.s3.us-west-2.vpce.amazonaws.com
```

在 Amazon MWAA 上管理您自己的 Amazon VPC 端点

亚马逊 MWAA 使用亚马逊 VPC 终端节点与设置 Apache Airflow 环境所需的各种 AWS 服务集成。管理自己的端点有两个主要使用案例：

1. 这意味着，当你使用管理 AWS 账户 多个资源和共享资源时，你可以在共享的 Amazon VPC 中创建 Apache Airfl AWS Organizationsow 环境。
2. 您可以通过将权限范围缩小到使用端点的特定资源，从而执行更严格的访问策略。

如果您选择管理自己的 VPC 端点，则需要负责为环境 RDS for PostgreSQL 数据库和环境 Web 服务器创建自己的端点。

有关 Amazon MWAA 如何在云中部署 Apache Airflow 的更多信息，请参阅 Amazon MWAA 架构图。

> **Important**
>
> Amazon MWAA 不会验证客户管理的终端节点的 IP 地址类型 (AddressType) 选择，因此请务必正确指定AddressType（有效选项为 IPv4 或）。 IPv6

在共享 Amazon VPC 中创建环境

如果您使用管理多个 AWS 账户 共享资源的资源，则可以AWS Organizations将客户管理的 VPC 终端节点与 Amazon MWAA 配合使用，与组织中的其他账户共享环境资源。

配置共享 VPC 访问权限时，拥有主要 Amazon VPC 的账户（所有者）将与属于同一组织的其他账户（参与者）共享 Amazon MWAA 所需的两个私有子网。共享这些子网的参与者账户可以查看、创建、修改和删除共享 Amazon VPC 中的环境。

假设您有一个账户 Owner，该账户充当组织中的 Root 账户并拥有 Amazon VPC 资源，此外您还有一个参与者账户 Participant，该账户是同一组织的成员。当 Participant 在其与 Owner 共享的 Amazon VPC 中创建新的 Amazon MWAA 时，Amazon MWAA 将首先创建服务 VPC 资源，然后进入 PENDING 状态最长 72 小时。

环境状态从 CREATING 变为 PENDING 后，代表 Owner 的主体将创建所需的端点。为此，Amazon MWAA 会在 Amazon MWAA 控制台中列出数据库和 Web 服务器端点。您也可以调用 GetEnvironment API 操作来获取服务端点。

> **Note**
>
> 如果您用于共享资源的 Amazon VPC 是一个私有 Amazon VPC，则仍必须完成the section called “管理 VPC 端点访问”中所述的步骤。本主题介绍如何设置与集 AWS 成的其他 AWS 服务（例如 Amazon ECR、Amazon ECS 和 Amazon SQS）相关的另一组 Amazon VPC 终端节点。这些服务对于在云中运行和管理 Apache Airflow 环境至关重要。

先决条件

在共享 VPC 中创建 Amazon MWAA 环境前，您需要具备以下资源：

- 答 AWS 账户Owner，用作拥有亚马逊 VPC 的账户。
- 一个作为根创建的 AWS Organizations 组织单位 MyOrganization。

- 第二个 AWS 账户，Participant，MyOrganization用于为创建新环境的参与者账户提供服务。

此外，我们建议您首先自行了解在 Amazon VPC 中共享资源时，所有者和参与者的责任和权限。

创建 Amazon VPC

首先，创建一个所有者和参与者账户将会共享的新 Amazon VPC：

1. 使用登录控制台0wner，然后打开 CloudFormation 控制台。使用以下模板创建一个堆栈。此堆栈预置了多种网络资源，包括 Amazon VPC，以及这两个账户在此场景中将会共享的子网。

```
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. 配置新的 Amazon VPC 资源后，导航到 AWS Resource Access Manager 控制台，然后选择创建资源共享。
3. 从可与 `Participant` 共享的可用子网列表中选择您在第一步中创建的子网。

创建 环境

完成以下步骤，以使用客户管理型 Amazon VPC 端点创建 Amazon MWAA 环境。

1. 使用 `Participant` 登录并打开 Amazon MWAA 控制台。完成第一步：指定详细信息，为您的新环境指定 Amazon S3 存储桶、DAG 文件夹和依赖项等。有关更多信息，请参阅开始使用。
2. 在配置高级设置页面的联网下，从共享 Amazon VPC 中选择子网。
3. 在端点管理下，从下拉列表中选择客户。
4. 保持页面上其余选项的默认值，然后在检查并创建页面上选择创建环境。

环境开始时处于 CREATING 状态，然后会变为 PENDING 状态。环境处于 PENDING 状态时，使用控制台写下数据库端点服务名称和 Web 服务器端点服务名称（如果您设置了私有 Web 服务器）。

当您使用 Amazon MWAA 控制台创建新环境时，Amazon MWAA 会创建一个新的安全组，其中包含所需的入站和出站规则。记下安全组 ID。

在下一节中，Owner 将使用服务端点和安全组 ID 在共享 Amazon VPC 中创建新的 Amazon VPC 端点。

创建 Amazon VPC 端点

完成以下步骤，为环境创建所需的 Amazon VPC 端点。

1. 登录 “ AWS 管理控制台 使用”Owner，“打开” https://console.aws.amazon.com/vpc/。
2. 从左侧导航面板中选择安全组，然后使用以下入站和出站规则在共享 Amazon VPC 中创建新的安全组：

	Type	协议	Source type（源类型）	来源
入站	所有流量	All	全部	您的环境安全组
出站	所有流量	All	全部	0.0.0.0/0

> **Warning**
>
> Owner 账户必须在 Owner 账户中设置一个安全组，以允许从新环境到共享 Amazon VPC 的流量。为此，您可以在 Owner 中创建一个新安全组，也可以编辑现有的安全组。

3. 选择端点，然后使用之前步骤中的端点服务名称为环境数据库和 Web 服务器（如果处于私有模式）创建新的端点。选择共享 Amazon VPC、您用于环境的子网以及环境的安全组。

如果成功，环境将从 PENDING 状态变回 CREATING 状态，最后变为 AVAILABLE 状态。如果为 AVAILABLE 状态，则您可以登录到 Apache Airflow 控制台。

共享 Amazon VPC 问题排查

可以使用以下参考来解决您在共享 Amazon VPC 中创建环境时遇到的问题。

环境在 **PENDING** 状态后进入 **CREATE_FAILED** 状态

- 验证 Owner 是在使用 AWS Resource Access Manager 与 Participant 共享子网。
- 验证数据库和 Web 服务器的 Amazon VPC 端点是在与环境关联的相同子网中创建的。
- 验证用于端点的安全组允许来自用于环境的安全组的流量。Owner 账户会创建将 Participant 中的安全组引用为 *123456789012*/*security-group-id* 的规则。

Type	协议	Source type（源类型）	来源
所有流量	All	全部	*123456789012* /*sg-0909e8e81919*

有关更多信息，请参阅所有者和参与者的责任和权限。

环境停滞在 **PENDING** 状态

验证每个 VPC 端点的状态，以确保其状态为 Available。如果使用私有 Web 服务器配置环境，则还必须为该 Web 服务器创建端点。如果环境停止在 PENDING 状态，则可能表明缺少私有 Web 服务器端点。

收到 **The Vpc Endpoint Service '*vpce-service-name*' does not exist** 错误

如果您看到以下错误，请验证创建端点的账户是否位于拥有该共享 VPC 的 Owner 账户中：

```
ClientError: An error occurred (InvalidServiceName) when calling the
 CreateVpcEndpoint operation:

The Vpc Endpoint Service 'vpce-service-name' does not exist
```

Amazon MWAA 的教程

本指南包括使用和配置适用于 Apache Airflow 的亚马逊托管工作流程环境的 step-by-step教程。

主题

教程：使用配置私有网络访问权限 AWS Client VPN

本教程将引导您完成为 Amazon MWAA 环境创建从计算机到 Apache Airflow Web 服务器的 VPN 隧道的步骤。要通过 VPN 隧道连接到互联网，您首先需要创建一个 AWS Client VPN 终端节点。设置完成后，客户端 VPN 端点将充当 VPN 服务器，允许计算机与 VPC 中的资源进行安全连接。然后，您将使用桌面版AWS Client VPN 从电脑连接到客户端 VPN。

> **Note**
>
> 本教程适用于专用网络访问模式。如果您选择 “同时访问公用和私有网络”，则不需要 VPN 即可访问 Apache Airflow 用户界面，因为它可以通过互联网进行访问。

Sections

- 步骤 6：为客户端 VPN 添加授权入口规则
- 步骤 7：下载客户端 VPN 端点配置文件
- 第八步：Connect 到 AWS Client VPN
- 接下来做什么？

私有网络

本教程假设您已为 Apache Airflow Web 服务器选择了私有网络访问模式。

私有网络访问模式将访问 Apache Airflow UI 的权限限制为 Amazon VPC 中已获准访问环境 IAM 策略的用户。

创建具有私有网络访问权限的环境时，必须将所有依赖项打包到 Python 轮档案 (`.whl`) 中，然后在`.whl`中引用`requirements.txt`。有关使用 Wheel 打包和安装依赖项的说明，请参阅使用 Python Wheel 管理依赖项。

下图描述了在 Amazon MWAA 控制台上哪里可以找到私有网络选项。

使用案例

您可以在创建 Amazon MWAA 环境之前或之后使用本教程。您必须使用与您的环境相同的 Amazon VPC、VPC 安全组和私有子网。如果您在创建 Amazon MWAA 环境后使用本教程，则在完成这些步骤后，您可以返回 Amazon MWAA 控制台并将 Apache Airflow Web 服务器访问模式更改为私有网络。

开始前的准备工作

1. 检查用户权限。请确保您在 AWS Identity and Access Management (IAM) 中的账户拥有足够的权限来创建和管理 VPC 资源。
2. 使用 Amazon MWAA VPC。本教程假设您正在将客户端 VPN 关联到现有 VPC。Amazon VPC 必须与 Amazon MWAA 环境 AWS 区域 相同，并且有两个私有子网。如果您尚未创建 Amazon VPC，请使用中的 CloudFormation 模板选项三：创建不可访问互联网的 Amazon VPC 网络。

目标

在本教程中，您将执行以下操作：

1. 使用现有 Amazon VPC 的 CloudFormation 模板创建 AWS Client VPN 终端节点。
2. 生成服务器和客户端证书和密钥，然后将服务器证书和密钥上传到 AWS Certificate Manager ，与 Amazon MWAA 环境 AWS 区域 相同。
3. 为客户端 VPN 下载并修改客户端 VPN 端点配置文件，然后使用该文件创建 VPN 配置文件，以便使用桌面版客户端 VPN 进行连接。

（可选）步骤 1：确定 VPC、CIDR 规则和 VPC 安全

下一节介绍如何为 Amazon VPC、VPC 安全组查找 ID，以及在后续步骤中识别创建客户端 VPN 所需的 CIDR 规则的方法。

确定 CIDR 规则

下一节介绍如何识别 CIDR 规则，您需要使用这些规则来创建客户端 VPN。

识别客户端 VPN 的 CIDR

1. 在 Amazon VPC 控制台，打开 Amazon VPC 页面。
2. 使用导航栏中的区域选择器选择与 Amazon MWAA 环境 AWS 区域 相同的区域。
3. 选择 Amazon VPC。
4. 假设私有子网的 CIDR 是：

   - 私有子网 1：10.192.10.0/24
   - 私有子网 2：10.192.11.0/24

   如果 Amazon VPC 的 CIDR 是 10.192.0.0/16，那么您为客户端 VPN 指定的客户端 IPv4 CIDR 将是 10.192.0.0/22。
5. 保存此 CIDR 值以及 VPC ID 的值以供后续步骤使用。

识别 VPC 和安全组

下一节介绍如何查找创建客户端 VPC 所需的 Amazon VPC 和安全组的 ID。

> **Note**
>
> 您可能正在使用多个安全组。在后续步骤中，您需要指定所有 VPC 的安全组。

识别安全组

1. 在 Amazon VPC 控制台打开安全组页面。
2. 在导航栏中，使用区域选择器来选择 AWS 区域。
3. 在 VPC ID 中搜索 Amazon VPC，然后识别与 VPC 关联的安全组。
4. 保存安全组和 VPC 的 ID，以供后续步骤使用。

步骤 2：创建服务器证书和客户端证书

客户端 VPN 端点仅支持 1024 位和 2048 位 RSA 密钥大小。以下部分介绍如何使用 OpenVPN easy-rsa 生成服务器和客户端的证书和密钥，然后使用 () 将证书上传到 ACM。 AWS Command Line Interface AWS CLI

创建客户端证书

1. 按照以下快速步骤，通过客户端身份验证和授权：相互身份验证 AWS CLI中创建证书并将其上传到 ACM。
2. 在这些步骤中，您必须在上传服务器和客户端 AWS 区域 证书时在 AWS CLI 命令中指定与 Amazon MWAA 环境相同的内容。以下是有关如何在这些命令中指定区域的一些示例：

   a. Example服务器证书的区域

   ```
   aws acm import-certificate --certificate fileb://server.crt --private-key
    fileb://server.key --certificate-chain fileb://ca.crt --region us-west-2
   ```

   b. Example客户端证书的区域

   ```
   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. 完成这些步骤后，保存服务器证书和客户端证书 ARN 的 AWS CLI 响应中返回的值。您将在 CloudFormation 模板中指定这些 ARN 来创建 Client VPN。
3. 在这些步骤中，客户端证书和私钥将保存到计算机中。以下是有关在哪里可以找到这些凭证的示例：

a. Example在 macOS 上

在 macOS 上，内容保存在 /Users/*your-user*/custom_folder。如果您列出此目录的所有 (ls -a) 内容，则会看到类似以下的内容：

```
.
..
ca.crt
client1.domain.tld.crt
client1.domain.tld.key
server.crt
server.key
```

b. 完成这些步骤后，保存内容或在 client1.domain.tld.crt 中记下客户端证书的位置，以及在 client1.domain.tld.key 中记下私钥的位置。您将把这些值添加到客户端 VPN 的配置文件中。

第三步：保存 CloudFormation 本地模板

下一节包含创建 Client VPN 的 CloudFormation 模板。您必须指定与 Amazon MWAA 环境相同的 Amazon VPC、VPC 安全组和私有子网。

- 复制以下模板的内容并将其作为 mwaa_vpn_client.yaml 保存在本地中。您也可以使用下载模板。

  替换以下值：

  - **YOUR_CLIENT_ROOT_CERTIFICATE_ARN**— 在 ClientRootCertificateChainArn 中的 client1.domain.tld 证书的 ARN。
  - **YOUR_SERVER_CERTIFICATE_ARN**— 在 ServerCertificateArn 中的服务器证书的 ARN。
  - 在 ClientCidrBlock 中的客户端 IPv4 CIDR 规则。提供 10.192.0.0/22 的 CIDR 规则。
  - Amazon VPC ID，如 VpcId 所示。提供 vpc-010101010101 的 VPC。
  - VPC 安全组 ID，如 SecurityGroupIds 所示。提供了 sg-0101010101 的安全组。

```
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

如果您在环境中使用多个安全组，则可以按以下格式指定多个安全组：

```
SecurityGroupIds:
 - sg-0112233445566778b
 - sg-0223344556677889f
```

第四步：创建 Client VPN CloudFormation 堆栈

要创建 AWS Client VPN

1. 打开 AWS CloudFormation 控制台。
2. 选择模板已准备就绪，然后选择上传模板文件。
3. 选择选择文件，然后选择 `mwaa_vpn_client.yaml` 文件。
4. 选择下一步、下一步。
5. 选择堆栈，然后选择创建堆栈。

步骤 5：将子网关联到客户端 VPN

将私有子网关联到 AWS Client VPN

1. 打开 Amazon VPC 控制台。
2. 选择客户端 VPN 端点页面。
3. 选择客户端 VPN，然后选择关联选项卡、关联。
4. 在下拉列表中选择以下内容：
   - Amazon VPC，如 VPC 所示。
   - 在选择要关联的子网中的一个私有子网。
5. 选择关联 。

> **Note**
>
> VPC 和子网关联到客户端 VPN 需要几分钟时间。

步骤 6：为客户端 VPN 添加授权入口规则

您需要使用 VPC 的 CIDR 规则向客户端 VPN 添加授权入口规则。如果您想要授权 Active Directory 组或 SAML-based 身份提供商 (IdP) 中的特定用户或组，请参阅 C lient VPN 指南中的授权规则。

要将 CIDR 添加到 AWS Client VPN

1. 打开 Amazon VPC 控制台。

2. 选择客户端 VPN 端点页面。
3. 选择客户端 VPN，然后选择授权选项卡、授权入口。
4. 指定以下内容：
   - 要在目标网络中启用 Amazon VPC 的 CIDR 规则。例如：

     ```
     10.192.0.0/16
     ```
   - 在授予访问权限中，选择允许所有用户访问。
   - 在描述中，输入一个描述性名称。
5. 选择添加授权规则。

> **Note**
>
> 根据 Amazon VPC 的联网组件，您可能还需要将此授权入口规则添加到网络访问控制列表 (NACL) 中。

步骤 7：下载客户端 VPN 端点配置文件

下载配置文件

1. 按照以下快速步骤在下载客户端 VPN 端点配置文件中下载客户端 VPN 配置文件。
2. 在这些步骤中，系统会要求您在客户端 VPN 端点 DNS 名称前面加一个字符串。示例如下：
   - Example端点 DNS 名称

     如果客户端 VPN 端点 DNS 名称为：

     ```
     remote cvpn-endpoint-0909091212aaee1.prod.clientvpn.us-west-1.amazonaws.com 443
     ```

     您可以添加一个字符串来识别客户端 VPN 端点，如下所示：

     ```
     remote mwaavpn.cvpn-endpoint-0909091212aaee1.prod.clientvpn.us-
     west-1.amazonaws.com 443
     ```
3. 在这些步骤中，系统会要求您在一组新 <cert></cert> 标签之间添加客户端证书的内容，而在一组新 <key></key> 标签之间添加私有密钥的内容。示例如下：

a. 打开命令提示符并将目录更改为客户端证书和私钥的位置。

b. Example macOS client1.domain.tld.crt

要在 macOS 上显示 `client1.domain.tld.crt` 文件内容，您可以使用 `cat client1.domain.tld.crt`。

从终端复制值并粘贴在 `downloaded-client-config.ovpn` 中，如下所示：

```
ZZZ1111dddaBBB
-----END CERTIFICATE-----
</ca>
<cert>
-----BEGIN CERTIFICATE-----
YOUR client1.domain.tld.crt
-----END CERTIFICATE-----
</cert>
```

c. Example macOS client1.domain.tld.key

要显示 `client1.domain.tld.key` 文件内容，您可以使用 `cat client1.domain.tld.key`。

从终端复制值并粘贴在 `downloaded-client-config.ovpn` 中，如下所示：

```
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>
```

第八步：Connect 到 AWS Client VPN

的客户端 AWS Client VPN 是免费提供的。您可以将计算机直接连接到，以 AWS Client VPN 获得端到端 VPN 体验。

连接到客户端 VPN

1. 下载并安装桌面版AWS Client VPN。
2. 打开 AWS Client VPN.
3. 在 VPN 客户端菜单中选择文件、托管配置文件。
4. 选择添加配置文件，然后选择 `downloaded-client-config.ovpn`。
5. 在显示名称中输入描述性名称。
6. 选择添加配置文件，完成。
7. 选择连接。

连接到客户端 VPN 后，您需要断开与其他 VPN 的连接才能访问 Amazon VPC 中的任何资源。

> **Note**
>
> 您可能需要退出客户端，然后重新开始，然后才能建立连接。

接下来做什么？

- 要了解如何创建 Amazon MWAA 环境，请参阅 开始使用 Amazon MWAA。您必须创建与 Client VPN AWS 区域 相同的环境，并使用与 Client VPN 相同的 VPC、私有子网和安全组。

教程：使用 Linux 堡垒主机配置私有网络访问权限

本教程将引导您完成为 Amazon MWAA 环境创建从计算机到 Apache Airflow Web 服务器的 SSH 隧道的步骤。本教程假设您已经创建了 Amazon MWAA 环境。设置完成后，Linux 堡垒主机将充当跳转服务器，允许计算机与 VPC 中的资源进行安全连接。然后，您将使用 SOCKS 代理管理附加组件来控制浏览器中的代理设置，以访问 Apache Airflow UI。

> **Note**
>
> 本教程适用于专用网络访问模式。如果您选择 “同时访问公用和私有网络”，则无需堡垒主机即可访问 Apache Airflow 用户界面，因为它可以通过互联网进行访问。

Sections

私有网络

本教程假设您已为 Apache Airflow Web 服务器选择了私有网络访问模式。

私有网络访问模式将访问 Apache Airflow UI 的权限限制为 Amazon VPC 中已获准访问环境 IAM 策略的用户。

创建具有私有网络访问权限的环境时，必须将所有依赖项打包到 Python 轮档案 (`.whl`) 中，然后在`.whl`中引用`requirements.txt`。有关使用 Wheel 打包和安装依赖项的说明，请参阅使用 Python Wheel 管理依赖项。

下图描述了在 Amazon MWAA 控制台上哪里可以找到私有网络选项。

使用案例

您可以在创建 Amazon MWAA 环境后使用本教程。您必须使用与环境相同的 Amazon VPC、VPC 安全组和公有子网。

开始前的准备工作

1. 检查用户权限。请确保您在 AWS Identity and Access Management (IAM) 中的账户拥有足够的权限来创建和管理 VPC 资源。
2. 使用 Amazon MWAA VPC。本教程假定您将堡垒主机关联到 VPC。Amazon VPC 必须与 Amazon MWAA 环境位于同一区域，并且拥有两个私有子网，如 创建 VPC 网络 中所定义。
3. 创建 SSH 密钥。您需要在与 Amazon MWAA 环境相同的区域创建 Amazon EC2 SSH 密钥（.pem）才能连接到虚拟服务器。如果您没有 SSH 密钥，请参阅《Amazon EC2 用户指南》中的创建或导入密钥对。

目标

在本教程中，您将执行以下操作：

1. 使用现有 VPC 的CloudFormation 模板创建 Linux 堡垒主机实例。
2. 使用端口 22 上的入口规则授权进入堡垒机实例安全组的入站流量。
3. 授权从 Amazon MWAA 环境的安全组流向堡垒机实例的安全组的入站流量。
4. 创建通往堡垒机实例的 SSH 隧道。
5. 安装并配置 Firefox 浏览器的 FoxyProxy 插件以访问 Apache Airflow 用户界面。

步骤 1：创建堡垒机实例

以下部分介绍在 CloudFormation 控制台上使用现有 VPC 的CloudFormation 模板创建 linux 堡垒实例的步骤。

创建 Linux 堡垒主机

1. 在 CloudFormation 控制台上打开 “部署快速入门” 页面。
2. 使用导航栏中的区域选择器选择与您的 Amazon MWAA 环境 AWS 区域 相同的区域。
3. 选择下一步。

4. 在堆栈名称文本字段中输入名称，例如 `mwaa-linux-bastion`。
5. 在参数、网络配置窗格上，选择以下选项：
   a. 选择 Amazon MWAA 环境的 VPC ID。
   b. 选择 Amazon MWAA 环境的公有子网 1 ID。
   c. 选择 Amazon MWAA 环境的公有子网 2 ID。
   d. 在允许的堡垒机外部访问 CIDR 中输入尽可能窄的地址范围（例如，内部 CIDR 范围）。

> **Note**
>
> 识别范围的最简单方法是使用与公有子网相同的 CIDR 范围。例如，创建 VPC 网络页面 CloudFormation 模板中的公用子网为`10.192.10.0/24`和。`10.192.11.0/24`

6. 在 Amazon EC2 配置窗格上，选择以下选项：
   a. 在密钥对名称的下拉列表中选择 SSH 密钥。
   b. 在堡垒主机名中输入名称。
   c. 对于 TCP 转发，选择 true。

> **Warning**
>
> 在此步骤中，必须将 TCP 转发设置为 true。否则，您将无法在下一步创建 SSH 隧道。

7. 选择下一步、下一步。
8. 选择堆栈，然后选择创建堆栈。

要了解有关 Linux 堡垒主机架构的更多信息，请参阅AWS 云上的 Linux 堡垒主机：架构。

步骤 2：创建 SSH 隧道

以下步骤介绍如何创建通往 Linux 堡垒机的 SSH 隧道。SSH 隧道接收从本地 IP 地址发送到 Linux 堡垒机的请求，这就是为何在前述步骤中将 Linux 堡垒机的 TCP 转发设置为 `true`。

macOS/Linux

使用命令行创建隧道

1. 打开 Amazon EC2 控制台的实例页面。
2. 选择一个实例。
3. 复制公有 IPv4 DNS下的 IP 地址。例如 `ec2-4-82-142-1.compute-1.amazonaws.com`。
4. 在命令提示符下，导航到存储 SSH 密钥的目录。
5. 运行以下命令以使用 SSH 连接堡垒机实例。将示例值替换为 `mykeypair.pem` 中的 SSH 密钥名称。

```
ssh -i mykeypair.pem -N -D 8157 ec2-user@YOUR_PUBLIC_IPV4_DNS
```

Windows (PuTTY)

使用 PuTTTY 创建隧道

1. 打开 Amazon EC2 控制台的实例页面。
2. 选择一个实例。
3. 复制公有 IPv4 DNS下的 IP 地址。例如 `ec2-4-82-142-1.compute-1.amazonaws.com`。
4. 打开 Putty，选择会话。
5. 在“主机名”中输入主机名为 ec2-user@***YOUR_PUBLIC_IPV4_DNS***，将端口输入为。 22
6. 展开 SSH 选项卡，选择身份验证。在用于身份验证的私钥文件中，选择本地“ppk”文件。
7. 在 SSH 下，选择隧道选项卡，然后选择动态和自动选项。
8. 在源端口中，添加 8157 端口（或任何其他未使用的端口），然后将目标端口留空。选择添加。
9. 选择会话选项卡并输入会话名称。例如 `SSH Tunnel`。
10. 选择保存，打开。

> **Note**
>
> 您可能需要为公有密钥输入密码短语。

> **Note**
>
> 如果您收到`Permission denied (publickey)`错误，我们建议您使用该AWSSupport-TroubleshootSSH工具，然后选择“运行此自动化（控制台）”来排除您的 SSH 设置故障。

步骤 3：将堡垒机安全组配置为入站规则

通过与这些服务器关联的特殊维护安全组，允许从服务器访问服务器和定期访问互联网。以下步骤介绍如何将堡垒机安全组配置为某环境的 VPC 安全组的入站流量来源。

1. 在 Amazon MWAA 控制台上打开环境页面。
2. 选择环境。
3. 在联网窗格上，选择 VPC 安全组。
4. 选择编辑入站规则。
5. 选择添加规则。
6. 在源下拉列表中选择 VPC 安全组 ID。
7. 将其余选项留空，或将其设置为默认值。
8. 选择保存规则。

步骤 4：复制 Apache Airflow URL

以下步骤介绍如何打开 Amazon MWAA 控制台并将 URL 复制到 Apache Airflow UI。

1. 在 Amazon MWAA 控制台上打开环境页面。
2. 选择环境。
3. 在 Airflow UI中复制 URL 以执行后续步骤。

步骤 5：配置代理设置

如果您使用带有动态端口转发的 SSH 隧道，则必须使用 SOCKS 代理管理附加组件来控制浏览器中的代理设置。例如，你可以使用 Chromium 的`--proxy-server`功能来启动浏览器会话，或者在 Mozilla FireFox 浏览器中使用该 FoxyProxy 扩展程序。

选项一：使用本地端口转发设置 SSH 隧道

如果您不想使用 SOCKS 代理，您可以使用本地端口转发设置 SSH 隧道。以下示例命令通过转发本地端口 8157 上的流量来访问 Amazon EC2 ResourceManager网页界面。

1. 打开新的命令提示符窗口。
2. 输入以下命令以打开 SSH 隧道。

```
ssh -i mykeypair.pem -N -L 8157:YOUR_VPC_ENDPOINT_ID-vpce.us-
east-1.airflow.amazonaws.com:443 ubuntu@YOUR_PUBLIC_IPV4_DNS.us-
east-1.compute.amazonaws.com
```

   -L 代表使用本地端口转发，由此，您就能指定一个本地端口，用于将数据转发到节点的本地 Web 服务器上标识的远程端口。

3. 在浏览器中输入 `http://localhost:8157/`。

> **Note**
>
> 您可能需要使用 `https://localhost:8157/`。

选项二：使用命令行配置代理

您可以使用大多数 Web 浏览器，通过命令行或配置参数来配置代理。例如，使用 Chromium，您可以在 Chromium 中通过以下命令启动浏览器：

```
chromium --proxy-server="socks5://localhost:8157"
```

这将启动浏览器会话，该会话使用您在前述步骤中创建的 SSH 隧道来代理其请求。您可以按如下方式打开 Amazon MWAA 私有环境 URL（使用 https:// ）：

```
https://YOUR_VPC_ENDPOINT_ID-vpce.us-east-1.airflow.amazonaws.com/home.
```

选项三：用 FoxyProxy 于 Mozilla Firefox 的代理

以下示例演示了 Mozilla Firefox 的 FoxyProxy 标准（版本 7.5.1）配置。 FoxyProxy 提供了一组代理管理工具。该工具使您可以使用与 Apache Airflow UI 所使用的域对应模式相匹配的 URL 代理服务器。

1. 在 Firefox 中，打开FoxyProxy 标准扩展页面。
2. 选择添加到 Firefox。
3. 选择添加。
4. 选择浏览器工具栏中的 FoxyProxy 图标，然后选择选项。
5. 复制以下代码并在本地另存为 mwaa-proxy.json。用你的 Apache Air ***YOUR_HOST_NAME*** flow 网址替换中的示例值。

```
{
  "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. 在“从 FoxyProxy 6.0 及以上版本导入设置”窗格中，选择“导入设置”，然后选择文件。`mwaa-proxy.json`
7. 选择确定。

步骤 6：打开 Apache Airflow UI

以下步骤介绍如何打开 Apache Airflow UI。

1. 在 Amazon MWAA 控制台上打开环境页面。
2. 选择打开 Airflow UI。

接下来做什么？

- 要了解如何在 baol 通往堡垒主机的 SSH 隧道上运行 Airflow CLI 命令，请参阅 Apache Airflow CLI 命令参考。
- 要了解如何将 DAG 代码上传到 Amazon S3 存储桶，请参阅 添加或更新 DAG。

教程：限制 Amazon MWAA 用户访问其中的一个子集 DAGs

Amazon MWAA 通过将 IAM 主体映射到一个或多个 Apache Airflow 的默认角色来管理对环境的访问权限。使用以下教程限制个人 Amazon MWAA 用户只能访问特定的 DAG 或一组并与之交互。 DAGs

> **Note**
>
> 只要可以担任 IAM 角色，就可以使用联合访问完成本教程中的步骤。

主题

先决条件

要完成本教程中的步骤，您需要做好以下准备：

- 具有多个 Amazon MWAA 环境 DAGs
- `Admin`具有AdministratorAccess权限的 IAM 委托人和可以限制 DAG 访问权限的 IAM 用户作为委托人。`MWAAUser`有关管理员角色的更多信息，请参阅《IAM 用户指南》中的管理员任务函数。

  > **Note**
  >
  > 请勿将权限策略直接附加到 IAM 用户。我们建议设置用户可以代入的 IAM 角色来临时访问 Amazon MWAA 资源。

- AWS Command Line Interface 已安装@@ 版本 2。

步骤 1：使用默认 **Public** Apache Airflow 角色向 IAM 主体提供 Amazon MWAA Web 服务器访问权限。

要授予权限，请使用 AWS 管理控制台

1. 使用Admin角色登录您的 AWS 账户 ，然后打开 IAM 控制台。
2. 在左侧导航窗格中，选择用户，然后从用户表中选择 Amazon MWAA IAM 用户。
3. 在用户详细信息页面的摘要下，选择权限选项卡，然后选择权限策略以展开卡片并选择添加权限。
4. 在设置权限部分中，选择直接附加现有策略，然后选择创建策略。
5. 在创建策略页面上，选择 JSON，然后将以下 JSON 权限策略复制并粘贴到策略编辑器中。该策略向具有默认 Public Apache Airflow 角色的用户授予 Web 服务器访问权限。

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "airflow:CreateWebLoginToken",
            "Resource": [
            "arn:aws:airflow:us-
east-1:111122223333:role/YOUR_ENVIRONMENT_NAME/Public"
            ]
        }
    ]
}
```

步骤 2：创建新的 Apache Airflow 自定义角色

使用 Apache Airflow UI 创建新角色

1. 使用管理员 IAM 角色，打开 Amazon MWAA 控制台并启动环境的 Apache Airflow UI。
2. 在顶部的导航窗格中，将鼠标悬停在安全上以打开下拉列表，然后选择列出角色以访问默认的 Apache Airflow 角色。
3. 从角色列表中选择用户，然后在页面顶部选择操作以打开下拉列表。选择复制角色，然后确认确定

> **Note**
>
> 复制操作或查看者角色以分别授予或多或少的访问权限。

4. 在表格中找到您创建的新角色，然后选择编辑记录。
5. 在切换角色页面上，执行以下操作：
   - 对于名称，在文本字段中输入角色的新名称。例如 **Restricted**。
   - 要查看权限列表，请移除`can read on DAGs`和`can edit on DAGs`，然后为 DAGs 要提供访问权限的集合添加读写权限。例如，对于 DAG `example_dag.py`，添加 **can read on DAG:*example_dag*** 和 **can edit on DAG:*example_dag***。

   选择保存。现在，您有了新的角色，该角色将访问权限限制为 Amazon MWAA 环境中 DAGs 可用的部分内容。您可以将此角色分配给任何现有的 Apache Airflow 用户。

步骤 3：将您创建的角色分配给 Amazon MWAA 用户

分配新角色

1. 使用 `MWAAUser` 的访问凭证，运行以下 CLI 命令来检索环境的 Web 服务器 URL。

   ```
   aws mwaa get-environment --name YOUR_ENVIRONMENT_NAME | jq
    '.Environment.WebserverUrl'
   ```

   如果成功，您将看到以下输出内容：

   ```
   "ab1b2345-678a-90a1-a2aa-34a567a8a901.c13.us-west-2.airflow.amazonaws.com"
   ```

2. `MWAAUser`登录后 AWS 管理控制台，打开新的浏览器窗口并访问以下内容 URI。将 `Webserver-URL` 替换为您的信息。

   ```
   https://<Webserver-URL>/home
   ```

   如果成功，您将收到一个 `Forbidden` 错误页面，因为 `MWAAUser` 尚未获得访问 Apache Airflow UI 的权限。

3. Admin登录后 AWS 管理控制台，再次打开亚马逊 MWAA 控制台并启动环境的 Apache Airflow 用户界面。
4. 在 UI 控制面板中，展开安全下拉列表，这次选择列出用户。
5. 在用户表中，找到新的 Apache Airflow 用户并选择编辑记录。用户的名字将按以下模式匹配 IAM 用户名：user/*mwaa-user*。
6. 在编辑用户页面的角色部分，添加您创建的新自定义角色，然后选择保存。

> **Note**
>
> 姓氏字段是必填字段，但空格可以满足要求。

IAM Public 委托人授予访问 Apache Airflow 用户界面的MWAAUser权限，而新角色则提供获取用户界面所需的额外权限。 DAGs

> **Important**
>
> 使用 Apache Airflow UI 添加的 5 个未经 IAM 授权的默认角色（例如 Admin）中的任何一个都将在用户下次登录时移除。

后续步骤

- 要了解有关管理 Amazon MWAA 环境访问权限的更多信息，并获得可供环境用户使用的 JSON IAM 策略示例，请参阅 the section called “访问 Amazon MWAA 环境”。

相关资源

- 访问控制（Apache Airflow 文档）— 在 Apache Airflow 文档网站上了解有关默认 Apache Airflow 角色的更多信息。

教程：在 Amazon MWAA 上自动管理您自己的环境端点

如果您使用 AWS Organizations 管理共享资源的多个 AWS 账户，则可借助 Amazon MWAA 来创建和管理自己的 Amazon VPC 端点。这意味着您可以使用更严格的安全策略，仅允许访问您的环境所需的资源。

在共享 Amazon VPC 中创建环境时，拥有主要 Amazon VPC 的账户（所有者）将与属于同一组织的其他账户（参与者）共享 Amazon MWAA 所需的两个私有子网。然后，共享这些子网的参与者账户可以查看、创建、修改和删除共享 VPC 中的环境。

在共享的存在其他策略限制的 Amazon VPC 中创建环境时，Amazon MWAA 将首先创建服务 VPC 资源，然后进入 `PENDING` 状态最长 72 小时。

当环境状态从 `CREATING` 变为 `PENDING` 时，Amazon MWAA 会发送一条 Amazon EventBridge 状态变更通知。这让所有者账户能够根据端点服务信息，从 Amazon MWAA 控制台或 API 或通过编程方式创建所需的端点。在下例中，我们将使用一个 Lambda 函数和一条侦听 Amazon MWAA 状态变更通知的 EventBridge 规则，创建新的 Amazon VPC 端点。

在此例中，我们将在与环境相同的 Amazon VPC 中创建新的端点。要设置共享 Amazon VPC，请在所有者账户中创建 EventBridge 规则和 Lambda 函数，并在参与者账户中创建 Amazon MWAA 环境。

主题

先决条件

要完成本教程的步骤，您需要做好以下准备：

- ...

创建 Amazon VPC

使用以下 CloudFormation 模板和 AWS CLI 命令创建一个新 Amazon VPC。该模板会设置 Amazon VPC 资源并修改端点策略以限制对特定队列的访问。

1. 下载 CloudFormation 模板，然后解压缩 .yml 文件。
2. 在新的命令提示符窗口中，导航到保存模板的文件夹，然后使用 create-stack 创建堆栈。--template-body 标志用于指定模板的路径。

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

在下一部分中，您将创建 Lambda 函数。

创建 Lambda 函数

使用以下 Python 代码和 IAM JSON 策略创建新的 Lambda 函数和执行角色。该函数将为一个私有 Apache Airflow Web 服务器和 Amazon SQS 队列创建 Amazon VPC 端点。在扩展环境时，Amazon MWAA 会使用 Amazon SQS 在多个 Worker 节点之间进行 Celery 任务排队。

1. 下载 Python 函数代码。
2. 下载 IAM 权限策略，然后解压缩该文件。
3. 打开命令提示符，然后导航到保存 JSON 权限策略的文件夹。使用 IAM create-role 命令创建该新角色。

```
aws iam create-role --role-name function-role \
      --assume-role-policy-document file://lambda-mwaa-vpce-policy.json
```

记下 AWS CLI 响应中的角色 ARN。在下一步中，我们将使用新角色的 ARN 将其指定为函数的执行角色。

4. 导航到保存函数代码的文件夹，然后使用 create-function 命令创建新函数。

```
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
```

记下 AWS CLI 响应中的函数 ARN。在下一步中，我们将指定该 ARN，以将该函数配置为新 EventBridge 规则的目标。

在下一部分中，您将创建 EventBridge 规则，此规则在环境进入 PENDING 状态时调用该函数。

创建 EventBridge 规则

完成以下步骤，创建一条用于侦听 Amazon MWAA 通知并以您的新 Lambda 函数为目标的新规则。

1. 使用 EventBridge `put-rule` 命令创建新的 EventBridge 规则。

```
aws events put-rule --name "mwaa-lambda-rule" \
--event-pattern "{\"source\":[\"aws.airflow\"],\"detail-type\":[\"MWAA Environment
 Status Change\"]}"
```

事件模式用于侦听 Amazon MWAA 每次在环境状态发生变化时发送的通知。

```
{
     "source": ["aws.airflow"],
     "detail-type": ["MWAA Environment Status Change"]
     }
```

2. 使用 `put-targets` 命令将该 Lambda 函数添加为新规则的目标。

```
aws events put-targets --rule "mwaa-lambda-rule" \
--targets "Id"="1","Arn"="arn:aws:lambda:us-east-1:123456789012:function:mwaa-vpce-
lambda"
```

您已准备就绪，可以使用客户管理型 Amazon VPC 端点创建新的 Amazon MWAA 环境。

创建 Amazon MWAA 环境

通过 Amazon MWAA 控制台使用客户管理型 Amazon VPC 端点创建新的 Amazon MWAA 环境。

1. 打开 Amazon MWAA 控制台并选择创建环境。
2. 对于名称，输入一个唯一名称。
3. 对于 Airflow 版本，请选择最新版本。
4. 选择要用于该环境的 Amazon S3 存储桶和 DAG 文件夹（例如 `dags/`），然后选择下一步。
5. 在配置高级设置页面上，执行以下操作：
   a. 对于虚拟私有云，选择您在上一步中创建的 Amazon VPC。
   b. 对于 Web 服务器访问，请选择公共网络（可访问互联网）。

c. 对于安全组，请选择您使用 CloudFormation 创建的安全组。由于之前步骤中 AWS PrivateLink 端点的安全组是自引用的，因此必须为您的环境选择同一安全组。

d. 对于端点管理，请选择客户管理型端点。

6. 请保留其余的默认设置，然后选择下一步。

7. 检查您的选择，然后选择创建环境。

> **Tip**
>
> 有关设置新环境的更多信息，请参阅 Amazon MWAA 入门。

环境处于 `PENDING` 状态时，Amazon MWAA 会发送与为您的规则设置的事件模式相匹配的通知。该规则会调用您的 Lambda 函数。该函数将解析通知事件并获取 Web 服务器和 Amazon SQS 队列所需的端点信息，然后在您的 Amazon VPC 中创建端点。

当端点可用时，Amazon MWAA 会继续创建环境。准备就绪后，环境状态将变为 `AVAILABLE`，您可以使用 Amazon MWAA 控制台访问 Apache Airflow Web 服务器。

Amazon MWAA 的代码示例

本指南包含可在适用于 Apache Airflow 的亚马逊托管工作流程环境中使用的代码示例，包括 DAGs 自定义插件。有关将 Apache Airflow AWS 用于服务的更多示例，请参阅 Apache Airflow 存储库中的目录。dags GitHub

样本

- 使用 DAG 在 CLI 中导入变量
- 使用 SSHOperator 创建 SSH 连接
- 使用 AWS Secrets Manager 中的密钥进行 Apache Airflow Snowflake 连接
- 使用 DAG 在 CloudWatch 中编写自定义指标
- 在 Amazon MWAA 环境中清理 Aurora PostgreSQL 数据库
- 将环境元数据导出到 Amazon S3 上的 CSV 文件
- 为 Apache Airflow 变量使用 AWS Secrets Manager 中的密钥
- 使用 AWS Secrets Manager 中的密钥进行 Apache Airflow 连接
- 使用 Oracle 创建自定义插件
- 在 Amazon MWAA 上更改 DAG 的时区
- 刷新 CodeArtifact 令牌
- 使用 Apache Hive 和 Hadoop 创建自定义插件
- 为 Apache Airflow PythonVirtualenvOperator 创建自定义插件
- 使用 Lambda DAGs 函数进行调用
- DAGs 在不同的 Amazon MWAA 环境中调用
- 将 Amazon RDS for Microsoft SQL Server 与 Amazon MWAA 一起使用
- 将 Amazon MWAA 与 Amazon EKS 一起使用
- 使用 ECSOperator 连接 Amazon ECS
- 在 Amazon MWAA 中使用 dbt
- AWS 博客和教程

使用 DAG 在 CLI 中导入变量

以下示例代码会使用 Amazon MWAA 上的 CLI 导入变量。

主题

版本

您可以在 Python 3.10 中将本页上的代码示例与 Apache Airflow v2 一起使用，在 Python 3.11 中与 Apache Airflow v3 一起使用。

先决条件

无需其他权限即可使用本页上的代码示例。

权限

AWS 账户 需要访问 `AmazonMWAAAirflowCliAccess` 策略。要了解更多信息，请参阅 Apache Airflow CLI 政策： AmazonMWAAAirflowCliAccess。

依赖项

要在 Apache Airflow v2 和更高版本中使用此代码示例，无需附加依赖项。使用 aws-mwaa-docker-images 安装 Apache Airflow。

代码示例

以下示例代码需要三个输入：Amazon MWAA 环境名称（在 `mwaa_env` 中）、环境 AWS 区域（在 `aws_region` 中）和包含要导入的变量的本地文件（在 `var_file` 中）。

```
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)
```

接下来做什么？

- 要了解如何将本示例中的 DAG 代码上传到 Amazon S3 存储桶的 dags 文件夹，请参阅 添加或更新 DAG。

使用 SSHOperator 创建 SSH 连接

以下示例介绍如何使用有向无环图（DAG）中的 SSHOperator 从 Amazon MWAA 环境连接到远程 Amazon EC2 实例。您可以使用类似的方法连接到任何具有 SSH 访问权限的远程实例。

在以下示例中，您将 SSH 密钥 (.pem) 上传到在 Amazon S3 上的环境的 dags 目录中。然后，您可以使用 requirements.txt 安装必要的依赖项，并在 UI 中创建新的 Apache Airflow 连接。最后，您编写一个 DAG 来创建与远程实例的 SSH 连接。

主题

- 版本
- 先决条件
- 权限
- 要求
- 将密钥复制到 Amazon S3

- 创建新的 Apache Airflow 连接
- 代码示例

版本

您可以在 Python 3.10 中将本页上的代码示例与 Apache Airflow v2 一起使用，在 Python 3.11 中与 Apache Airflow v3 一起使用。

先决条件

要使用本页上的示例代码，您需要以下内容：

- Amazon MWAA 环境。
- SSH 密钥。该代码示例假设您有 Amazon EC2 实例，并且与 Amazon MWAA 环境位于同一区域的 `.pem`。如果您没有密钥，请参阅《Amazon EC2 用户指南》中的创建或导入密钥对。

权限

无需其他权限即可使用本页上的代码示例。

要求

将以下参数添加到 `requirements.txt`，以将 `apache-airflow-providers-ssh` 程序包安装到 Web 服务器上。当环境更新并且 Amazon MWAA 成功安装依赖项后，您将在 UI 中获得新的 SSH 连接类型。

```
-c https://raw.githubusercontent.com/apache/airflow/constraints-Airflow-version/
constraints-Python-version.txt
apache-airflow-providers-ssh
```

> **Note**
>
> `-c` 定义了 `requirements.txt` 中的约束 URL。这样可以确保 Amazon MWAA 为环境安装了正确的程序包版本。

将密钥复制到 Amazon S3

使用以下 AWS Command Line Interface 命令将 .pem 密钥复制到 Amazon S3 中的环境 dags 目录中。

```
aws s3 cp your-secret-key.pem s3://amzn-s3-demo-bucket/dags/
```

Amazon MWAA 将包括 .pem 密钥在内的 dags 中的内容复制到本地 /usr/local/airflow/dags/ 目录，这样，Apache Airflow 就可以访问密钥了。

创建新的 Apache Airflow 连接

使用 Apache Airflow UI 创建新的 SSH 连接

1. 在 Amazon MWAA 控制台上打开环境页面。
2. 从环境列表中，为环境选择打开 Airflow UI。
3. 在 Apache Airflow UI 页面上，从主导航栏中选择管理员以展开下拉列表，然后选择连接。
4. 在列出连接页面上，选择 + 或添加新记录按钮以添加新连接。
5. 在连接到 AD 页面上，提供以下信息：

    a. 在连接名称中，输入 **ssh_new**。

    b. 在连接类型中，从下拉列表中选择 SSH。

    > **Note**
    >
    > 如果列表中没有 SSH 连接类型，则表示 Amazon MWAA 尚未安装所需的 apache-airflow-providers-ssh 程序包。请更新 requirements.txt 文件以包含此程序包，然后重试。

    c. 在主机中，请输入要连接的 Amazon EC2 实例的 IP 地址。例如 **12.345.67.89**。

    d. 在用户名中，请输入 **ec2-user**，以检查您是否正在连接到 Amazon EC2 实例。用户名可能会有所不同，具体取决于您希望 Apache Airflow 连接到的远程实例的类型。

    e. 在附加中，请以 JSON 格式输入以下键/值对：

    ```
    { "key_file": "/usr/local/airflow/dags/your-secret-key.pem" }
    ```

    此键值对指示 Apache Airflow 在本地 /dags 目录中搜索密钥。

代码示例

以下 DAG 使用 SSHOperator 连接到目标 Amazon EC2 实例，然后运行 hostname Linux 命令来打印该实例的名称。您可以修改 DAG 以在远程实例上运行任何命令或脚本。

1. 打开终端，导航到存储 DAG 代码的目录。例如：

```
cd dags
```

2. 复制以下代码示例的内容，并在本地另存为 ssh.py。

```
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. 运行以下 AWS CLI 命令将 DAG 复制到环境的存储桶，然后使用 Apache Airflow UI 触发 DAG。

```
aws s3 cp your-dag.py s3://your-environment-bucket/dags/
```

4. 如果成功，您将在 ssh_operator_example DAG 的任务日志中看到输出，类似于 ssh_task 的任务日志中的以下内容：

```
[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
```

使用 AWS Secrets Manager 中的密钥进行 Apache Airflow Snowflake 连接

以下示例调用 AWS Secrets Manager 在 Amazon MWAA 上获取 Apache Airflow Snowflake 连接的密钥。它假设您已完成 使用密钥配置 Apache Airflow 连接 AWS Secrets Manager 中的步骤。

主题

版本

您可以在 Python 3.10 中将本页上的代码示例与 Apache Airflow v2 一起使用，在 Python 3.11 中与 Apache Airflow v3 一起使用。

先决条件

要使用本页上的示例代码，您需要以下内容：

- 创建 Secrets Manager 后端作为 Apache Airflow 配置选项，如 使用密钥配置 Apache Airflow 连接 AWS Secrets Manager 所列。
- Secrets Manager 中的 Apache Airflow 连接字符串，如 使用密钥配置 Apache Airflow 连接 AWS Secrets Manager 所列。

权限

- Secrets Manager 权限，如 使用密钥配置 Apache Airflow 连接 AWS Secrets Manager 所列。

要求

要使用本页上的示例代码，请将以下依赖项添加到 `requirements.txt`。要了解更多信息，请参阅 安装 Python 依赖项。

```
apache-airflow-providers-snowflake==1.3.0
```

代码示例

以下步骤描述了如何创建 DAG 代码，以便调用 Secrets Manager 来获取密钥。

1. 在命令提示符下，导航到存储 DAG 代码的目录。例如：

   ```
   cd dags
   ```

2. 复制以下代码示例的内容，并在本地另存为 `snowflake_connection.py`。

   ```
   """
   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",
    )
```

接下来做什么？

- 要了解如何将本示例中的 DAG 代码上传到 Amazon S3 存储桶的 dags 文件夹，请参阅 添加或更新 DAG。

使用 DAG 在 CloudWatch 中编写自定义指标

您可以使用以下代码示例编写有向无环图（DAG），该图运行 PythonOperator 以检索 Amazon MWAA 环境的操作系统级指标。DAG 随后将数据作为自定义指标发布到 Amazon CloudWatch。

自定义操作系统级指标可让您进一步了解环境工作线程如何使用虚拟内存和 CPU 等资源。您可以使用此信息来选择最适合您的工作负载的环境类。

主题

- 版本
- 先决条件
- 权限
- 依赖项

- 代码示例

版本

您可以在 Python 3.10 中将本页上的代码示例与 Apache Airflow v2 一起使用，在 Python 3.11 中与 Apache Airflow v3 一起使用。

先决条件

要使用本页上的代码示例，您需要以下内容：

- Amazon MWAA 环境。

权限

无需其他权限即可使用本页上的代码示例。

依赖项

- 无需其他依赖项即可使用本页上的代码示例。

代码示例

1. 在命令提示符下，导航到存储 DAG 代码的文件夹。例如：

```
cd dags
```

2. 复制以下代码示例的内容并本地另存为 `dag-custom-metrics.py`。用环境名称替换 `MWAA-ENV-NAME`。

```
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. 运行以下 AWS CLI 命令将 DAG 复制到环境的存储桶，然后使用 Apache Airflow UI 触发 DAG。

```
aws s3 cp your-dag.py s3://your-environment-bucket/dags/
```

4. 如果 DAG 成功运行，您会在 Apache Airflow 日志中收到类似以下的内容：

```
[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
```

在 Amazon MWAA 环境中清理 Aurora PostgreSQL 数据库

Amazon Managed Workflows for Apache Airflow 使用 Aurora PostgreSQL 数据库作为 DAG 运行并存储任务实例的 Apache Airflow 元数据库。以下示例代码会定期为 Amazon MWAA 环境清除专用 Aurora PostgreSQL 数据库中的条目。

主题

版本

本页上的代码示例特定于亚马逊 MWAA 支持的 Apache Airflow v2 和 v3。请参阅支持的 Apache Airflow 版本。

先决条件

要使用本页上的示例代码，您需要以下内容：

- Amazon MWAA 环境。

依赖项

要在 Apache Airflow v2 中使用此代码示例，无需附加依赖项。使用 aws-mwaa-docker-images 安装 Apache Airflow。

代码示例

以下 DAG 会清理 TABLES_TO_CLEAN 中指定表的元数据数据库。该示例将删除指定表中存在超过 30 天的数据。要调整删除条目的存续时间，请将 MAX_AGE_IN_DAYS 设置为其他值。

Apache Airflow v3.0.6 to 3.2.1

```
from datetime import datetime
from airflow import DAG
from airflow.providers.standard.operators.bash import BashOperator

# Note: Database commands might time out if running longer than 5 minutes. If this
 occurs, please increase the MAX_AGE_IN_DAYS (or change
# timestamp parameter to an earlier date) for initial runs, then reduce on
 subsequent runs until the desired retention is met.

MAX_AGE_IN_DAYS = 30

# To clean specific tables, please provide a comma-separated list per
# https://airflow.apache.org/docs/apache-airflow/stable/cli-and-env-variables-
ref.html#clean

# A value of None will clean all tables
TABLES_TO_CLEAN = None

with DAG(
    dag_id="clean_db_dag",
    schedule=None,
    catchup=False,
    start_date=datetime(2026, 1, 1),
) as dag:
```

```
    tables_flag = f"--tables '{TABLES_TO_CLEAN}' " if TABLES_TO_CLEAN else ""

    bash_command = (
        f"TIMESTAMP=$(date -u -d '{MAX_AGE_IN_DAYS} days ago' '+%Y-%m-%d %H:%M:%S'
 2>/dev/null "
        f"|| date -u -v-{MAX_AGE_IN_DAYS}d '+%Y-%m-%d %H:%M:%S') && "
        "echo \"Cleaning records before: $TIMESTAMP\" && "
        "airflow db clean "
        "--clean-before-timestamp \"$TIMESTAMP\" "
        f"{tables_flag}"
        "--skip-archive --yes"
    )

    cli_command = BashOperator(
        task_id="bash_command",
        bash_command=bash_command,
    )
```

Apache Airflow v2.7.2 to 2.11.0

```
from airflow import DAG
from airflow.models.param import Param
from airflow.operators.bash_operator import BashOperator
from airflow.utils.dates import days_ago

from datetime import datetime, timedelta

# Note: Database commands might time out if running longer than 5 minutes. If this
 occurs, please increase the MAX_AGE_IN_DAYS (or change
# timestamp parameter to an earlier date) for initial runs, then reduce on
 subsequent runs until the desired retention is met.

MAX_AGE_IN_DAYS = 30

# To clean specific tables, please provide a comma-separated list per
# https://airflow.apache.org/docs/apache-airflow/stable/cli-and-env-variables-
ref.html#clean
# A value of None will clean all tables

TABLES_TO_CLEAN = None

with DAG(
    dag_id="clean_db_dag",
```

```
    schedule_interval=None,
    catchup=False,
    start_date=days_ago(1),
    params={
        "timestamp": Param(
            default=(datetime.now()-timedelta(days=MAX_AGE_IN_DAYS)).strftime("%Y-
%m-%d %H:%M:%S"),
            type="string",
            minLength=1,
            maxLength=255,
        ),
    }
) as dag:
    if TABLES_TO_CLEAN:
        bash_command="airflow db clean --clean-before-timestamp
 '{{ params.timestamp }}' --tables '"+TABLES_TO_CLEAN+"' --skip-archive --yes"
    else:
        bash_command="airflow db clean --clean-before-timestamp
 '{{ params.timestamp }}' --skip-archive --yes"

    cli_command = BashOperator(
        task_id="bash_command",
        bash_command=bash_command
    )
```

将环境元数据导出到 Amazon S3 上的 CSV 文件

使用以下代码示例创建有向无环图 (DAG)，该图在数据库中查询一系列 DAG 运行信息，并将数据写入存储在 Amazon S3 上的 `.csv` 文件中。

您可能需要从环境的 Aurora PostgreSQL 数据库中导出信息，以便在本地检查数据，将其存档到对象存储中，或者将它们与诸如 Amazon S3 到 Amazon Redshift 运算符和数据库清理之类的工具结合使用，以便将 Amazon MWAA 元数据移出环境，但保留它们以备将来分析。

您可以在数据库中查询 Apache Airflow 模型中列出的任何对象。此代码示例使用三个模型：`DagRun`、`TaskFail` 和 `TaskInstance`，它们提供与 DAG 运行相关的信息。

主题

- 版本
- 先决条件

版本

您可以在 Python 3.10 中将本页上的代码示例与 Apache Airflow v2 一起使用，在 Python 3.11 中与 Apache Airflow v3 一起使用。

先决条件

要使用本页上的示例代码，您需要以下内容：

- Amazon MWAA 环境。
- 您想要将元数据导出到新的 Amazon S3 存储桶。

Permissions

Amazon MWAA 需要获得 Amazon S3 操作 `s3:PutObject` 的权限，才能将查询的元数据信息写入 Amazon S3。将以下策略声明添加到环境的执行角色中。

```
{
  "Effect": "Allow",
  "Action": "s3:PutObject*",
  "Resource": "arn:aws:s3:::amzn-s3-demo-bucket"
}
```

此政策将写入权限限制为*amzn-s3-demo-bucket*。

要求

要在 Apache Airflow v2 和更高版本中使用此代码示例，无需附加依赖项。用于aws-mwaa-docker-images安装 Apache Airflow。

代码示例

以下步骤描述了如何创建 DAG，以查询 Aurora PostgreSQL 并将结果写入新的 Amazon S3 存储桶。

1. 在您的终端，导航到存储 DAG 代码的目录。例如：

```
cd dags
```

2. 复制以下代码示例的内容并本地另存为 metadata_to_csv.py。您可以更改分配给 MAX_AGE_IN_DAYS 的值，以控制 DAG 从元数据数据库中查询的最早记录的龄期。

```
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.end_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. 运行以下 AWS CLI 命令将 DAG 复制到环境的存储桶，然后使用 Apache Airflow 用户界面触发 DAG。

```
aws s3 cp your-dag.py s3://your-environment-bucket/dags/
```

4. 如果成功，输出将类似于在 `export_db` 任务的任务日志中的以下内容：

```
[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
```

现在，您可以在 `/files/export/` 中的新 Amazon S3 存储桶中访问和下载导出的 `.csv` 文件。

为 Apache Airflow 变量使用 AWS Secrets Manager 中的密钥

以下示例调用 AWS Secrets Manager 来获取 Amazon MWAA 上的 Apache Airflow 变量的密钥。它假设您已完成 使用密钥配置 Apache Airflow 连接 AWS Secrets Manager 中的步骤。

主题

版本

您可以在 Python 3.10 中将本页上的代码示例与 Apache Airflow v2 一起使用，在 Python 3.11 中与 Apache Airflow v3 一起使用。

先决条件

要使用本页上的示例代码，您需要以下内容：

- 创建 Secrets Manager 后端作为 Apache Airflow 配置选项，如 使用密钥配置 Apache Airflow 连接 AWS Secrets Manager 所列。
- Secrets Manager 中的 Apache Airflow 变量字符串，如 使用密钥配置 Apache Airflow 连接 AWS Secrets Manager 所列。

权限

- Secrets Manager 权限，如 使用密钥配置 Apache Airflow 连接 AWS Secrets Manager 所列。

要求

要在 Apache Airflow v2 和更高版本中使用此代码示例，无需附加依赖项。使用 aws-mwaa-docker-images 安装 Apache Airflow。

代码示例

以下步骤描述了如何创建 DAG 代码，以便调用 Secrets Manager 来获取密钥。

1. 在命令提示符下，导航到存储 DAG 代码的目录。例如：

```
cd dags
```

2. 复制以下代码示例的内容，并在本地另存为 `secrets-manager-var.py`。

```
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
    )
```

接下来做什么？

- 要了解如何将本示例中的 DAG 代码上传到 Amazon S3 存储桶的 dags 文件夹，请参阅 添加或更新 DAG。

使用 AWS Secrets Manager 中的密钥进行 Apache Airflow 连接

以下示例调用 AWS Secrets Manager 在 Amazon MWAA 上获取 Apache Airflow 连接的密钥。它假设您已完成 使用密钥配置 Apache Airflow 连接 AWS Secrets Manager 中的步骤。

主题

版本

您可以在 Python 3.10 中将本页上的代码示例与 Apache Airflow v2 一起使用，在 Python 3.11 中与 Apache Airflow v3 一起使用。

先决条件

要使用本页上的示例代码，您需要以下内容：

- 创建 Secrets Manager 后端作为 Apache Airflow 配置选项，如 使用密钥配置 Apache Airflow 连接 AWS Secrets Manager 所列。
- Secrets Manager 中的 Apache Airflow 连接字符串，如 使用密钥配置 Apache Airflow 连接 AWS Secrets Manager 所列。

权限

- Secrets Manager 权限，如 使用密钥配置 Apache Airflow 连接 AWS Secrets Manager 所列。

要求

要在 Apache Airflow v2 和更高版本中使用此代码示例，无需附加依赖项。使用 aws-mwaa-docker-images 安装 Apache Airflow。

代码示例

以下步骤描述了如何创建 DAG 代码，以便调用 Secrets Manager 来获取密钥。

1. 在命令提示符下，导航到存储 DAG 代码的目录。例如：

```
cd dags
```

2. 复制以下代码示例的内容，并在本地另存为 `secrets-manager.py`。

```
"""
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(region_name='us-east-1')
    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
    )
```

接下来做什么？

- 要了解如何将本示例中的 DAG 代码上传到 Amazon S3 存储桶的 dags 文件夹，请参阅 添加或更新 DAG。

使用 Oracle 创建自定义插件

以下示例将引导您完成使用 Oracle 为 Amazon MWAA 创建自定义插件的步骤，该插件可以与 plugins.zip 文件中的其他自定义插件和二进制文件组合使用。

目录

版本

您可以在 Python 3.10 中将本页上的代码示例与 Apache Airflow v2 一起使用，在 Python 3.11 中与 Apache Airflow v3 一起使用。

先决条件

要使用本页上的示例代码，您需要以下内容：

- Amazon MWAA 环境。
- 为环境启用任何日志级别、CRITICAL 或在上一部分中的工作线程日志记录。有关 Amazon MWAA 日志类型以及如何管理日志组的更多信息，请参阅 the section called “访问 Airflow 日志”。

权限

无需其他权限即可使用本页上的代码示例。

要求

要使用本页上的示例代码，请将以下依赖项添加到 `requirements.txt`。要了解更多信息，请参阅[安装 Python 依赖项]。

```
-c https://raw.githubusercontent.com/apache/airflow/constraints-2.0.2/
constraints-3.7.txt
cx_Oracle
apache-airflow-providers-oracle
```

代码示例

以下步骤介绍如何创建用于测试自定义插件的 DAG 代码。

1. 在命令提示符下，导航到存储 DAG 代码的目录。例如：

   ```
   cd dags
   ```

2. 复制以下代码示例的内容，并在本地另存为 `oracle.py`。

   ```
   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
    )
```

创建自定义插件

本节介绍如何下载依赖项、创建自定义插件和 plugins.zip。

下载依赖项

Amazon MWAA 会将 plugins.zip 的内容提取到每个 Amazon MWAA 计划程序和工作线程容器上的 `/usr/local/airflow/plugins`。这用于向环境中添加二进制文件。以下步骤介绍如何组装自定义插件所需的文件。

拉取 Amazon Linux 容器镜像

1. 在命令提示符下，提取 Amazon Linux 容器镜像，然后在本地运行该容器。例如：

   ```
   docker pull amazonlinux
         docker run -it amazonlinux:latest /bin/bash
   ```

   命令提示符可以调用 bash 命令行。例如：

   ```
   bash-4.2#
   ```

2. 安装 Linux 原生异步 I/O 工具（libaio）。

   ```
   yum -y install libaio
   ```

3. 请将此窗口保持打开状态以供后续步骤使用。我们将在本地复制以下文件：`lib64/libaio.so.1`、`lib64/libaio.so.1.0.0`、`lib64/libaio.so.1.0.1`。

下载客户端文件夹

1. 在本地安装解压缩包。例如：

   ```
   sudo yum install unzip
   ```

2. 创建 `oracle_plugin` 目录。例如：

```
mkdir oracle_plugin
cd oracle_plugin
```

3. 使用以下 curl 命令从适用于 Linux x86-64（64 位）的 Oracle 即时客户端下载中下载 instantclient-basic-linux.x64-18.5.0.0.0dbru.zip。

```
curl https://download.oracle.com/otn_software/linux/instantclient/185000/
instantclient-basic-linux.x64-18.5.0.0.0dbru.zip > client.zip
```

4. 解压缩 client.zip 文件。例如：

```
unzip *.zip
```

从 Docker 中提取文件

1. 在新的命令提示符下，显示并写下 Docker 容器 ID。例如：

```
docker container ls
```

命令提示符可以返回所有容器及其 ID。例如：

```
debc16fd6970
```

2. 在 oracle_plugin 目录中，将 lib64/libaio.so.1、lib64/libaio.so.1.0.0、lib64/libaio.so.1.0.1 文件解压缩到本地 instantclient_18_5 文件夹。例如：

```
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/
```

自定义插件

Apache Airflow 将在启动时执行插件文件夹中的 Python 文件内容。这用于设置和修改环境变量。以下步骤介绍了此自定义插件的示例代码。

- 复制以下代码示例的内容，并在本地另存为 env_var_plugin_oracle.py。

```
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

以下步骤说明如何创建 plugins.zip。此示例的内容可以与其他插件和二进制文件合并到单个 plugins.zip 文件中。

压缩插件目录中的内容。

1. 在命令行提示符中，导航到 oracle_plugin 目录。例如：

```
cd oracle_plugin
```

2. 将 instantclient_18_5 目录压缩到 plugins.zip 中。例如：

```
zip -r ../plugins.zip ./
```

您的命令提示符将显示：

```
oracle_plugin$ ls
client.zip  instantclient_18_5
```

3. 移除该 client.zip 文件。例如：

```
rm client.zip
```

压缩 env_var_plugin_oracle.py 文件

1. 将 env_var_plugin_oracle.py 文件添加到 plugins.zip 文件的根目录。例如：

```
zip plugins.zip env_var_plugin_oracle.py
```

2. plugins.zip 现在包含以下内容：

```
env_var_plugin_oracle.py
instantclient_18_5/
```

Airflow 配置选项

如果您使用的是 Apache Airflow v2，请添加 `core.lazy_load_plugins : False` 为 Apache Airflow 配置选项。要了解更多信息，请参阅 2 中的使用配置选项加载插件。

接下来做什么？

- 要了解如何将本示例中的 `requirements.txt` 文件上传到 Amazon S3 存储桶，请参阅 安装 Python 依赖项。
- 要了解如何将本示例中的 DAG 代码上传到 Amazon S3 存储桶的 `dags` 文件夹，请参阅 添加或更新 DAG。
- 要了解如何将本示例中的 `plugins.zip` 文件上传到 Amazon S3 存储桶，请参阅 安装自定义插件。

在 Amazon MWAA 上更改 DAG 的时区

默认情况下，Apache Airflow 将有向无环图（DAG）安排在 UTC+0 中。以下步骤展示了如何使用 Pendulum 更改 Amazon MWAA 运行 DAG 所在的时区。或者，本主题演示如何创建自定义插件来更改环境的 Apache Airflow 日志的时区。

主题

- 版本
- 先决条件
- 权限
- 创建插件以更改 Airflow 日志中的时区
- 创建 plugins.zip
- 代码示例

- 接下来做什么？

版本

您可以在 Python 3.10 中将本页上的代码示例与 Apache Airflow v2 一起使用，在 Python 3.11 中与 Apache Airflow v3 一起使用。

先决条件

要使用本页上的示例代码，您需要以下内容：

- Amazon MWAA 环境。

权限

无需其他权限即可使用本页上的代码示例。

创建插件以更改 Airflow 日志中的时区

Apache Airflow 在启动时运行 `plugins` 目录中的 Python 文件。使用以下插件，您可以覆盖执行程序的时区，该时区会修改 Apache Airflow 写入日志的时区。

1. 创建名为 `plugins` 的目录并导航到其中。例如：

```
$ mkdir plugins
$ cd plugins
```

2. 复制以下代码示例的内容，并将其本地另存为 `dag-timezone-plugin.py`，保存在 `plugins` 文件夹中。

```
import time
import os

os.environ['TZ'] = 'America/Los_Angeles'
time.tzset()
```

3. 在 `plugins` 目录中创建名为 `__init__.py` 的空 Python 文件。`plugins` 目录应类似于以下内容。

```
plugins/
```

```
|-- __init__.py
|-- dag-timezone-plugin.py
```

创建 plugins.zip

以下步骤说明如何创建 plugins.zip。此示例的内容可以与其他插件和二进制文件组合成单个 plugins.zip 文件。

1. 在命令提示符下，导航到上一步中的 plugins 目录。例如：

```
cd plugins
```

2. 将内容压缩到 plugins 目录中。

```
zip -r ../plugins.zip ./
```

3. 将 plugins.zip 上传到 S3 存储桶。

```
aws s3 cp plugins.zip s3://your-mwaa-bucket/
```

代码示例

要更改 DAG 运行的默认时区（UTC+0），我们将使用一个名为 Pendulum 的库，这是一个用于处理时区感知日期时间的 Python 库。

1. 在命令提示符下，导航到存储 DAG 的目录。例如：

```
cd dags
```

2. 复制以下示例的内容并另存为 tz-aware-dag.py。

```
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. 运行以下 AWS CLI 命令将 DAG 复制到环境的存储桶，然后使用 Apache Airflow UI 触发 DAG。

```
aws s3 cp your-dag.py s3://your-environment-bucket/dags/
```

4. 如果成功，您将输出类似于在 `tz_test` DAG 中的 `tz_aware_task` 的任务日志中的以下内容：

```
[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
```

接下来做什么？

- 要了解如何将本示例中的 `plugins.zip` 文件上传到 Amazon S3 存储桶，请参阅 安装自定义插件。

刷新 CodeArtifact 令牌

如果您使用 CodeArtifact 来安装 Python 依赖项，则 Amazon MWAA 需要有效的令牌。要允许 Amazon MWAA 在运行时访问 CodeArtifact 存储库，您可以使用启动脚本并使用令牌设置 `PIP_EXTRA_INDEX_URL`。

以下主题介绍如何创建启动脚本，该脚本使用 `get_authorization_token` CodeArtifact API 操作在每次环境启动或更新时检索新令牌。

主题

版本

您可以在 Python 3.10 中将本页上的代码示例与 Apache Airflow v2 一起使用，在 Python 3.11 中与 Apache Airflow v3 一起使用。

先决条件

要使用本页上的示例代码，您需要以下内容：

- Amazon MWAA 环境。
- CodeArtifact 存储库，用于存储环境的依赖项。

权限

要刷新 CodeArtifact 令牌并将结果写入 Amazon S3，Amazon MWAA 的执行角色必须具有以下权限。

- 该 `codeartifact:GetAuthorizationToken` 操作允许 Amazon MWAA 从 CodeArtifact 中检索新令牌。以下策略为您创建的每个 CodeArtifact 域授予权限。您可以通过修改语句中的资源值并仅指定您希望环境访问的域来进一步限制对所有域的访问。

```
{
  "Effect": "Allow",
  "Action": "codeartifact:GetAuthorizationToken",
  "Resource": "arn:aws:codeartifact:us-west-2:*:domain/*"
}
```

- 该 sts:GetServiceBearerToken 操作是调用 CodeArtifact GetAuthorizationToken API 操作所必需的。此操作返回一个令牌，在将程序包管理器（例如 pip）与 CodeArtifact 配合使用时，必须使用该令牌。要将程序包管理器与 CodeArtifact 存储库配合使用，环境的执行角色必须允许 sts:GetServiceBearerToken，如以下策略声明所列。

```
{
  "Sid": "AllowServiceBearerToken",
  "Effect": "Allow",
  "Action": "sts:GetServiceBearerToken",
  "Resource": "*"
}
```

代码示例

以下步骤描述了如何创建用于更新 CodeArtifact 令牌的启动脚本。

1. 复制以下代码示例的内容，并在本地另存为 code_artifact_startup_script.sh。

```
#!/bin/sh

# Startup script for MWAA, refer to 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. 导航到保存该脚本的文件夹。在新提示窗口中使用 cp 将脚本上传到存储桶。用您的信息替换 *amzn-s3-demo-bucket*。

```
aws s3 cp code_artifact_startup_script.sh s3://amzn-s3-demo-bucket/
code_artifact_startup_script.sh
```

如果成功，Amazon S3 会输出该对象的 URL 路径：

```
upload: ./code_artifact_startup_script.sh to s3://amzn-s3-demo-bucket/
code_artifact_startup_script.sh
```

上传脚本后，环境会在启动时更新并运行脚本。

接下来做什么？

- 要了解如何使用启动脚本自定义环境，请参阅 the section called “使用启动脚本”。
- 要了解如何将本示例中的 DAG 代码上传到 Amazon S3 存储桶的 dags 文件夹，请参阅 添加或更新 DAG。
- 要了解如何将本示例中的 plugins.zip 文件上传到 Amazon S3 存储桶，请参阅 安装自定义插件。

使用 Apache Hive 和 Hadoop 创建自定义插件

Amazon MWAA 将 plugins.zip 的内容提取到 /usr/local/airflow/plugins。这可以用来向容器中添加二进制文件。此外，Apache Airflow 会在启动时执行 plugins 文件夹中的 Python 文件内容，使您能够设置和修改环境变量。以下示例将引导您完成在 Amazon MWAA 环境中使用 Apache Hive 和 Hadoop 创建自定义插件的步骤，该插件可以与其他自定义插件和二进制文件组合使用。

主题

- 版本

版本

您可以在 Python 3.10 中将本页上的代码示例与 Apache Airflow v2 一起使用，在 Python 3.11 中与 Apache Airflow v3 一起使用。

先决条件

要使用本页上的示例代码，您需要以下内容：

- Amazon MWAA 环境。

权限

无需其他权限即可使用本页上的代码示例。

要求

要使用本页上的示例代码，请将以下依赖项添加到 `requirements.txt`。要了解更多信息，请参阅安装 Python 依赖项。

```
-c https://raw.githubusercontent.com/apache/airflow/constraints-2.0.2/
constraints-3.7.txt
apache-airflow-providers-amazon[apache.hive]
```

下载依赖项

Amazon MWAA 会将 plugins.zip 的内容提取到每个 Amazon MWAA 计划程序和工作线程容器上的 `/usr/local/airflow/plugins`。这用于向环境中添加二进制文件。以下步骤介绍如何组装自定义插件所需的文件。

1. 在命令提示符下，导航到要创建插件的目录。例如：

```
cd plugins
```

2. 从镜像中下载 Hadoop，例如：

```
wget https://downloads.apache.org/hadoop/common/hadoop-3.3.0/hadoop-3.3.0.tar.gz
```

3. 从镜像中下载 Hive，例如：

```
wget https://downloads.apache.org/hive/hive-3.1.2/apache-hive-3.1.2-bin.tar.gz
```

4. 创建目录。例如：

```
mkdir hive_plugin
```

5. Hadoop 提取。

```
tar -xvzf hadoop-3.3.0.tar.gz -C hive_plugin
```

6. Hive 提取。

```
tar -xvzf apache-hive-3.1.2-bin.tar.gz -C hive_plugin
```

自定义插件

Apache Airflow 将在启动时执行插件文件夹中的 Python 文件内容。这用于设置和修改环境变量。以下步骤介绍了此自定义插件的示例代码。

1. 在命令行提示符中，导航到 `hive_plugin` 目录。例如：

```
cd hive_plugin
```

2. 复制以下代码示例的内容，并在 `hive_plugin` 目录中将其本地另存为 `hive_plugin.py`。

```
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. 复制以下文本的内容，并在 hive_plugin 目录中将其本地另存为 .airflowignore。

```
hadoop-3.3.0
apache-hive-3.1.2-bin
```

Plugins.zip

以下步骤说明如何创建 plugins.zip。此示例的内容可以与其他插件和二进制文件组合成一个 plugins.zip 文件。

1. 在命令提示符下，导航到上一步中的 hive_plugin 目录。例如：

```
cd hive_plugin
```

2. 将内容压缩到 plugins 文件夹中。

```
zip -r ../hive_plugin.zip ./
```

代码示例

以下步骤介绍如何创建用于测试自定义插件的 DAG 代码。

1. 在命令提示符下，导航到存储 DAG 代码的目录。例如：

```
cd dags
```

2. 复制以下代码示例的内容，并在本地另存为 hive.py。

```
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 配置选项

如果您使用的是 Apache Airflow v2，请添加 core.lazy_load_plugins : False 为 Apache Airflow 配置选项。要了解更多信息，请参阅 2 中的使用配置选项加载插件。

接下来做什么？

- 要了解如何将本示例中的 requirements.txt 文件上传到 Amazon S3 存储桶，请参阅 安装 Python 依赖项。
- 要了解如何将本示例中的 DAG 代码上传到 Amazon S3 存储桶的 dags 文件夹，请参阅 添加或更新 DAG。
- 要了解如何将本示例中的 plugins.zip 文件上传到 Amazon S3 存储桶，请参阅 安装自定义插件。

为 Apache Airflow PythonVirtualenvOperator 创建自定义插件

以下示例说明了如何在 Amazon Managed Workflows for Apache Airflow 上使用自定义插件修补 Apache Airflow PythonVirtualenvOperator。

主题

- 版本

版本

您可以在 Python 3.10 中将本页上的代码示例与 Apache Airflow v2 一起使用，在 Python 3.11 中与 Apache Airflow v3 一起使用。

先决条件

要使用本页上的示例代码，您需要以下内容：

- Amazon MWAA 环境。

权限

无需其他权限即可使用本页上的代码示例。

要求

要使用本页上的示例代码，请将以下依赖项添加到 `requirements.txt`。要了解更多信息，请参阅安装 Python 依赖项。

```
virtualenv
```

自定义插件示例代码

Apache Airflow 将在启动时执行插件文件夹中的 Python 文件内容。此插件将在启动过程中修补内置的 `PythonVirtualenvOperator`，使其与 Amazon MWAA 兼容。以下步骤显示了此自定义插件的示例代码。

1. 在命令提示符下，导航到上一部分中的 plugins 目录。例如：

```
cd plugins
```

2. 复制以下代码示例的内容，并在本地另存为 virtual_python_plugin.py。

```
"""
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'
```

Plugins.zip

以下步骤说明如何创建 plugins.zip。

1. 在命令提示符下，导航到上一部分中包含 virtual_python_plugin.py 的目录。例如：

```
cd plugins
```

2. 将内容压缩到 plugins 文件夹中。

```
zip plugins.zip virtual_python_plugin.py
```

代码示例

以下步骤介绍了如何创建自定义插件的 DAG 代码。

1. 在命令提示符下，导航到存储 DAG 代码的目录。例如：

```
cd dags
```

2. 复制以下代码示例的内容，并在本地另存为 virtualenv_test.py。

```
"""
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,
    )
```

Airflow 配置选项

如果您使用的是 Apache Airflow v2，请添加 `core.lazy_load_plugins : False` 为 Apache Airflow 配置选项。要了解更多信息，请参阅 2 中的使用配置选项加载插件。

接下来做什么？

- 要了解如何将本示例中的 `requirements.txt` 文件上传到 Amazon S3 存储桶，请参阅 安装 Python 依赖项。
- 要了解如何将本示例中的 DAG 代码上传到 Amazon S3 存储桶的 `dags` 文件夹，请参阅 添加或更新 DAG。
- 要了解如何将本示例中的 `plugins.zip` 文件上传到 Amazon S3 存储桶，请参阅 安装自定义插件。

使用 Lambda DAGs 函数进行调用

以下代码示例使用 AWS Lambda 函数获取 Apache Airflow CLI 令牌并在 Amazon MWAA 环境中调用有向无环图（DAG）。

主题

版本

您可以在 Python 3.10 中将本页上的代码示例与 Apache Airflow v2 一起使用，在 Python 3.11 中与 Apache Airflow v3 一起使用。

先决条件

要使用此代码示例，您必须：

- 使用 Amazon MWAA 环境公共网络访问模式。
- 使用最新的 Python 运行时创建一个 Lambda 函数。

> **Note**
>
> 如果 Lambda 函数和 Amazon MWAA 环境处于同一 VPC 中，则可以在私有网络上使用此代码。对于本配置，Lambda 函数的执行角色需要获得调用 Amazon Elastic Compute Cloud（Amazon EC2）CreateNetworkInterface API 操作的权限。您可以使用 `AWSLambdaVPCAccessExecutionRole` AWS-managed 策略提供此权限。

Permissions

要使用本页上的代码示例，Amazon MWAA 环境的执行角色需要访问权限才能执行 `airflow:CreateCliToken` 操作。您可以使用 `AmazonMWAAAirflowCliAccess` AWS-managed 策略提供此权限：

JSON

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

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "airflow:CreateCliToken"
            ],
            "Resource": "*"
        }
    ]
}
```

有关更多信息，请参阅Apache Airflow CLI 政策： AmazonMWAAAirflowCliAccess。

依赖项

要在 Apache Airflow v2 和更高版本中使用此代码示例，无需附加依赖项。用于aws-mwaa-docker-images安装 Apache Airflow。

代码示例

1. 打开 AWS Lambda 控制台，网址为https://console.aws.amazon.com/lambda/。
2. 从 Functions 列表中选择 Lambda 函数。
3. 在函数页面上，复制以下代码并将以下代码替换为资源名称：
   - YOUR_ENVIRONMENT_NAME – Amazon MWAA 环境名称。
   - YOUR_DAG_NAME – 您想调用的 DAG 名称。

```
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. 选择部署。
5. 选择测试，使用 Lambda 控制台调用函数。
6. 要验证 Lambda 是否成功调用了 DAG，请使用 Amazon MWAA 控制台导航到环境的 Apache Airflow UI 界面，然后执行以下操作：
   a. 在该DAGs页面上，在列表中找到您的新目标 DAG DAGs。
   b. 在上次运行下，查看最新 DAG 运行的时间戳。此时间戳应与您其他环境中 `invoke_dag` 的最新时间戳非常匹配。
   c. 在近期任务下，检查上次运行是否成功。

DAGs 在不同的 Amazon MWAA 环境中调用

以下代码示例创建了一个 Apache Airflow CLI 令牌。然后，该代码使用一个 Amazon MWAA 环境中的有向无环图（DAG）在另一个 Amazon MWAA 环境中调用 DAG。

主题

版本

您可以在 Python 3.10 中将本页上的代码示例与 Apache Airflow v2 一起使用，在 Python 3.11 中与 Apache Airflow v3 一起使用。

先决条件

要使用本页上的代码示例，您需要以下内容：

- 两个具有公有网络 Web 服务器访问权限的 Amazon MWAA 环境，包括您当前的环境。
- 上传到目标环境的 Amazon Simple Storage Service（Amazon S3）桶的示例 DAG。

Permissions

要使用本页上的代码示例，环境的执行角色必须具有创建 Apache Airflow CLI 令牌的权限。您可以附加 AWS-managed 策略`AmazonMWAAAirflowCliAccess`来授予此权限。

JSON

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

有关更多信息，请参阅Apache Airflow CLI 政策： AmazonMWAAAirflowCliAccess。

依赖项

要在 Apache Airflow v2 和更高版本中使用此代码示例，无需附加依赖项。用于aws-mwaa-docker-images安装 Apache Airflow。

代码示例

以下代码示例假设您在当前环境中使用 DAG 在另一个环境中调用 DAG。

1. 在您的终端，导航到存储 DAG 代码的目录。例如：

```
cd dags
```

2. 复制以下示例的内容并本地另存为 invoke_dag.py。用您自己的信息替换以下值。

   - your-new-environment-name— 您要调用 DAG 的另一个环境的名称。
   - your-target-dag-id— 您要调用 DAG 的另一个环境中的 DAG 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. 运行以下 AWS CLI 命令将 DAG 复制到环境的存储桶，然后使用 Apache Airflow 用户界面触发 DAG。

```
aws s3 cp your-dag.py s3://your-environment-bucket/dags/
```

4. 如果 DAG 成功运行，您将看到类似于 `invoke_dag_task` 的任务日志中的以下内容的输出。

```
[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
```

要验证 DAG 是否已成功调用，请导航到新环境的 Apache Airflow UI，然后执行以下操作：

a. 在该DAGs页面上，在列表中找到您的新目标 DAG DAGs。

b. 在上次运行下，查看最新 DAG 运行的时间戳。此时间戳应与您其他环境中 `invoke_dag` 的最新时间戳非常匹配。

c. 在近期任务下，检查上次运行是否成功。

将 Amazon RDS for Microsoft SQL Server 与 Amazon MWAA 一起使用

您可以使用 Amazon MWAA 连接到 RDS for SQL Server。以下示例代码使用 Amazon Managed Workflows for Apache Airflow 环境中的 DAG 连接到 Amazon RDS for Microsoft SQL Server 并在其上执行查询。

主题

- 版本
- 先决条件
- 依赖项
- Apache Airflow v2 连接

- 代码示例
- 接下来做什么？

版本

您可以在 Python 3.10 中将本页上的代码示例与 Apache Airflow v2 一起使用，在 Python 3.11 中与 Apache Airflow v3 一起使用。

先决条件

要使用本页上的示例代码，您需要以下内容：

- Amazon MWAA 环境。
- Amazon MWAA 和 RDS for SQL Server 在同一个 Amazon VPC/上运行
- Amazon MWAA 和服务器的 VPC 安全组配置有以下连接：
  - Amazon MWAA 安全组中对 Amazon RDS 1433 开放端口的入站规则
  - 或者是从 Amazon MWAA 到 RDS 开放端口 1433 的出站规则
- RDS for SQL Server 的 Apache Airflow 连接反映了在之前过程中创建的 Amazon RDS SQL 服务器数据库的主机名、端口、用户名和密码。

依赖项

要使用本节中的示例代码，请将以下依赖项添加到 `requirements.txt`。要了解更多信息，请参阅安装 Python 依赖项。

```
apache-airflow-providers-microsoft-mssql==1.0.1
   apache-airflow-providers-odbc==1.0.1
   pymssql==2.2.1
```

Apache Airflow v2 连接

如果您在 Apache Airflow v2 中使用连接，请确保 Airflow 连接对象包含以下键值对：

1. 连接 ID：mssql_default
2. 连接类型：Amazon Web Services
3. 主机：YOUR_DB_HOST

4. 架构：
5. 登录：管理员
6. 密码：
7. 端口：1433
8. 附加依赖项：

代码示例

1. 在命令提示符下，导航到存储 DAG 代码的目录。例如：

```
cd dags
```

2. 复制以下代码示例的内容，并在本地另存为 sql-server.py。

```
"""
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
```

接下来做什么？

- 要了解如何将本示例中的 `requirements.txt` 文件上传到 Amazon S3 存储桶，请参阅 安装 Python 依赖项。
- 要了解如何将本示例中的 DAG 代码上传到 Amazon S3 存储桶的 `dags` 文件夹，请参阅 添加或更新 DAG。
- 浏览示例脚本和其他 pymssql 模块示例。
- 在《Apache Airflow 参考指南》中详细了解如何使用 mssql_operator 在特定的 Microsoft SQL 数据库中执行 SQL 代码。

将 Amazon MWAA 与 Amazon EKS 一起使用

以下示例演示了如何将 Amazon MWAA 与 Amazon EKS 一起使用。

主题

- 版本

- 先决条件
- 创建 Amazon EC2 公有密钥
- 创建集群
- 创建 mwaa 命名空间
- 为 mwaa 命名空间创建角色
- 创建并附加 Amazon EKS 集群的 IAM 角色
- 创建 requirements.txt 文件
- 为 Amazon EKS 创建身份映射
- 创建 kubeconfig
- 创建 DAG
- 将 DAG 和 kube_config.yaml 添加到 Amazon S3 存储桶中
- 启用并触发示例

版本

您可以在 Python 3.10 中将本页上的代码示例与 Apache Airflow v2 一起使用，在 Python 3.11 中与 Apache Airflow v3 一起使用。

先决条件

要使用本主题中的示例，您需要以下内容：

- Amazon MWAA 环境。
- eksctl。要了解更多信息，请参阅安装 eksctl。
- kubectl。要了解更多信息，请参阅安装和设置 kubectl。在某些情况下，它是与 eksctl 一起安装的。
- 在您创建 Amazon MWAA 环境的区域中的 EC2 密钥对。要了解更多信息，请参阅创建或导入密钥对。

> **Note**
>
> 使用 `eksctl` 命令时，可以包含 `--profile`，以指定默认配置文件以外的配置文件。

创建 Amazon EC2 公有密钥

使用以下命令，以从私有密钥对中创建公有密钥。

```
ssh-keygen -y -f myprivatekey.pem > mypublickey.pub
```

要了解更新信息，请参阅检索密钥对的公有密钥。

创建集群

使用以下命令来创建集群。如果您想要为集群自定义名称或在其他区域创建集群，请替换名称和区域值。您必须在与您在创建 Amazon MWAA 环境的同一区域中创建集群。替换子网的值，使其与您用于 Amazon MWAA 的 Amazon VPC 网络中的子网相匹配。替换 ssh-public-key 的值以匹配您使用的密钥。您可以使用位于同一区域的 Amazon EC2 中的现有密钥，也可以在创建 Amazon MWAA 环境的同一区域创建新密钥。

```
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"
```

完成集群的创建需要一段时间。完成后，您可以使用以下命令验证集群是否已成功创建并配置了 IAM OIDC 提供商：

```
eksctl utils associate-iam-oidc-provider \
--region us-west-2 \
--cluster mwaa-eks \
--approve
```

创建 mwaa 命名空间

确认集群已成功创建后，使用以下命令为 pod 创建命名空间。

```
kubectl create namespace mwaa
```

为 mwaa 命名空间创建角色

创建命名空间后，在 EKS 上为可在 MWAA 命名空间中运行 pod 的 Amazon MWAA 用户创建角色和角色绑定。如果您为命名空间使用了不同的名称，请将 -n *mwaa* 中的 mwaa 名称替换为您使用的名称。

```
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
```

运行以下命令来确认新角色可以访问 Amazon EKS 集群。如果您没有使用以下名称，请务必使用正确的名称*mwaa*：

```
kubectl get pods -n mwaa --as mwaa-service
```

您会收到一条含有如下内容的消息：

```
No resources found in mwaa namespace.
```

创建并附加 Amazon EKS 集群的 IAM 角色

您必须创建一个 IAM 角色，然后将其绑定到 Amazon EKS（k8s）集群，这样该角色才能通过 IAM 进行身份验证。该角色仅用于登录集群，没有任何控制台或 API 调用的权限。

使用 Amazon MWAA 执行角色 中的步骤为 Amazon MWAA 环境创建新角色。但是，与其创建和附加该主题中描述的策略，不如附加以下策略：

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "airflow:PublishMetrics",
            "Resource": "arn:aws:airflow:us-east-1:111122223333: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:us-east-1:111122223333: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:us-east-1:*:airflow-celery-*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey",
                "kms:GenerateDataKey*",
                "kms:Encrypt"
            ],
            "NotResource": "arn:aws:kms:*:111122223333:key/*",
            "Condition": {
                "StringLike": {
                    "kms:ViaService": [
                    "sqs.us-east-1.amazonaws.com"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "eks:DescribeCluster"
            ],
            "Resource": "arn:aws:eks:us-east-1:111122223333:cluster/
${EKS_CLUSTER_NAME}"
        }
    ]
}
```

创建角色后，编辑 Amazon MWAA 环境，使用您创建的角色作为环境的执行角色。要更改角色，请编辑要使用的环境。您可以在“权限”下选择执行角色。

已知问题：

- 角色 ARNs 存在一个已知问题，子路径无法通过 Amazon EKS 进行身份验证。解决方法是手动创建服务角色，而不是使用 Amazon MWAA 自己创建的服务角色。要了解更多信息，请参阅在 aws-auth ConfigMap 中当 ARN 包含路径时，带有路径的角色不起作用
- 如果 IAM 中没有 Amazon MWAA 服务列表，则需要选择备用服务策略，例如 Amazon EC2，然后更新该角色的信任策略以匹配以下内容：

  JSON

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

  要了解更多信息，请参阅如何在 IAM 角色中使用信任策略。

创建 requirements.txt 文件

要使用本节中的示例代码，请确保已向 `requirements.txt` 中添加了以下数据库选项之一。要了解更多信息，请参阅 安装 Python 依赖项。

```
kubernetes
apache-airflow[cncf.kubernetes]==3.0.0
```

为 Amazon EKS 创建身份映射

使用您在以下命令中创建的角色的 ARN 为 Amazon EKS 创建身份映射。将区域*us-east-1*更改为创建环境的区域。替换角色的 ARN，最后替换*mwaa-execution-role*为环境的执行角色。

```
eksctl create iamidentitymapping \
```

```
--region us-east-1 \
--cluster mwaa-eks \
--arn arn:aws:iam::123456789012:role/mwaa-execution-role \
--username mwaa-service
```

创建 kubeconfig

使用以下命令创建 kubeconfig：

```
aws eks update-kubeconfig \
--region us-west-2 \
--kubeconfig ./kube_config.yaml \
--name mwaa-eks \
--alias aws
```

如果您在运行 update-kubeconfig 时使用了特定的配置文件，则需要删除添加到 kube_config.yaml 文件中的 env: 部分，这样它才能在 Amazon MWAA 中正常运行。为此，请从文件中删除以下内容，然后将其保存：

```
env:
 - name: AWS_PROFILE
 value: profile_name
```

创建 DAG

使用以下代码示例创建 Python 文件，例如 DAG 的 mwaa_pod_example.py 文件。

```
"""
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'
                       )
```

将 DAG 和 **kube_config.yaml** 添加到 Amazon S3 存储桶中

将您创建的 DAG 和 kube_config.yaml 文件放入 Amazon MWAA 环境的 Amazon S3 存储桶中。您可以使用 Amazon S3 控制台或 AWS Command Line Interface将所有文件放入存储桶中。

启用并触发示例

在 Apache Airflow 中，启用该示例，然后将其触发。

成功运行并完成后，使用以下命令验证 Pod：

```
kubectl get pods -n mwaa
```

您将获得与下内容类似的输出：

```
NAME READY STATUS RESTARTS AGE
mwaa-pod-test-aa11bb22cc3344445555666677778888 0/1 Completed 0 2m23s
```

然后，您可以使用以下命令验证 Pod 的输出。请将名称值替换为上一个命令返回的值：

```
kubectl logs -n mwaa mwaa-pod-test-aa11bb22cc3344445555666677778888
```

使用 ECSOperator 连接 Amazon ECS

此主题介绍如何从 Amazon MWAA 使用 ECSOperator 连接到 Amazon Elastic Container Service（Amazon ECS）容器。在以下步骤中，您将向环境的执行角色添加所需的权限，使用 CloudFormation 模板创建 Amazon ECS Fargate 集群，最后创建并上传连接到新集群的 DAG。

主题

版本

您可以在 Python 3.10 中将本页上的代码示例与 Apache Airflow v2 一起使用，在 Python 3.11 中与 Apache Airflow v3 一起使用。

先决条件

要使用本页上的示例代码，您需要以下内容：

- Amazon MWAA 环境。

Permissions

- 环境的执行角色需要权限才能在 Amazon ECS 中运行任务。您可以将 Amazonecs_ FullAccess AWS托管策略附加到您的执行角色，也可以创建以下策略并将其附加到您的执行角色。

  JSON

```
{
    "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"
                }
            }
        }
    ]
}
```

- 除了添加在 Amazon ECS 中运行任务所需的权限外，您还必须修改 Amazon MWAA 执行角色中的 CloudWatch 日志策略声明，以允许访问 Amazon ECS 任务日志组，如下所示。Amazon ECS 日志组由中的 CloudFormation 模板创建the section called “创建 Amazon ECS 集群”。

```
{
  "Effect": "Allow",
  "Action": [
    "logs:CreateLogStream",
    "logs:CreateLogGroup",
```

```
      "logs:PutLogEvents",
      "logs:GetLogEvents",
      "logs:GetLogRecord",
      "logs:GetLogGroupFields",
      "logs:GetQueryResults"
    ],
    "Resource": [
      "arn:aws:logs:us-east-1:123456789012:log-group:airflow-environment-name-*",
      "arn:aws:logs:*:*:log-group:ecs-mwaa-group:*"
    ]
}
```

有关 Amazon MWAA 执行角色，以及如何附加策略的更多信息，请参阅 执行角色。

创建 Amazon ECS 集群

使用以下 CloudFormation 模板，您将构建一个 Amazon ECS Fargate 集群，用于您的 Amazon MWAA 工作流程。有关更多信息，请参阅《Amazon Elastic Container Service 开发人员指南》中的创建一个任务定义。

1. 创建一个包含以下代码的 JSON 文件，并将该文件另存为 `ecs-mwaa-cfn.json`。

```
{
    "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. 在命令提示符下，使用以下 AWS CLI 命令创建新堆栈。您必须将 `SecurityGroups` 和 `SubnetIds` 的值替换为 Amazon MWAA 环境的安全组和子网的值。

```
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
```

或者，您可以使用以下 Shell 脚本：该脚本使用[get-environment](#) AWS CLI 命令检索环境的安全组和子网所需的值，然后相应地创建堆栈。要运行该脚本，请运行以下命令。

a. 复制脚本并将其保存到与 CloudFormation 模板相同的目录中。`ecs-stack-helper.sh`

```
#!/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. 使用以下命令运行该脚本。将 `environment-name` 和 `stack-name` 替换为您的信息。

```
chmod +x ecs-stack-helper.sh
./ecs-stack-helper.bash environment-name stack-name
```

如果成功，您将参考以下显示您的新 CloudFormation 堆栈 ID 的输出。

```
{
  "StackId": "arn:aws:cloudformation:us-west-2:123456789012:stack/my-ecs-
stack/123456e7-8ab9-01cd-b2fb-36cce63786c9"
}
```

在 CloudFormation 堆栈完成并 AWS 预配置 Amazon ECS 资源后，您就可以创建和上传您的 DAG 了。

代码示例

1. 打开命令提示符，然后导航到存储 DAG 代码的目录。例如：

```
cd dags
```

2. 复制以下代码示例的内容并将其本地另存为 mwaa-ecs-operator.py，然后将新 DAG 上传到 Amazon S3。

```
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**
>
> 在 DAG 的示例中，对于 awslogs_group，您可能需要使用 Amazon ECS 任务日志组的名称修改日志组。示例假设名为 mwaa-ecs-zero 的日志组。对于 awslogs_stream_prefix，使用 Amazon ECS 任务日志流前缀。该示例假设日志流前缀为 ecs。

3. 运行以下 AWS CLI 命令将 DAG 复制到环境的存储桶，然后使用 Apache Airflow 用户界面触发 DAG。

```
aws s3 cp your-dag.py s3://your-environment-bucket/dags/
```

4. 如果成功，您将在 ecs_fargate_dag DAG 的任务日志中看到输出，类似于 ecs_operator_task 的任务日志中的以下内容：

```
[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
```

在 Amazon MWAA 中使用 dbt

本主题演示了如何在 Amazon MWAA 中使用 dbt 和 Postgres。在以下步骤中，您将所需的依赖项添加到 `requirements.txt` 中，并将示例 dbt 项目上传到环境的 Amazon S3 存储桶。然后，您将使用示例 DAG 来验证 Amazon MWAA 是否已安装依赖项，最后使用 `BashOperator` 来运行 dbt 项目。

主题

版本

您可以在 Python 3.10 中将本页上的代码示例与 Apache Airflow v2 一起使用，在 Python 3.11 中与 Apache Airflow v3 一起使用。

先决条件

在完成以下步骤之前，您需要具备以下条件：

- 使用 Apache Airflow v2.2.2 的 Amazon MWAA 环境。此示例已编写，并使用 v2.2.2 进行了测试。您可能需要修改示例以与其他 Apache Airflow 版本一起使用。
- dbt 项目示例。要开始在 Amazon MWAA 中使用 dbt，你可以创建一个分支并从 dbt-labs 存储库中克隆 dbt 入门项目。 GitHub

依赖项

要将 Amazon MWAA 与 dbt 配合使用，请将以下启动脚本添加到环境中。要了解更多信息，请参阅在 Amazon MWAA 中使用启动脚本。

```
#!/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
```

在以下章节中，您可将 dbt 项目目录上传到 Amazon S3 并运行 DAG 来验证 Amazon MWAA 是否已成功安装所需的 dbt 依赖项。

将 dbt 项目上传到 Amazon S3

为了能够在 Amazon MWAA 环境中使用 dbt 项目，您可以将整个项目目录上传到环境的 dags 文件夹中。当环境更新时，Amazon MWAA 会将 dbt 目录下载到本地 usr/local/airflow/dags/ 文件夹。

要将 dbt 项目上传到 Amazon S3，请执行以下操作

1. 导航到您克隆 dbt 入门项目的目录。
2. 运行以下 Amazon S3 AWS CLI 命令，使用--recursive参数以递归方式将项目内容复制到您的环境dags文件夹。该命令会创建一个名为 dbt 的子目录，您可以将其用于所有 dbt 项目。如果子目录已经存在，则项目文件将被复制到现有目录中，并且不会创建新目录。该命令还会为该特定入门项目在 dbt 目录中创建一个子目录。

   ```
   aws s3 cp dbt-starter-project s3://amzn-s3-demo-bucket/dags/dbt/dbt-starter-project
    --recursive
   ```

   您可以为项目子目录使用不同的名称，以便在 dbt 父目录中组织多个 dbt 项目。

使用 DAG 验证 dbt 依赖项的安装

以下 DAG 使用 BashOperator 和 bash 命令来验证 Amazon MWAA 是否已成功安装 requirements.txt 中指定的 dbt 依赖项。

```
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/python3-virtualenv/dbt-env/bin/dbt --version"
```

```
    )
```

执行以下操作以访问任务日志并验证是否已安装 dbt 及其依赖项。

1. 导航到 Amazon MWAA 控制台，然后从可用环境列表中选择打开 Airflow UI。
2. 在 Apache Airflow UI 上，从列表中找到 dbt-installation-test DAG，然后在 Last Run 列中选择日期以打开上一个成功的任务。
3. 使用图表视图，选择 bash_command 任务以打开任务实例的详细信息。
4. 选择日志来打开任务日志，然后验证日志是否成功列出了我们在 requirements.txt 中指定的 dbt 版本。

使用 DAG 来运行 dbt 项目

以下 DAG 使用 BashOperator 将您从本地 usr/local/airflow/dags/ 目录上传到 Amazon S3 的 dbt 项目复制到可写入的 /tmp 目录，然后运行 dbt 项目。bash 命令假设一个名为 dbt-starter-project 的 入门 dbt 项目。根据您项目目录的名称修改目录名称。

```
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 博客和教程

- 使用 Amazon EKS 和 Apache Airflow v2.x 的 Amazon MWAA

Amazon MMWAA 的最佳实践

本指南介绍了在使用 Amazon MWAA 时我们推荐的最佳实践。

主题

- Amazon MWAA 上的 Apache Airflow 的性能调整
- 在 requirements.txt 中管理 Python 依赖项

Amazon MWAA 上的 Apache Airflow 的性能调整

本主题介绍如何使用 在 Amazon MWAA 上使用 Apache Airflow 配置选项 调整 Amazon MWAA 环境的性能。

目录

添加 Apache Airflow 配置选项。

执行以下过程将 Airflow 配置选项添加到环境中。

1. 在 Amazon MWAA 控制台上打开环境页面。
2. 选择环境。
3. 选择编辑。

4. 选择下一步。
5. 在 Airflow 配置选项窗格中选择添加自定义配置。
6. 从下拉列表中选择配置并输入值，或者输入自定义配置并输入值。
7. 为每个您想要添加的配置选择添加自定义配置选项。
8. 选择保存。

要了解更多信息，请参阅 在 Amazon MWAA 上使用 Apache Airflow 配置选项。

Apache Airflow 计划程序

Apache Airflow 计划程序是 Apache Airflow 的核心组件。计划程序出现问题可能会导致无法解析 DAG 且无法调度任务。有关 Apache Airflow 调度程序调整的更多信息，请参阅 Fine-tuning Apache Airflow 文档网站上的调度器性能。

参数

本节介绍可用于 Apache Airflow 计划程序（Apache Airflow v2 和更高版本）的配置选项及其使用案例。

Apache Airflow v3

配置	使用案例
celery.sync_parallelism Celery 执行程序用于同步任务状态的进程数。 默认值：1	您可以使用此选项通过限制 Celery 执行程序使用的进程来防止队列冲突。默认情况下，将值设置为，1以防止在将任务日志传送到 CloudWatch 日志时出错。将该值设为 0 意味着使用最大进程数，但在传送任务日志时可能会导致错误。
scheduler.scheduler_idle_sleep_time 在计划程序“循环”中连续处理 DAG 文件之间等待的秒数。 默认值：1	在计划程序检索 DAG 解析结果、查找任务和对任务进行排队以及在执行程序中执行排队任务后，您可以使用此选项通过延长计划程序的休眠时间来释放计划程序上的 CPU 使用率。增加此值会消耗在 Apache Airflow v2 和 Apache Airflow v3 的 dag_proce

配置	使用案例
	ssor.parsing_processes 的环境中运行的计划程序线程数。这可能会降低计划程序解析 DAG 的容量，并增加 DAG 出现在 Web 服务器上的时间。
scheduler.max_dagruns_to_create_per_loop 为每个调度程序“循环”创建DagRuns的最大 DAG 数。 默认值：10	您可以使用此选项通过减少调度程序“循环”的最大数量来腾出资源DagRuns用于调度任务。
dag_processor.parsing_processes 计划程序可以并行运行以调度 DAG 的线程数。 默认：使用 (2 * number of vCPUs) - 1	您可以使用此选项通过减少计划程序并行运行解析 DAG 的进程数来释放资源。如果 DAG 解析影响任务调度，我们建议将此数量保持在较低水平。您必须指定一个小于环境中 vCPU 计数的值。要了解更多信息，请参阅限制。

Apache Airflow v2

配置	使用案例
celery.sync_parallelism Celery 执行程序用于同步任务状态的进程数。 默认值：1	您可以使用此选项通过限制 Celery 执行程序使用的进程来防止队列冲突。默认情况下，将值设置为，1以防止在将任务日志传送到 CloudWatch 日志时出错。将该值设为 0 意味着使用最大进程数，但在传送任务日志时可能会导致错误。
scheduler.idle_sleep_time 在计划程序“循环”中连续处理 DAG 文件之间等待的秒数。	在计划程序检索 DAG 解析结果、查找任务和对任务进行排队以及在执行程序中执行排队任务后，您可以使用此选项通过延长计划程序的休眠时间来释放计划程序上的 CPU

配置	使用案例
默认值：1	使用率。增加此值会消耗在 Apache Airflow v2 和 Apache Airflow v3 的 `scheduler.parsing_processes` 的环境中运行的计划程序线程数。这可能会降低计划程序解析 DAG 的容量，并增加 DAG 出现在 Web 服务器上的时间。
scheduler.max_dagruns_to_create_per_loop 为每个调度程序 “循环” 创建DagRuns的最大 DAG 数。 默认值：10	您可以使用此选项通过减少调度程序 “循环” 的最大数量来腾出资源DagRuns用于调度任务。
scheduler.parsing_processes 计划程序可以并行运行以调度 DAG 的线程数。 默认：使用 `(2 * number of vCPUs) - 1`	您可以使用此选项通过减少计划程序并行运行解析 DAG 的进程数来释放资源。如果 DAG 解析影响任务调度，我们建议将此数量保持在较低水平。您必须指定一个小于环境中 vCPU 计数的值。要了解更多信息，请参阅限制。

限制

本节介绍调整计划程序的默认参数时要考虑的限制。

scheduler.parsing_processes、scheduler.max_threads（仅限于 v2）

对于环境类，每个 vCPU 允许使用两个线程。必须为环境类的计划程序保留至少一个线程。如果您发现任务计划出现延迟，则可能需要增加环境类。例如，大型环境为其计划程序设有一个 4 vCPU 的 Fargate 容器实例。这意味着可供其他进程使用的 7 个线程总数的上限。也就是说，两个线程乘以四个 vCPU，计划程序本身减一。您在 `scheduler.max_threads`（仅限于 v2）和 `scheduler.parsing_processes` 中指定的值不得超过环境类的可用线程数，如下所列：

- mw1.small — 其他进程不得超过 1 个线程。剩下的线程是为计划程序保留的。
- mw1.medium — 其他进程不得超过 3 个线程。剩下的线程是为计划程序保留的。
- mw1.large — 其他进程不得超过 7 个线程。剩下的线程是为计划程序保留的。

DAG 文件夹

Apache Airflow 计划程序会持续扫描环境中的 DAG 文件夹。任何包含的 `plugins.zip` 文件，或包含“Airflow”导入语句的 Python (`.py`) 文件。然后，所有生成的 Python DAG 对象都将放入该文件中，由调度器处理，以确定需要安排哪些任务（如果有）。DagBag无论文件是否包含任何可行的 DAG 对象，都会进行 DAG 文件解析。

参数

本节介绍可用于 DAG 文件夹的配置选项（Apache Airflow v2 和更高版本）及其使用案例。

Apache Airflow v3

配置	使用案例
dag_processor.refresh_interval 必须扫描 DAG 文件夹以查找新文件的秒数。 默认值：300 秒	您可以使用此选项通过增加解析 DAG 文件夹的秒数来释放资源。如果您发现 `total_parse_time metrics` 中的解析时间较长（这可能是由于 DAG 文件夹中有大量文件所致），建议您加大此值。
dag_processor.min_file_process_interval 反映计划程序解析 DAG 并更新 DAG 之后的秒数。 默认值：30 秒	您可以使用此选项通过增加计划程序在解析 DAG 之前等待的秒数来释放资源。例如，如果指定 `30` 的值，则将每 30 秒解析一次 DAG 文件。我们建议将此秒数保持在较高水平，以减小环境上的 CPU 使用率。

Apache Airflow v2

配置	使用案例
scheduler.dag_dir_list_interval 必须扫描 DAG 文件夹以查找新文件的秒数。 默认值：300 秒	您可以使用此选项通过增加解析 DAG 文件夹的秒数来释放资源。如果您发现 `total_parse_time metrics` 中的解析时间较长（

配置	使用案例
	这可能是由于 DAG 文件夹中有大量文件所致），建议您加大此值。
scheduler.min_file_process_interval 反映计划程序解析 DAG 并更新 DAG 之后的秒数。 默认值：30 秒	您可以使用此选项通过增加计划程序在解析 DAG 之前等待的秒数来释放资源。例如，如果指定 30 的值，则将每 30 秒解析一次 DAG 文件。我们建议将此秒数保持在较高水平，以减小环境上的 CPU 使用率。

DAG 文件

作为 Apache Airflow 计划程序循环的一部分，将解析单个 DAG 文件以提取 DAG Python 对象。在 Apache Airflow v2 及更高版本中，计划程序可以同时解析最大数量的解析进程。`scheduler.min_file_process_interval` (v2) 或 `dag_processor.min_file_process_interval` (v3) 中指定的秒数必须流逝后，才能再次解析同一个文件。

参数

本节介绍可用于 Apache Airflow DAG 文件的配置选项（Apache Airflow v2 和更高版本）及其使用案例。

Apache Airflow v3

配置	使用案例
dag_processor.dag_file_processor_timeout 处理 DAG 文件超DagFileProcessor时之前的秒数。 默认值：50 秒	您可以使用此选项通过增加超时之前所需的时间来释放资源。DagFileProcessor如果您在 DAG 处理日志中遇到超时导致无法加载可行的 DAG，我们建议您增大此值。
core.dagbag_import_timeout	

配置	使用案例
导入 Python 文件之前的秒数超时。 默认值：30 秒	在导入 Python 文件来提取 DAG 对象的同时延长计划程序超时之前所需的时间，您可以使用此选项来释放资源。此选项作为计划程序“循环”的一部分进行处理，并且必须包含一个小于 `dag_processor.dag_file_processor_timeout` 中指定值的值。
core.min_serialized_dag_update_interval 更新数据库中序列化的 DAG 之后的最小秒数。 默认值：30	通过增加更新数据库中序列化的 DAG 之后的秒数，您可以使用此选项来释放资源。如果您有大量 DAG 或 DAG 复杂，我们建议您增加此值。当 DAG 被序列化时，增大此值可减少计划程序和数据库的负载。
core.min_serialized_dag_fetch_interval 序列化的 DAG 已加载到数据库中时从数据库中重新提取的秒数。 DagBag 默认值：10	通过增加序列化的 DAG 重新提取的秒数，您可以使用此选项来释放资源。该值必须大于 `core.min_serialized_dag_update_interval` 中指定的值才能降低数据库的“写入”速率。当 DAG 被序列化时，增大此值可减少 Web 服务器和数据库的负载。

Apache Airflow v2

配置	使用案例
core.dag_file_processor_timeout 处理 DAG 文件超DagFileProcessor时之前的秒数。 默认值：50 秒	您可以使用此选项通过增加超时之前所需的时间来释放资源。DagFileProcessor如果您在 DAG 处理日志中遇到超时导致无法加载可行的 DAG，我们建议您增大此值。
core.dagbag_import_timeout	在导入 Python 文件来提取 DAG 对象的同时延长计划程序超时之前所需的时间，您可以使用

配置	使用案例
导入 Python 文件之前的秒数超时。 默认值：30 秒	此选项来释放资源。此选项作为计划程序“循环”的一部分进行处理，并且必须包含一个小于 `core.dag_file_processor_timeout` 中指定值的值。
core.min_serialized_dag_update_interval 更新数据库中序列化的 DAG 之后的最小秒数。 默认值：30	通过增加更新数据库中序列化的 DAG 之后的秒数，您可以使用此选项来释放资源。如果您有大量 DAG 或 DAG 复杂，我们建议您增加此值。当 DAG 被序列化时，增大此值可减少计划程序和数据库的负载。
core.min_serialized_dag_fetch_interval 序列化的 DAG 已加载到数据库中时从数据库中重新提取的秒数。 DagBag 默认值：10	通过增加序列化的 DAG 重新提取的秒数，您可以使用此选项来释放资源。该值必须大于 `core.min_serialized_dag_update_interval` 中指定的值才能降低数据库的“写入”速率。当 DAG 被序列化时，增大此值可减少 Web 服务器和数据库的负载。

任务

Apache Airflow 计划程序和工作线程都参与排队和出队任务。计划程序将已解析的准备调度的任务从无状态变为已计划状态。也在 Fargate 的计划程序容器上运行的执行程序，对这些任务进行排队并将其状态设置为已排队。当工作线程有容量时，它会从队列中提取任务并将状态设置为正在运行，然后根据任务成功还是失败将其状态更改为成功或失败。

参数

本节介绍可用于 Apache Airflow 任务的配置选项及其用例。

标记 Amazon MWAA 覆盖的默认配置选项。*red*

Apache Airflow v3

配置	使用案例

配置	使用案例
core.parallelism 状态为 Running 的最大任务实例数。 默认值：基于 (maxWorkers * maxCeleryWorkers) / schedulers * 1.5 动态设置。	通过增加可以同时运行的任务实例数，您可以使用此选项来释放资源。指定的值必须为可用工作线程数乘以工作线程任务密度。建议您仅在有大量任务处于“正在运行”或“已排队”状态时才更改此值。
core.execute_tasks_new_python_interpreter 确定 Apache Airflow 是通过分叉父进程还是通过创建新的 Python 进程来执行任务。 默认值：True	设置为 True 时，Apache Airflow 会将您对插件所做的更改识别为为执行任务而创建的新 Python 进程。
celery.worker_concurrency Amazon MWAA 会覆盖此选项的 Airflow 基础版安装，以作为自动扩缩组件的一部分扩缩工作线程。 默认值：不适用	*Any value specified for this option is ignored.*

配置	使用案例
celery.worker_autoscale 工作线程的任务并发度。 默认值： • mw1.micro - 3,0 • mw1.small - 5,0 • mw1.medium - 10,0 • mw1.large - 20,0 • mw1.xlarge – 40,0 • mw1.2xlarge – 80,0	通过降低工作线程的 minimum、maximum 任务并发度，您可以使用此选项来释放资源。无论是否有足够的资源，工作人员最多可以接受配置的maximum并发任务。如果在没有足够资源的情况下调度任务，则任务会立即失败。我们建议为资源密集型任务更改此值，方法是将该值减少到小于默认值，以允许每个任务有更多容量。

Apache Airflow v2

配置	使用案例
core.parallelism 状态为 Running 的最大任务实例数。 默认值：基于 (maxWorkers * maxCeleryWorkers) / schedulers * 1.5 动态设置。	通过增加可以同时运行的任务实例数，您可以使用此选项来释放资源。指定的值必须为可用工作线程数乘以工作线程任务密度。建议您仅在有大量任务处于“正在运行”或“已排队”状态时才更改此值。
core.dag_concurrency 允许为每个 DAG 同时运行的任务实例数。 默认值：10000	通过增加可以并发运行的任务实例数，您可以使用此选项来释放资源。例如，如果您有一百个 DAG 和十个并行任务，并且您希望所有 DAG 并发运行，则可以将最大并行度计算为可用工作线程数乘以 celery.worker_conc

配置	使用案例
	urrency 中的工作线程任务密度，除以 DAG 数量。
core.execute_tasks_new_python_interpreter 确定 Apache Airflow 是通过分叉父进程还是通过创建新的 Python 进程来执行任务。 默认值：True	设置为 True 时，Apache Airflow 会将您对插件所做的更改识别为为执行任务而创建的新 Python 进程。
celery.worker_concurrency Amazon MWAA 会覆盖此选项的 Airflow 基础版安装，以作为自动扩缩组件的一部分扩缩工作线程。 默认值：不适用	*Any value specified for this option is ignored.*
celery.worker_autoscale 工作线程的任务并发度。 默认值： • mw1.micro - 3,0 • mw1.small - 5,0 • mw1.medium - 10,0 • mw1.large - 20,0 • mw1.xlarge – 40,0 • mw1.2xlarge – 80,0	通过降低工作线程的 minimum、maximum 任务并发度，您可以使用此选项来释放资源。无论是否有足够的资源，工作人员最多可以接受配置的maximum并发任务。如果在没有足够资源的情况下调度任务，则任务会立即失败。我们建议为资源密集型任务更改此值，方法是将该值减少到小于默认值，以允许每个任务有更多容量。

在 requirements.txt 中管理 Python 依赖项

本主题介绍了在 `requirements.txt` 文件中如何为 Amazon MWAA 环境安装和管理 Python 依赖项。

目录

使用 Amazon MWAA CLI 实用工具测试 DAG

- 命令行界面 (CLI) 实用工具可在本地复制 Amazon MWAA 环境。
- CLI 在本地构建 Docker 容器镜像，类似于 Amazon MWAA 生产镜像。您可以使用它运行本地 Apache Airflow 环境来开发和测试 DAG、自定义插件和依赖项，然后部署到 Amazon MWAA。
- 要运行 CLI，请参阅上的 aws-mwaa-docker-images。 GitHub

使用 PyPi.org 需求文件格式安装 Python 依赖项

以下部分介绍了根据 PyPi.org 需求文件格式安装 Python 依赖项的不同方法。

选项一：来自 Python 程序包索引的 Python 依赖关系

下一节介绍如何在 `requirements.txt` 文件中指定 Python 程序包索引中的 Python 依赖项。

Apache Airflow v3

1. 本地测试。在创建 `requirements.txt` 文件之前，以迭代方式添加其他库以找到程序包及其版本的正确组合。要运行 Amazon MWAA CLI 实用程序，请参阅上的 aws-mwaa-docker-images。 GitHub
2. 查看 Apache Airflow 程序包的 Extras。要访问亚马逊 MWAA 上为 Apache Airflow v3 安装的软件包列表，请参阅网站上的 aws-mwaa-docker-images。`requirements.txt` GitHub
3. 添加约束语句。在文件顶部添加 Apache Airflow v3 环境的约束文件。`requirements.txt`Apache Airflow 约束文件指定了 Apache Airflow 发布时可用的提供程序版本。

   在以下示例中，用环境的版本号替换 *`{environment-version}`* 以及用与环境兼容的 Python 版本替换 *`{Python-version}`*。

   有关与 Apache Airflow 环境兼容的 Python 版本的信息，请参阅 Apache Airflow 版本。

   ```
   --constraint "https://raw.githubusercontent.com/apache/airflow/
   constraints-{Airflow-version}/constraints-{Python-version}.txt"
   ```

   如果约束文件确定该 `xyz==1.0` 程序包与环境中的其他程序包不兼容，则 `pip3 install` 将无法阻止不兼容的库安装到环境中。如果任何软件包的安装失败，则可以在日志的相应日志流中访问每个 Apache Airflow 组件（调度程序、工作程序和 Web 服务器）的错误日志。CloudWatch 有关日志类型的更多信息，请参阅 the section called “访问 Airflow 日志”。
4. Apache Airflow 程序包。添加程序包 Extras及其版本 (==)。这有助于防止在环境中安装同名但版本不同的程序包。

   ```
   apache-airflow[package-extra]==2.5.1
   ```
5. Python 库。在 `requirements.txt` 文件中添加程序包名称和版本 (==)。这有助于防止自动应用 future PyPi.org的重大更新。

   ```
   library == version
   ```

   Example boto3 和 psycopg2-binary

   此示例仅用于演示目的。boto 和 psycopg2-binary 库包含在 Apache Airflow v3 基础版安装中，无需在 `requirements.txt` 文件中指定。

```
boto3==1.17.54
boto==2.49.0
botocore==1.20.54
psycopg2-binary==2.8.6
```

如果指定的包没有版本，则 Amazon MWAA 会从中安装该软件包的最新版本。PyPi.org此版本可能与您 `requirements.txt` 中的其他程序包冲突。

Apache Airflow v2

1. 本地测试。在创建 `requirements.txt` 文件之前，以迭代方式添加其他库以找到程序包及其版本的正确组合。要运行 Amazon MWAA CLI 实用程序，请参阅上的 aws-mwaa-docker-images。 GitHub
2. 查看 Apache Airflow 程序包的 Extras。要访问亚马逊 MWAA 上为 Apache Airflow v2 安装的软件包列表，请访问网站上的 aws-mwaa-docker-images。`requirements.txt` GitHub
3. 添加约束语句。在 `requirements.txt` 文件顶部添加 Apache Airflow v2 环境的约束文件。Apache Airflow 约束文件指定了 Apache Airflow 发布时可用的提供程序版本。

   从 Apache Airflow v2.7.2 开始，要求文件必须包含一条 `--constraint` 语句。如果您未提供约束条件，Amazon MWAA 将为您指定一个约束条件，以确保您的要求中列出的程序包与您正在使用的 Apache Airflow 版本兼容。

   在以下示例中，用环境的版本号替换 *`{environment-version}`* 以及用与环境兼容的 Python 版本替换 *`{Python-version}`*。

   有关与 Apache Airflow 环境兼容的 Python 版本的信息，请参阅 Apache Airflow 版本。

   ```
   --constraint "https://raw.githubusercontent.com/apache/airflow/
   constraints-{Airflow-version}/constraints-{Python-version}.txt"
   ```

   如果约束文件确定该 `xyz==1.0` 程序包与环境中的其他程序包不兼容，则 `pip3 install` 将无法阻止不兼容的库安装到环境中。如果任何软件包的安装失败，则可以在日志的相应日志流中访问每个 Apache Airflow 组件（调度程序、工作程序和 Web 服务器）的错误日志。CloudWatch 有关日志类型的更多信息，请参阅 the section called “访问 Airflow 日志”。
4. Apache Airflow 程序包。添加程序包 Extras及其版本 (==)。这有助于防止在环境中安装同名但版本不同的程序包。

```
apache-airflow[package-extra]==2.5.1
```

5. Python 库。在 `requirements.txt` 文件中添加程序包名称和版本 (==)。这有助于防止自动应用 future PyPi.org的重大更新。

```
library == version
```

Example boto3 和 psycopg2-binary

此示例仅用于演示目的。boto 和 psycopg2-binary 库包含在 Apache Airflow v2 基础版安装中，无需在 `requirements.txt` 文件中指定。

```
boto3==1.17.54
boto==2.49.0
botocore==1.20.54
psycopg2-binary==2.8.6
```

如果指定的包没有版本，则 Amazon MWAA 会从中安装该软件包的最新版本。PyPi.org此版本可能与您 `requirements.txt` 中的其他程序包冲突。

选项二：Python Wheel（.whl）

Python Wheel 是一种程序包格式，旨在发布包含已编译构件的库。将 Wheel 程序包作为在 Amazon MWAA 中安装依赖项的方法有几个好处：

- 安装速度更快 — WHL 文件作为单个 ZIP 文件复制到容器中，然后安装到本地，无需下载每个文件。
- 减少冲突-您可以提前确定程序包的版本兼容性。因此，无需 `pip` 以递归方式计算出兼容版本。
- 更高的弹性 — 对于外部托管的库，下游要求可能会发生变化，从而导致 Amazon MWAA 环境中容器之间的版本不兼容。由于不依赖外部来源来获取依赖项，因此无论每个容器何时实例化，其上每个容器都有相同的库。

我们建议使用以下方法来安装 `requirements.txt` 中来自 Python Wheel 档案 (`.whl`) 的 Python 依赖项。

方法

- 在亚马逊 S3 存储桶上使用 plugins.zip 文件

- 使用托管在 URL 上的 WHL 文件
- 从 DAG 创建 WHL 文件

在亚马逊 S3 存储桶上使用 **plugins.zip** 文件

Apache Airflow 调度程序、工作程序和网络服务器（适用于 Apache Airflow v2.2.2 及更高版本）在启动期间在您的环境的托管的 Fargate 容器上搜索自定义插件，网址为。 AWS/usr/local/airflow/plugins/*此过程在 Python 依赖项的 Amazon MWAA 的 pip3 install -r requirements.txt 和 Apache Airflow 服务启动之前开始。plugins.zip 文件用于您在环境执行期间不想持续更改的任何文件，或者您可能不想向编写 DAG 的用户授予访问权限的任何文件。例如，Python 库 Wheel 文件、证书 PEM 文件和配置 YAML 文件。

下一节介绍如何在 Amazon S3 存储桶上安装 plugins.zip 文件中的 Wheel。

1. 下载必要的 WHL 文件，您可以在 Amazon MWAA aws-mwaa-docker-images 或另一个 Amazon Linux 2 容器上，将 pip download 和现有 requirements.txt 一起使用来解析和下载必要的 Python Wheel 文件。

   ```
   pip3 download -r "$AIRFLOW_HOME/dags/requirements.txt" -d "$AIRFLOW_HOME/plugins"
   cd "$AIRFLOW_HOME/plugins"
   zip "$AIRFLOW_HOME/plugins.zip" *
   ```

2. 在 **requirements.txt** 中指定路径。使用 --find-links 指定位于 requirements.txt 顶部的插件目录，并指示 pip 不要使用 --no-index 从其他来源安装，如下面的代码所列：

   ```
   --find-links /usr/local/airflow/plugins
   --no-index
   ```

   Example在 requirements.txt 中的 wheel

   以下示例假设您已将 Wheel 上传到 Amazon S3 存储桶根目录中的 plugins.zip 文件中。例如：

   ```
   --find-links /usr/local/airflow/plugins
   --no-index

   numpy
   ```

Amazon MWAA 从 `plugins` 文件夹中提取 `numpy-1.20.1-cp37-cp37m-manylinux1_x86_64.whl` Wheel 并将其安装到环境中。

使用托管在 URL 上的 WHL 文件

下一节将介绍如何安装托管在 URL 上的 Wheel。此 URL 必须是可公开访问的，或者可以从您为 Amazon MWAA 环境指定的自定义 Amazon VPC 中访问。

- 提供 URL。向 `requirements.txt` 中的 Wheel 提供 URL。

  Example公有 URL 上的 Wheel 档案

  以下示例从公有站点下载 Wheel。

  ```
  --find-links https://files.pythonhosted.org/packages/
  --no-index
  ```

  Amazon MWAA 从您指定的 URL 中获取 Wheel 并将其安装到环境中。

  > **Note**
  >
  > 从 Amazon MWAA v2.2.2 及更高版本中安装要求的私有 Web 服务器上无法访问 URL。

从 DAG 创建 WHL 文件

如果您的私有网络环境使用 Apache Airflow v2.2 或更高版本，并且由于您的环境无法访问外部存储库而无法安装要求，则可以使用以下 DAG 来获取现有的 Amazon MWAA 要求并将其打包到 Amazon S3 上：

```
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://amzn-s3-demo-bucket/{S3_KEY}"
)
```

运行 DAG 后，使用这个新文件作为 Amazon MWAA `plugins.zip`，也可以选择与其他插件一起打包。然后通过在前面添加 `--find-links /usr/local/airflow/plugins` 和 `--no-index`，但不添加 `--constraint` 来更新 `requirements.txt`。

此方法允许您离线使用相同的库。

选项三：托管在私有 PyPi/PEP-503 兼容存储库上的 Python 依赖项

下一节介绍如何安装托管在私有 URL 上且经过身份验证的 Apache Airflow Extra。

1. 将用户名和密码添加为 Apache Airflow 配置选项。例如：
   - `foo.user`：*YOUR_USER_NAME*
   - `foo.pass`：*YOUR_PASSWORD*
2. 创建 `requirements.txt` 文件 用私有 URL 以及作为 Apache Airflow 配置选项添加的用户名和密码替换以下示例中的占位符。例如：

   ```
   --index-url https://${AIRFLOW__FOO__USER}:${AIRFLOW__FOO__PASS}@my.privatepypi.com
   ```
3. 将任何其他库添加到 `requirements.txt` 文件中。例如：

   ```
   --index-url https://${AIRFLOW__FOO__USER}:${AIRFLOW__FOO__PASS}@my.privatepypi.com
   my-private-package==1.2.3
   ```

在 Amazon MWAA 控制台上启用日志

您的 Amazon MWAA 环境的执行角色需要权限才能将日志发送到日志。 CloudWatch 要更新执行角色的权限，请参阅 Amazon MWAA 执行角色。

您可以启用 `INFO`、`WARNING`、`ERROR` 或 `CRITICAL` 级别的 Apache Airflow 日志。当您选择日志级别时，Amazon MWAA 会发送该级别和所有更高级别的严重性级别的日志。例如，如果您在`INFO`级别启用日志，Amazon MWAA 会向`INFO`日志发送日志`WARNING`、`ERROR`、和`CRITICAL`日

志级别。 CloudWatch 我们建议启用 INFO 级别的 Apache Airflow 日志，以便计划程序访问为 requirements.txt 收到的日志。

在日志控制台上访问 CloudWatch 日志

您可以访问调度工作流程并解析 dags 文件夹的计划程序的 Apache Airflow 日志。以下步骤介绍如何在 Amazon MWAA 控制台上打开计划程序的日志组，以及如何在日志控制台上访问 Apache Airflow 日志。 CloudWatch

访问 **requirements.txt** 的日志

1. 在 Amazon MWAA 控制台上打开环境页面。
2. 选择环境。
3. 在监控窗格上选择 Airflow 计划程序日志组。
4. 在日志流中选择 requirements_install_ip 日志。
5. 请参阅 /usr/local/airflow/.local/bin 上环境中安装的程序包列表。例如：

```
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. 查看程序包列表以及其中任何程序包在安装过程中是否遇到错误。如果出现问题，您可能会收到类似以下内容的错误：

```
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))
```

在 Apache Airflow UI 中访问错误

您可能还需要检查 Apache Airflow UI，以确定错误是否与其他问题有关。在 Amazon MWAA 上使用 Apache Airflow 时，您可能遇到的最常见错误是：

```
Broken DAG: No module named x
```

如果您在 Apache Airflow UI 中发现此错误，则 `requirements.txt` 文件中可能缺少必需的依赖项。

登录 Apache Airflow

你需要你的 AWS 账户 内部 AWS Identity and Access Management (IAM) Apache Airflow 用户界面访问策略： AmazonMWAAWebServerAccess 权限才能访问你的 Apache Airflow 用户界面。

要访问 Apache Airflow UI，请执行以下操作

1. 在 Amazon MWAA 控制台上打开环境页面。
2. 选择环境。
3. 选择打开 Airflow UI。

`requirements.txt` 场景示例

您可以在 `requirements.txt` 中混合搭配不同的格式。以下示例使用不同方式的组合来安装 Extras。

Example额外内容 PyPi.org 和公共网址

除了在公共 URL 上指定软件包（例如符合 PEP 503 的自定义 repo URL）之外，您还需要使用该`--index-url`选项。 PyPi.org

```
aws-batch == 0.6
    phoenix-letter >= 0.3

    --index-url http://dist.repoze.org/zope2/2.10/simple
    zopelib
```

Amazon MWAA 的监控和指标

监控是维护适用于 Apache Airflow 的亚马逊托管工作流程和您的解决方案的可靠性、可用性和性能的重要组成部分。 AWS 我们建议从 AWS 解决方案的各个部分收集监控数据，以便在出现多点故障时可以更轻松地进行调试。本主题 AWS 介绍用于监控您的 Amazon MWAA 环境和响应潜在事件的资源。

> **Note**
>
> Apache Airflow 指标和日志记录受亚马逊标准定价的约束。 CloudWatch

有关监控 Apache Airflow 的更多信息，请参阅 Apache Airflow 文档网站中的日志和监控。

Sections

- Amazon MWAA 上的监控概述
- 访问审核日志 AWS CloudTrail
- 访问 Amazon 中的 Airflow 日志 CloudWatch
- 监控 Amazon MWAA 上的控制面板和警报
- 中的 Apache Airflow 环境指标 CloudWatch
- Amazon MWAA 的容器、队列和数据库指标

Amazon MWAA 上的监控概述

本页介绍用于监控适用于 Apache Airflow 的亚马逊托管工作流程环境的 AWS 服务。

目录

亚马逊 CloudWatch 概述

CloudWatch 是一个 AWS 服务指标存储库，可用于根据服务发布的指标和维度检索统计数据。您可以使用这些指标来配置警报、计算统计数据，然后在控制面板中显示数据，以帮助您在 Amazon CloudWatch 控制台中评估环境的运行状况。

Apache Airflow 已经设置为向亚马逊发送适用于 Apache Airflow 的亚马逊托管工作流程环境的 StatSD 指标。 CloudWatch

要了解更多信息，请参阅什么是亚马逊 CloudWatch？ 。

AWS CloudTrail 概览

CloudTrail 是一项审计服务，用于记录用户、角色或 AWS 服务在 Amazon MWAA 中执行的操作。使用收集到的信息 CloudTrail，您可以确定向 Amazon MWAA 发出的请求、发出请求的 IP 地址、谁发出了请求、何时提出请求，以及审计日志中提供的其他详细信息。

要了解更多信息，请参阅什么是 AWS CloudTrail？ 。

访问审核日志 AWS CloudTrail

AWS CloudTrail 在你创建 AWS 账户 时已在你上启用。 CloudTrail 记录 IAM 实体或 AWS 服务（例如适用于 Apache Airflow 的亚马逊托管工作流程）所进行的活动，该活动被记录为事件。 CloudTrail 您可以在 CloudTrail 控制台中查看、搜索和下载过去 90 天的事件历史记录。 CloudTrail 捕获亚马逊 MWAA 控制台上的所有事件以及对亚马逊 MWAA API 的所有调用。但不会捕获只读操作（例如 `GetEnvironment` 或 `PublishMetrics` 动作）。本页介绍 CloudTrail 如何使用监控 Amazon MWAA 的事件。

目录

在中创建跟踪 CloudTrail

您需要创建跟踪才能访问您中的持续事件记录 AWS 账户，包括 Amazon MWAA 的事件。跟踪允许 CloudTrail 将日志文件传输到 Amazon S3 存储桶。如果您不创建跟踪，您仍然可以在 CloudTrail 控制台中访问可用的事件历史记录。例如，使用收集的信息 CloudTrail，您可以确定向 Amazon MWAA 发出的请求、发出请求的 IP 地址、谁发出了请求、何时发出请求以及其他详细信息。要了解更多信息，请参阅为您的 AWS 账户创建跟踪。

使用事件历史记录访问 CloudTrail 事件

您可以通过查看事件历史记录，在 CloudTrail 控制台中对过去 90 天的操作和安全事件进行故障排除。例如，您可以按区域访问与创建、修改或删除您的 AWS 账户 资源（例如 IAM 用户或其他 AWS 资源）相关的事件。要了解更多信息，请参阅使用事件历史记录访问 CloudTrail 事件。

1. 打开 CloudTrail 控制台。
2. 选择事件历史记录。
3. 选择要查看的事件，然后选择比较事件详细信息。

`CreateEnvironment` 的示例跟踪

跟踪记录是一种配置，可用于将事件作为日志文件传送到您指定的 Simple Storage Service（Amazon S3）存储桶。

CloudTrail 日志文件包含一个或多个日志条目。事件代表来自任何来源的单个请求，包括有关所请求操作的信息，例如操作的日期和时间或请求参数。 CloudTrail 日志文件不是公共 API 调用的有序堆栈跟踪，也没有按任何特定顺序列出。以下示例是由于缺乏权 限而被拒绝的 `CreateEnvironment` 操作的日志条目。为了保护隐私，`AirflowConfigurationOptions` 中的值已被删除。

```
{
  "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"
}
```


接下来做什么？

- 在CloudTrail 支持的 AWS 服务和集成中，了解如何为 CloudTrail 日志中收集的事件数据配置其他服务。
- 要了解如何在向 Amazon S3 存储桶 CloudTrail 发布新日志文件时收到通知，请参阅为其配置 Amazon SNS 通知。 CloudTrail

访问 Amazon 中的 Airflow 日志 CloudWatch

亚马逊 MWAA 可以向亚马逊发送 Apache Airflow 日志。 CloudWatch您可以从一个位置访问多个环境的日志，从而轻松识别 Apache Airflow 任务延迟或工作流程错误，而无需其他第三方工具。需要在适用于 Apache Airflow 的亚马逊托管工作流程控制台上启用 Apache Airflow 日志，才能访问 Apache Airflow DAG 处理、任务、网络服务器、工作人员登录。 CloudWatch

目录

定价

- 收取标准 CloudWatch 日志费用。有关更多信息，请参阅 CloudWatch 定价。

开始前的准备工作

- 您必须拥有可以访问登录的角色 CloudWatch。有关更多信息，请参阅访问 Amazon MWAA 环境。

日志类型

Amazon MWAA 会为您启用的每个 Airflow 日志选项创建一个日志组，并将日志推送到与环境关联的 CloudWatch 日志组。日志组以 `YourEnvironmentName-`**`LogType`** 格式命名。例如，如果环境名为 `Airflow-v202-Public`，则 Apache Airflow 任务日志将发送到 `Airflow-v202-Public-`**`Task`**。

日志类型	说明
`YourEnvironmentName-` **`DAGProcessing`**	DAG 处理器管理器（计划程序中处理 DAG 文件的部分）的日志。

日志类型	说明
YourEnvironmentName- **Scheduler**	Airflow 计划程序生成的日志。
YourEnvironmentName- **Task**	DAG 生成的任务日志。
YourEnvironmentName- **WebServer**	Airflow Web 界面生成的日志。
YourEnvironmentName- **Worker**	作为工作流程和 DAG 执行的一部分生成的日志。

启用 Apache Airflow 日志

您可以启用 INFO、WARNING、ERROR 或 CRITICAL 级别的 Apache Airflow 日志。当您选择日志级别时，Amazon MWAA 会发送该级别和所有更高级别的严重性级别的日志。例如，如果您在INFO级别启用日志，Amazon MWAA 会向INFO日志发送日志WARNING、ERROR、和CRITICAL日志级别。CloudWatch

1. 在 Amazon MWAA 控制台上打开环境页面。
2. 选择环境。
3. 选择编辑。
4. 选择下一步。
5. 选择下列一个或多个选项：
    a. 在监控窗格上选择 Airflow 计划程序日志组。
    b. 在监控窗格上选择 Airflow Web 服务器日志组。
    c. 在监控窗格上选择 Airflow 工作线程日志组。
    d. 在监控窗格上选择 Airflow DAG 处理日志组。
    e. 在监控窗格上选择 Airflow 任务日志组。
    f. 在 日志级别中选择日志级别。
6. 选择下一步。
7. 选择保存。

访问 Apache Airflow 日志

以下部分介绍如何在控制台中访问 Apache Airflow 日志。 CloudWatch

1. 在 Amazon MWAA 控制台上打开环境页面。
2. 选择环境。
3. 在监控窗格中选择一个日志组。
4. 在日志流中选择日志。

示例计划程序日志

您可以访问调度工作流程并解析 dags 文件夹的计划程序的 Apache Airflow 日志。以下步骤介绍如何在 Amazon MWAA 控制台上打开计划程序的日志组，以及如何在日志控制台上访问 Apache Airflow 日志。 CloudWatch

访问 **requirements.txt** 的日志

1. 在 Amazon MWAA 控制台上打开环境页面。
2. 选择环境。
3. 在监控窗格上选择 Airflow 计划程序日志组。
4. 在日志流中选择 requirements_install_ip 日志。
5. 请参阅 /usr/local/airflow/.local/bin 上环境中安装的程序包列表。例如：

```
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. 查看程序包列表以及其中任何程序包在安装过程中是否遇到错误。如果出现问题，您可能会收到类似以下内容的错误：

```
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))
```

接下来做什么？

- 要了解如何配置 CloudWatch 警报，请参阅使用 Amazon CloudWatch 警报。
- 要了解如何创建 CloudWatch 仪表板，请参阅使用 CloudWatch 仪表板。

监控 Amazon MWAA 上的控制面板和警报

您可以在亚马逊 CloudWatch 创建自定义控制面板，并为特定指标添加警报，以监控适用于 Apache Airflow 的亚马逊托管工作流程环境的运行状况。当某个警报位于控制面板上且处于 ALARM 状态，则会变成红色，便于主动监控 Amazon MWAA 的运行状况。

Apache Airflow 公开了几个进程的指标，包括 DAG 进程数、DAG 程序包的大小、当前正在运行的任务、任务失败和成功。当您创建环境时，Airflow 会自动将亚马逊 MWAA 环境的指标发送到。CloudWatch本页介绍如何为 Amazon MWAA 环境中的 CloudWatch Airflow 指标创建运行状况控制面板。

目录

指标

您可以为 Apache Airflow 版本的任何可用指标创建自定义控制面板和警报。每个指标都对应一个 Apache Airflow 关键性能指标（KPI）。要访问指标列表，请参阅：

- 中的 Apache Airflow 环境指标 CloudWatch

警报状态概述

指标告警可能具有以下几种状态：

- OK – 指标或表达式在定义的阈值范围内。
- ALARM – 指标或表达式超出定义的阈值。
- INSUFFICIENT_DATA（数据不足）– 告警刚刚启动，指标不可用，或者指标没有足够的数据以确定告警状态。

自定义控制面板和警报示例

您可以构建自定义监控控制面板，显示 Amazon MWAA 环境所选指标的图表。

关于这些指标

以下列表描述了通过本节中的教程和模板定义在自定义控制面板中创建的每个指标。

- QueuedTasks-处于队列状态的任务数。对应于 `executor.queued_tasks` Apache Airflow 指标。
- TasksPending-执行器中待处理的任务数。对应于 `scheduler.tasks.pending` Apache Airflow 指标。

> **Note**
>
> 不适用于 Apache Airflow v2.2 及更高版本。

- RunningTasks-在执行器中运行的任务数。对应于 `executor.running_tasks` Apache Airflow 指标。
- SchedulerHeartbeat-Apache Airflow 在调度程序作业中执行的签到次数。与 `scheduler_heartbeat` Apache Airflow 指标相对应。
- TotalParseTime-一次扫描和导入所有 DAG 文件所花费的秒数。对应于 `dag_processing.total_parse_time` Apache Airflow 指标。

关于控制面板

下图显示了根据本节中的教程和模板定义创建的监控面板。

使用 AWS 教程

您可以使用以下 AWS 教程为当前部署的任何 Amazon MWAA 环境自动创建运行状况控制面板。它还会针对所有 Amazon MWAA 环境中的不健康工作人员和计划程序心跳故障创建 CloudWatch 警报。

- CloudWatch 亚马逊 MWAA 控制面板自动化

使用 CloudFormation

您可以使用本节中的 CloudFormation 模板定义在中创建监控面板 CloudWatch，然后在 CloudWatch 控制台上添加警报，以便在指标超过特定阈值时接收通知。要使用此模板定义创建堆栈，请参阅在CloudFormation 控制台上创建堆栈。要向控制面板添加警报，请参阅使用警报。

```
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
                    }
                }
            ]
        }'
```

删除指标和控制面板

如果您删除 Amazon MWAA 环境，相应的控制面板也会被删除。 CloudWatch 指标存储十五 (15) 个月，无法删除。 CloudWatch 控制台将指标的搜索限制在上次采集指标后的两 (2) 周内，以确保显示您的 Amazon MWAA 环境的最新实例。要了解更多信息，请参阅 Amazon CloudWatch 常见问题。

接下来做什么？

- 了解如何创建 DAG 来查询您的环境的 Amazon Aurora PostgreSQL 元数据数据库并将自定义指标发布到中。 CloudWatch 使用 DAG 在 CloudWatch 中编写自定义指标

中的 Apache Airflow 环境指标 CloudWatch

Apache Airflow v2 和 v3 已经设置为收集适用于 Apache Airflow 的亚马逊托管工作流程环境的 StatSD 指标并将其发送到亚马逊。 CloudWatchApache Airflow 发送的指标的完整列表可在《Apache Airflow 参考指南》的 指标页面上找到。本页介绍了中可用的 Apache Airflow 指标以及如何在控制台 CloudWatch 中访问指标。 CloudWatch

目录

术语

命名空间

命名空间是 AWS 服务 CloudWatch 指标的容器。对于Amazon MWAA，命名空间为 AmazonMWAA。

CloudWatch 指标

CloudWatch 指标表示特定于的一组按时间顺序排列的数据点。 CloudWatch

Apache Airflow 指标

特定于 Apache Airflow 的指标。

维度

维度是作为指标标识一部分的 name/value 配对。

单位

所有统计数据都有度量单位。对于 Amazon MWAA，单位包括计数、秒和毫秒。对于 Amazon MWAA，单位是根据原始 Airflow 指标中的单位设置的。

Dimensions

本节介绍中 Apache Airflow 指标的 CloudWatch 维度分组。 CloudWatch

维度	说明
DAG	表示特定的 Apache Airflow DAG 名称。
DAG 文件名	表示特定的 Apache Airflow DAG 文件名称。
函数	此维度用于改进中的指标分组 CloudWatch。
任务	表示计划程序运行的 Apache Airflow 任务。始终具有 `Job` 的值。
运算符	表示特定的 Apache Airflow 运算符。
池	表示特定的 Apache Airflow 工作线程池。
Task	表示特定的 Apache Airflow 任务。
HostName	表示正在运行的特定的 Apache Airflow 进程的主机名。

在 CloudWatch 控制台中访问指标

本节介绍如何在中访问特定 DAG CloudWatch 的性能指标。

访问维度的性能指标

1. 在 CloudWatch 控制台上打开 “指标” 页面。
2. 选择你的 AWS 区域。
3. 选择 AmazonMWAA 命名空间。
4. 在所有指标选项卡中，选择一个维度。例如，DAG、环境。

5. 为维 CloudWatch 度选择一个指标。例如，TaskInstanceSuccesses 或 TaskInstanceDuration。选择绘制所有搜索结果的图表。
6. 选择图表化指标选项卡可访问 Apache Airflow 指标的性能统计信息，例如 DAG、环境、任务。

Apache Airflow 指标可用于 CloudWatch

本节介绍发送到的 Apache Airflow 指标和维度。 CloudWatch

Apache Airflow 计数器

本节中的 Apache Airflow 指标包含有关 Apache Airflow 计数器的数据。

CloudWatch 指标	Apache Airflow 指标	单位	维度	
SLAMissed Note 仅适用于 Apache Airflow v2.4.3 到 v2.10.3。	sla_missed	计数	函数，计划程序	
FailedSLACallback Note 仅适用于 Apache Airflow v2.4.3 到 v2.10.3。	sla_callback_notification_failure	计数	函数，计划程序	
更新	dataset.updates	计数	函数，计划程序	

CloudWatch 指标	Apache Airflow 指标	单位	维度	
Note 适用于 Apache Airflow v2.6.3 及更高版本。				
孤立 Note 适用于 Apache Airflow v2.6.3 及更高版本。	dataset.orphaned	计数	函数，计划程序	
FailedCeleryTaskExecution Note 适用于 Apache Airflow v2.4.3 及更高版本。	celery.execute_command.failure	计数	函数，Celery	
FilePathQueueUpdateCount Note 适用于 Apache Airflow v2.6.3 及更高版本。	dag_processing.file_path_queue_update_count	计数	函数，计划程序	
CriticalSectionBusy	scheduler.critical_section_busy	计数	函数，计划程序	

CloudWatch 指标	Apache Airflow 指标	单位	维度	
DagBagSize	dagbag_size	计数	函数，DAG 处理	
DagCallbackExceptions	dag.callback_exceptions	计数	DAG，全部	
FailedSLAEmailAttempts Note 适用于 Apache Airflow v3.0.6 及更高版本。	sla_email_notification_failure	计数	函数，计划程序	
TaskInstanceFinished	ti.finish.{dag_id}.{task_id}.{state}	计数	DAG, {dag_id} 任务，{task_id} 状态，{state}	
JobEnd	{job_name}_end	计数	任务，{job_name}	
JobHeartbeatFailure	{job_name}_heartbeat_failure	计数	任务，{job_name}	
JobStart	{job_name}_start	计数	任务，{job_name}	

CloudWatch 指标	Apache Airflow 指标	单位	维度	
ManagerStalls	dag_processing.manager_stalls	计数	函数，DAG 处理	
OperatorFailures	operator_failures_{operator_name}	计数	运算符，{operator_name}	
OperatorSuccesses	operator_successes_{operator_name}	计数	运算符，{operator_name}	
OtherCallbackCount Note 在 Apache Airflow v2.6.3 及更高版本中可用。	dag_processing.other_callback_count	计数	函数，计划程序	
进程	dag_processing 进程	计数	函数，DAG 处理	
SchedulerHeartbeat	scheduler_heartbeat	计数	函数，计划程序	

CloudWatch 指标	Apache Airflow 指标	单位	维度	
StartedTaskInstances	ti.start.{dag_id}.{task_id}	计数	DAG，全部 任务，全部	
SlaCallbackCount	dag_processing.sla_callback_count Note 适用于 Apache Airflow v2.6.3 及更高版本。	计数	函数，计划程序	
TasksKilledExternally	scheduler.tasks.killed_externally	计数	函数，计划程序	
TaskTimeoutError	celery.task_timeout_error	计数	函数，Celery	

CloudWatch 指标	Apache Airflow 指标	单位	维度	
TaskInstanceCreatedUsingOperator	task_instance_created-{operator_name}	计数	运算符，{operator_name}	
TaskInstancePreviouslySucceeded	previously_succeeded	计数	DAG，全部 任务，全部	
TaskInstanceFailures	ti_failures	计数	DAG，全部 任务，全部	
TaskInstanceSuccesses	ti_successes	计数	DAG，全部 任务，全部	
TaskRemovedFromDAG	task_removed_from_dag.{dag_id}	计数	DAG，{dag_id}	
TaskRestoredToDAG	task_restored_to_dag.{dag_id}	计数	DAG，{dag_id}	
TriggersSucceeded Note 适用于 Apache Airflow v2.7.2 及更高版本。	triggers.succeeded	计数	函数，触发	

CloudWatch 指标	Apache Airflow 指标	单位	维度	
TriggersFailed Note 适用于 Apache Airflow v2.7.2 及更高版本。	triggers.failed	计数	函数，触发	
TriggersBlockedMainThread Note 适用于 Apache Airflow v2.7.2 及更高版本。	triggers.blocked_main_thread	计数	函数，触发	
TriggerHeartbeat Note 适用于 Apache Airflow v2.8.1 及更高版本。	triggerer_heartbeat	计数	函数、触发器	

CloudWatch 指标	Apache Airflow 指标	单位	维度
TaskInstanceCreatedUsingOperator	airflow.task_instance_created_{operator_name} Note 适用于 Apache Airflow v2.7.2 及更高版本。	计数	运算符，{operator_name}
ZombiesKilled	zombies_killed	计数	DAG，全部 任务，全部

Apache Airflow 计

本节中的 Apache Airflow 指标包含有关 Apache Airflow 计的数据。

CloudWatch 指标	Apache Airflow 指标	单位	维度
DAGFileRefreshError	dag_file_refresh_error	计数	函数，DAG 处理

CloudWatch 指标	Apache Airflow 指标	单位	维度
ImportErrors	dag_processing.import_errors	计数	函数，DAG 处理
Exception Failures	smart_sensor_operator.exception_failures	计数	函数，智能传感器运算符
ExecutedTasks	smart_sensor_operator.executed_tasks	计数	函数，智能传感器运算符
InfraFailures	smart_sensor_operator.infailes	计数	函数，智能传感器运算符
LoadedTasks	smart_sensor_operator.loaded_tasks	计数	函数，智能传感器运算符
TotalParseTime	dag_processing.total_parse_time	秒	函数，DAG 处理
TriggeredDagRuns Note 在 Apache Airflow v2.6.3 及更高版本中可用。	dataset.triggered_dagruns	计数	函数，计划程序

CloudWatch 指标	Apache Airflow 指标	单位	维度	
TriggersRunning Note 在 Apache Airflow v2.7.2 及更高版本中可用。	triggers.runn。*{hostname}*	计数	函数，触发 HostName, *{hostname}*	
PoolDeferredSlots Note 在 Apache Airflow v2.7.2 及更高版本中可用。	pool.deferred_slots.{pool_name}	计数	池，{pool_name}	
DAGFileProcessingLastRunSecondsAgo	dag_processing.last_run.seconds_ago.{dag_filename}	秒	DAG 文件名，{dag_filename}	

CloudWatch 指标	Apache Airflow 指标	单位	维度	
OpenSlots	executor.open_slots	计数	函数，执行程序	
OrphanedTasksAdopted	scheduler.orphaned_tasks.adopted	计数	函数，计划程序	
OrphanedTasksCleared	scheduler.orphaned_tasks.cleared	计数	函数，计划程序	
PokedExceptions	smart_sensor_operator.poked_exception	计数	函数，智能传感器运算符	
PokedSuccess	smart_sensor_operator.poked_success	计数	函数，智能传感器运算符	
PokedTasks	smart_sensor_operator.poked_tasks	计数	函数，智能传感器运算符	
PoolFailures	pool.open_slots.{pool_name}	计数	池，{pool_name}	
PoolStarvingTasks	pool.starving_tasks.{pool_name}	计数	池，{pool_name}	
PoolOpenSlots	pool.open_slots.{pool_name}	计数	池，{pool_name}	
PoolQueuedSlots	pool.queued_slots.{pool_name}	计数	池，{pool_name}	

CloudWatch 指标	Apache Airflow 指标	单位	维度
PoolRunningSlots	pool.running_slots.{pool_name}	计数	池，{pool_name}
ProcessorTimeouts	dag_processing.processor_timeouts	计数	函数，DAG 处理
QueuedTasks	executor.queued_tasks	计数	函数，执行程序
RunningTasks	executor.running_tasks	计数	函数，执行程序
TasksExecutable	scheduler.tasks.executable	计数	函数，计划程序
TasksPending Note 不适用于 Apache Airflow v2.2 及更高版本。	scheduler.tasks.pending	计数	函数，计划程序
TasksRunning	scheduler.tasks.running	计数	函数，计划程序

CloudWatch 指标	Apache Airflow 指标	单位	维度	
TasksStarving	scheduler.tasks.starving	计数	函数，计划程序	
TasksWithoutDagRun	scheduler.tasks.without_dagrun	计数	函数，计划程序	
DAGFileProcessingLastNumOfDbQueries Note 在 Apache Airflow v2.10.1 及更高版本中可用。	dag_processing.last_num_of_db_queries.{dag_filename}	计数	DAG 文件名，{dag_filename}	

CloudWatch 指标	Apache Airflow 指标	单位	维度	
PoolScheduledSlots Note 在 Apache Airflow v2.10.1 及更高版本中可用。	pool.scheduled_slots.{pool_name}	计数	池，{pool_name}	
TaskCpuUsage Note 在 Apache Airflow v2.10.1 及更高版本中可用。	cpu.usage.{dag_id}.{task_id}	百分比	DAG, {dag_id} 任务，{task_id}	

CloudWatch 指标	Apache Airflow 指标	单位	维度	
TaskMemoryUsage Note 在 Apache Airflow v2.10.1 及更高版本中可用。	mem.usage.{dag_id}.{task_id}	百分比	DAG, {dag_id} 任务，{task_id}	

Apache Airflow 计时器

本节中的 Apache Airflow 指标包含有关 Apache Airflow 计时器的数据。

CloudWatch 指标	Apache Airflow 指标	单位	维度	
CollectDBDags	collect_db_dags	毫秒	函数，DAG 处理	
CriticalSectionDuration	scheduler.critical_section_duration	毫秒	函数，计划程序	
CriticalSectionQueryDuration	scheduler.critical_section_query_duration	毫秒	函数，计划程序	

CloudWatch 指标	Apache Airflow 指标	单位	维度	
Note 适用于 Apache Airflow v2.5.1 及更高版本。				
DAGDependencyCheck	dagrun.dependency-check.{dag_id}	毫秒	DAG, {dag_id}	
DAGDurationFailed	dagrun.duration.failed.{dag_id}	毫秒	DAG, {dag_id}	
DAGDurationSuccess	dagrun.duration.success.{dag_id}	毫秒	DAG, {dag_id}	
DAGFileProcessingLastDuration	dag_processing.last_duration.{dag_filename}	秒	DAG 文件名，{dag_filename}	
DAGScheduleDelay	dagrun.schedule_delay.{dag_id}	毫秒	DAG, {dag_id}	

CloudWatch 指标	Apache Airflow 指标	单位	维度	
FirstTask SchedulingDelay	dagrun.{dag_id}.first_task_scheduling_delay	毫秒	DAG, {dag_id}	
Scheduler LoopDuration Note 适用于 Apache Airflow v2.5.1 及更高版本。	scheduler.scheduler_loop_duration	毫秒	函数，计划程序	
TaskInstanceDuration	dag.{dag_id}.{task_id}.duration	毫秒	DAG, {dag_id} 任务，{task_id}	

CloudWatch 指标	Apache Airflow 指标	单位	维度	
TaskInstanceQueuedDuration	dag.{dag_id}.{t .queued_duration Note 适用于 Apache Airflow v2.7.2 及更高版本。	毫秒	DAG, {dag_id} 任务，{task_id}	
TaskInstanceScheduledDuration Note 适用于 Apache Airflow v2.7.2 及更高版本。	dag.{dag_id}.{t .scheduled_duration	毫秒	DAG, {dag_id} 任务，{task_id}	

选择要报告的指标

您可以使用以下 Amazon MWAA 配置选项来选择 CloudWatch向哪些 Apache Airflow 发送或屏蔽的 Apache Airflow 指标：

- **metrics.metrics_allow_list**— 逗号分隔的前缀列表，可用于选择您的环境向哪些指标发送到 CloudWatch 哪些指标。如果您希望 Apache Airflow 不发送所有可用指标，而是选择元素的子集，请使用此选项。例如 `scheduler,executor,dagrun`。
- **metrics.metrics_block_list** — 以逗号分隔的前缀列表，用于筛选出以列表元素开头的指标。例如 `scheduler,executor,dagrun`。

如果同时配置 `metrics.metrics_allow_list` 和 `metrics.metrics_block_list`，Apache Airflow 将忽略 `metrics.metrics_block_list`。如果您配置 `metrics.metrics_block_list` 但未配置 `metrics.metrics_allow_list`，Apache Airflow 会过滤掉您在 `metrics.metrics_block_list` 中指定的元素。

> **Note**
>
> `metrics.metrics_allow_list` 和 `metrics.metrics_block_list` 配置选项仅适用于 Apache Airflow v2.6.3 及更高版本。对于先前版本的 Apache Airflow，请改用 `metrics.statsd_allow_list` 和 `metrics.statsd_block_list`。

接下来做什么？

- 浏览用于发布环境运行状况指标的 Amazon MWAA API 操作，网址为。PublishMetrics

Amazon MWAA 的容器、队列和数据库指标

除了 Apache Airflow 指标外，您还可以 CloudWatch使用监控适用于 Apache Airflow 环境的亚马逊托管工作流程的底层组件，它收集原始数据并将数据处理为可读的近乎实时的指标。借助这些环境指标，您可以更清楚地了解关键性能指标，从而帮助您适当调整环境规模并调试工作流程中的问题。这些指标适用于 Amazon MWAA 上支持的所有 Apache Airflow 版本。

Amazon MWAA 将为每个 Amazon Elastic Container Service（Amazon ECS）容器和 Amazon Aurora PostgreSQL 实例提供 CPU 和内存使用率，提供 Amazon Simple Queue Service（Amazon SQS）指标指示消息数量和最旧消息存放时间，提供 Amazon Relational Database Service（Amazon RDS）指标指示数据库连接、队列磁盘深度、写入操作、延迟和吞吐量，以及提供 Amazon RDS 代理指标。这些指标还包括基础工作线程、额外工作线程、计划程序和 Web 服务器的数量。

这些统计数据会保存 15 个月，从而使您能够访问历史信息，并能够更好地了解计划失败的原因，并对潜在问题进行故障排除。您还可以设置警报来监控特定的阈值，并在达到那些阈值时发送通知或执行操作。有关更多信息，请参阅 A mazon CloudWatch 用户指南。

主题

术语

命名空间

命名空间是 AWS 服务 CloudWatch 指标的容器。Amazon MWAA 的命名空间为 AWS/MWAA。

CloudWatch 指标

CloudWatch 指标表示特定于的一组按时间顺序排列的数据点。 CloudWatch

维度

维度是作为指标标识一部分的 name/value 配对。

单位

所有统计数据都有度量单位。Amazon MWAA 的单位包括数量计数。

Dimensions

本节介绍中 Amazon MWAA 指标的 CloudWatch 维度分组。 CloudWatch

维度	说明
Cluster	Amazon MWAA 环境用于运行 Apache Airflow 组件的最少三个 Amazon ECS 容器的指标：调度器、Worker 节点和 Web 服务器。
队列	Amazon SQS 队列的指标，用于将计划程序与工作线程分离。当工作线程阅读消息时，它们被

维度	说明
	视为机上信息，不适用于其他工作线程。如果消息在 12 小时可见性超时之前未被删除，则这些消息可供其他工作线程读取。
数据库	Amazon MWAA 使用的 Aurora 集群的指标。这包括主数据库实例和支持读取操作的只读副本的指标。Amazon MWAA 同时发布 READER 和 WRITER 实例的数据库指标。

在 CloudWatch 控制台中访问指标

本节介绍如何在中访问您的亚马逊 MWAA 指标。 CloudWatch

访问维度的性能指标

1. 在 CloudWatch 控制台上打开 “指标” 页面。
2. 选择你的 AWS 区域。
3. 选择 AWS/MWAA 命名空间。
4. 在所有指标选项卡中，选择一个维度。例如，集群。
5. 为维 CloudWatch 度选择一个指标。例如，NumSchedulers或 CPU 利用率。然后，选择绘制所有搜索结果的图表。
6. 选择图表化指标选项卡以访问性能指标。

指标的列表

下表列出了 Amazon MWAA 的集群、队列和数据库服务指标。要访问直接从 Amazon ECS、Amazon SQS 或 Amazon RDS 发布的指标的描述，请选择相应的文档链接。

主题

- 集群指标
- 数据库指标
- 队列指标
- 应用程序负载均衡器指标

集群指标

以下指标适用于每个计划程序、基础工作线程、其他工作线程和 Web 服务器。有关每个集群指标的更多信息和描述，请参阅《Amazon ECS 开发人员指南》中的可用指标和维度。

命名空间	指标	单位
AWS/MWAA	CPUUtilization	百分比
AWS/MWAA	MemoryUtilization	百分比

评估额外工作线程和 Web 服务器容器的数量

您可以按以下过程所述，使用集群维度中提供的组件指标来评估环境在给定时间点正在使用的额外工作线程或 Web 服务器数量。为此，您可以绘制 CPUU ssization 或MemoryUtilization指标的图表，并将统计数据类型设置为 “样本数”。结果值是 AdditionalWorker 组件的 RUNNING 任务总数。了解环境使用的额外工作线程实例数量，有助您衡量环境的扩缩情况，并有利于您优化额外工作线程的数量。

Workers

要评估额外工作人员的人数，请使用 AWS 管理控制台

1. 选择 AWS/MWAA 命名空间。
2. 在所有指标选项卡中，选择集群维度。
3. 在 “集群” 维度中 AdditionalWorker，选择 CPUU利用率或指标。MemoryUtilization
4. 在绘成图表的指标选项卡上，将周期设置为 1 分钟，将统计数据更改为样本数。

webservers

要评估其他 Web 服务器的数量，请使用 AWS 管理控制台

1. 选择 AWS/MWAA 命名空间。
2. 在所有指标选项卡中，选择集群维度。
3. 在 “集群” 维度中 AdditionalWebservers，选择 CPUU利用率或指标。MemoryUtilization
4. 在绘成图表的指标选项卡上，将周期设置为 1 分钟，将统计数据更改为样本数。

有关更多信息，请参阅《Amazon Elastic Container Service 开发人员指南》中的服务 RUNNING 任务数。

数据库指标

以下指标适用于与 Amazon MWAA 环境关联的每个数据库实例。

命名空间	指标	单位
AWS/MWAA	CPUUtilization	百分比
AWS/MWAA	DatabaseConnections	计数
AWS/MWAA	DiskQueueDepth	计数
AWS/MWAA	FreeableMemory	字节
AWS/MWAA	VolumeWriteIOPS	每 5 分钟计数
AWS/MWAA	WriteIOPS	每秒计数
AWS/MWAA	WriteLatency	秒
AWS/MWAA	WriteThroughput	每秒字节数

队列指标

有关以下队列指标的单位和描述的更多信息，请参阅《亚马逊简单队列服务开发者指南》中的 Amazon SQS 可用 CloudWatch 指标。

命名空间	指标	单位
		秒

命名空间	指标	单位
AWS/MWAA	ApproximateAgeOfOldestTask	
AWS/MWAA	RunningTasks	计数
AWS/MWAA	QueuedTasks	计数

应用程序负载均衡器指标

应用程序负载均衡器指标适用于在环境中运行的 Web 服务器。Amazon MWAA 根据流量大小，使用这些指标来扩展 Web 服务器。有关以下负载均衡器指标的单位和描述的更多信息，请参阅《应用程序负载均衡器用户指南》中的 Application Load Bal ancer CloudWatch 指标。

命名空间	指标	单位
AWS/MWAA	ActiveConnectionCount	计数

Amazon MWAA 的安全性

云安全 AWS 是重中之重。作为 AWS 客户，您可以受益于专为满足大多数安全敏感型组织的要求而构建的数据中心和网络架构。

安全是 AWS 与您（客户）的共同责任。责任共担模式将其描述为云的安全性和云中的安全性：

- 云安全 — AWS 负责保护在 AWS 云中运行 AWS 服务的基础架构。 AWS 还为您提供可以安全使用的服务。 Third-party 作为AWS 合规计划合规计划合规计划合的一部分，审计师定期测试和验证我们安全的有效性。要了解适用于 Amazon MWAA 的合规计划，请参阅按合规计划提供的范围内的AWS服务（按合规计划划分）。
- 云端安全-您的责任由您使用的 AWS 服务决定。您还需要对其它因素负责，包括数据的敏感性、贵公司的要求以及适用的法律法规。

该文档帮助您了解如何在使用 Amazon MWAA 时应用责任共担模式。根据它配置 Amazon MWAA 以实现安全性和合规性目标。您还将学习如何使用其他 AWS 服务来帮助您监控和保护您的 Amazon MWAA 资源。

本节内容：

- Amazon MWAA
- AWS Identity and Access Management
- Amazon MWAA 的合规性验证
- Amazon MWAA 的弹性
- Amazon MWAA 中的基础设施安全性
- Amazon MWAA 中的配置和脆弱性分析
- Amazon MWAA 的安全最佳实践

Amazon MWAA

AWS 分担责任模式分担责任模型适用于适用于 Apache Airflow 的亚马逊托管工作流程中的数据保护。如本模型所述 AWS ，负责保护运行所有内容的全球基础架构 AWS Cloud。您负责维护对托管在此基础设施上的内容的控制。此内容包括您所使用的 AWS 服务 的安全配置和管理任务。有关数据隐私的更多信息，请参阅数据隐私常见问题。有关欧洲数据保护的信息，请参阅 AWS 安全博客上的 AWS 责任共担模式和 GDPR 博客文章。

出于数据保护目的，我们建议您保护 AWS 账户 凭证并使用 AWS Identity and Access Management (IAM) 设置个人用户账户。这仅向每个用户授予履行其工作职责所需的权限。我们还建议您通过以下方式保护数据：

- 对每个账户使用多重身份验证（MFA）。
- 用于 SSL/TLS 与 AWS 资源通信。建议使用 TLS 1.2 或更高版本。
- 使用设置 API 和用户活动日志 AWS CloudTrail。
- 使用 AWS 加密解决方案以及 AWS 服务中的所有默认安全控制。
- 使用高级托管安全服务（例如 Amazon Macie），其有助于发现和保护存储在 Simple Storage Service（Amazon S3）中的个人数据。

我们强烈建议您切勿将机密信息或敏感信息（例如您客户的电子邮件地址）放入标签或自由格式字段（例如名称字段）。这包括您使用控制台、API 或软件开发工具包使用 Amazon MWAA 或其他 AWS 服务时。 AWS CLI AWS 您在用于名称的标签或自由格式字段中输入的任何数据都可能会用于计费或诊断日志。如果您向外部服务器提供 URL，强烈建议您不要在网址中包含凭证信息来验证对该服务器的请求。

Amazon MWAA 上的加密

以下主题描述 Amazon MWAA 如何保护静态和传输中的数据。使用此信息了解 Amazon MWAA 如何与之集成 AWS KMS 以加密静态数据，以及如何使用传输层安全 (TLS) 协议对传输中的数据进行加密。

主题

- 静态加密
- 传输中加密

静态加密

在 Amazon MWAA 上，静态数据是服务保存到永久媒体的数据。

您可以使用AWS自有密钥进行静态数据加密，也可以选择在创建环境时提供用于额外加密的Customer-managed 密钥。如果您选择使用客户管理的 KMS 密钥，则该密钥必须与您在环境中使用的其他 AWS 资源和服务位于同一个账户中。

要使用客户管理的 KMS 密钥，必须附上 CloudWatch 访问密钥策略所需的策略声明。当您在环境中使用客户托管的 KMS 密钥时，Amazon MWAA 会代表您附加四项授权。有关 Amazon MWAA 附加到客户管理的 KMS 密钥的授权的更多信息，请参阅数据加密Customer-managed 密钥。

如果您未指定客户管理的 KMS 密钥，则默认情况下，Amazon MWAA 会使用自己的 KMS 密钥来加密和解密您的数据。 AWS 我们建议使用 AWS 自有的 KMS 密钥来管理 Amazon MWAA 上的数据加密。

> **Note**
>
> 您需要为在 Amazon MWAA 上存储和使用 AWS 自有或客户管理的 KMS 密钥付费。有关更多信息，请参阅 AWS KMS 定价。

加密构件

在创建 Amazon MWAA 环境时，您可以通过指定AWS拥有的密钥或密Customer-managed 钥来指定用于静态加密的加密工件。Amazon MWAA 会向指定密钥添加所需的授权。

亚马逊 S3 — Amazon S3 数据使用加密 (SSE) 在对象级别进行 Server-Side 加密。Amazon S3 加密和解密过程在存储 DAG 代码和支持文件的 Amazon S3 存储桶上进行。将对象上传到 Amazon S3 时对其进行加密，并在将其下载到 Amazon MWAA 环境时对其进行解密。默认情况下，如果您使用的是由客户托管的 KMS 密钥，Amazon MWAA 会使用它来读取和解密 Amazon S3 存储桶中的数据。

CloudWatch 日志-如果您使用的是 AWS 拥有的 KMS 密钥，则发送 CloudWatch 到日志的 Apache Airflow 日志将使用 SSE 使用日志 CloudWatch 拥有的 KMS 密钥进行加密。 AWS 如果您使用的是客户管理的 KMS 密钥，则必须向 KMS 密钥添加密钥策略以允许 CloudWatch Logs 使用您的密钥。

Amazon SQS — Amazon MWAA 为环境创建一个 Amazon SQS 队列。Amazon MWAA 使用 AWS 自有的 KMS 密钥或您指定的客户管理的 KMS 密钥使用 SSE 处理传入和传出队列的数据。无论您使用的是 AWS 自有的 KMS 密钥还是客户管理的 KMS 密钥，都必须向您的执行角色添加 Amazon SQS 权限。

Aurora PostgreSQL — Amazon MWAA 为环境创建一个 PostgreSQL 集群。Aurora PostgreSQL 使用 SSE 使用自有或客户管理的 KMS 密钥 AWS 对内容进行加密。如果您使用的是由客户托管的 KMS 密钥，Amazon RDS 会向该密钥添加至少两个授权：一个用于集群，一个用于数据库实例。如果您选择在多个环境中使用由客户托管的 KMS 密钥，Amazon RDS 可能会创建额外的授权。有关更多信息，请参阅 Amazon RDS 中的数据保护。

传输中加密

传输中数据是指在网络中传输时可能被拦截的数据。

传输层安全 (TLS) 对在您的环境的 Apache Airflow 组件和其他与亚马逊 MWAA 集成的 AWS 服务（例如亚马逊 S3）之间传输的 Amazon MWAA 对象进行加密。有关 Amazon S3 加密的更多信息，请参阅使用加密保护数据。

使用由客户托管的密钥进行加密

您可以选择为环境中的数据加密提供密Customer-managed 钥。您必须在与您的 Amazon MWAA 环境实例和存储工作流程资源的 Amazon S3 存储桶相同的区域中创建客户托管的 KMS 密钥。如果您指定的由客户托管的 KMS 密钥所在的账户与您用于配置环境的账户不同，则必须使用其 ARN 指定该密钥以进行跨账户访问。有关创建密钥的更多信息，请参阅《AWS Key Management Service 开发者指南》中的创建密钥。

支持什么？

AWS KMS 功能	支持
AWS KMS 密钥 ID 或 ARN。	是
AWS KMS 密钥别名。	否
AWS KMS 多区域密钥。	否

使用授权进行加密。

本主题介绍 Amazon MWAA 代表您附加到由客户托管的 KMS 密钥的授权，以加密和解密数据。

工作原理

客户管理的 KMS 密钥支持两种基于资源的访问控制机制：密钥策略和授权。 AWS KMS

当权限主要是静态且在同步服务模式下使用时，使用密钥政策。当需要更动态和更精细的权限时，例如当某服务需要为自己或其他账户定义不同的访问权限时，就会使用授权。

Amazon MWAA 使用四项授权策略并将其附加到由客户托管的 KMS 密钥。这是因为环境需要精细权限才能加密来自 CloudWatch 日志、Amazon SQS 队列、Aurora PostgreSQL 数据库数据库、Secrets Manager 密钥、亚马逊 S3 存储桶和 DynamoDB 表的静态数据。

当您创建 Amazon MWAA 环境并指定由客户托管的 KMS 密钥时，Amazon MWAA 会将授权策略附加到由客户托管的 KMS 密钥。这些策略允许 airflow.*us-east-1*.amazonaws.com 中的 Amazon MWAA 使用由客户托管的 KMS 密钥代表您加密 Amazon MWAA 拥有的资源。

Amazon MWAA 代表您为指定的 KMS 密钥创建并附加额外授权。这包括在删除环境后取消授权、使用客户管理的 KMS 密钥进行 Client-Side 加密 (CSE) 以及 AWS Fargate 执行角色需要在 Secrets Manager 中访问受客户管理的密钥保护的密钥的政策。

授权策略

Amazon MWAA 代表您向由客户托管的 KMS 密钥添加以下基于资源的策略授权。这些策略允许被授予者和主体 (Amazon MWAA) 执行策略中定义的操作。

授权 1：用于创建数据面板资源

```
{
  "Name": "mwaa-grant-for-env-mgmt-role-environment name",
  "GranteePrincipal": "airflow.us-east-1.amazonaws.com",
  "RetiringPrincipal": "airflow.us-east-1.amazonaws.com",
  "Operations": [
    "kms:Encrypt",
    "kms:Decrypt",
    "kms:ReEncrypt*",
    "kms:GenerateDataKey*",
    "kms:CreateGrant",
    "kms:DescribeKey",
    "kms:RetireGrant"
  ]
}
```

授权 2：用于 **ControllerLambdaExecutionRole** 访问权限

```
{
  "Name": "mwaa-grant-for-lambda-exec-environment name",
  "GranteePrincipal": "airflow.us-east-1.amazonaws.com",
  "RetiringPrincipal": "airflow.us-east-1.amazonaws.com",
  "Operations": [
    "kms:Encrypt",
    "kms:Decrypt",
    "kms:ReEncrypt*",
    "kms:GenerateDataKey*",
    "kms:DescribeKey",
    "kms:RetireGrant"
```

```
  ]
}
```

授权 3：用于 **CfnManagementLambdaExecutionRole** 访问权限

```
{
  "Name": " mwaa-grant-for-cfn-mgmt-environment name",
  "GranteePrincipal": "airflow.us-east-1.amazonaws.com",
  "RetiringPrincipal": "airflow.us-east-1.amazonaws.com",
  "Operations": [
    "kms:Encrypt",
    "kms:Decrypt",
    "kms:ReEncrypt*",
    "kms:GenerateDataKey*",
    "kms:DescribeKey"
  ]
}
```

授权 4：用于 Fargate 执行角色访问后端机密

```
{
  "Name": "mwaa-fargate-access-for-environment name",
  "GranteePrincipal": "airflow.us-east-1.amazonaws.com",
  "RetiringPrincipal": "airflow.us-east-1.amazonaws.com",
  "Operations": [
    "kms:Encrypt",
    "kms:Decrypt",
    "kms:ReEncrypt*",
    "kms:GenerateDataKey*",
    "kms:DescribeKey",
    "kms:RetireGrant"
  ]
}
```

将密钥策略附加到由客户托管的密钥

如果您选择在 Amazon MWAA 中使用自己的由客户托管的 KMS 密钥，则必须将以下策略附加到密钥上，以允许 Amazon MWAA 使用它来加密数据。

如果您在 Amazon MWAA 环境中使用的客户管理的 KMS 密钥尚未配置为可使用 CloudWatch，则必须更新密钥策略以允许使用加密日志。 CloudWatch 有关更多信息，请参阅 CloudWatch 使用 AWS Key Management Service 服务中的加密日志数据。

以下示例代表了 Lo CloudWatch gs 的密钥策略。替换为该区域提供的样本值。

```
{
  "Effect": "Allow",
  "Principal": {
    "Service": "logs.us-east-1.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-east-1:*:*"
    }
  }
}
```

AWS Identity and Access Management

AWS Identity and Access Management (IAM) 是一项 AWS 服务，可帮助管理员安全地控制对 AWS 资源的访问。IAM 管理员控制谁可以通过身份验证（登录）和授权（具有权限）使用 Amazon MWAA 资源。IAM 是一项无需额外付费即可使用的 AWS 服务。

本主题基本概述了 Amazon MWAA 的使用方式 AWS Identity and Access Management (IAM)。要了解如何管理对 Amazon MWAA 的访问权限，请参阅 管理对 Amazon MWAA 环境的访问。

内容

受众

您的使用方式 AWS Identity and Access Management (IAM) 因您的角色而异：

- 服务用户：如果您无法访问功能，请从管理员处请求权限（请参阅Amazon MWAA 身份和访问权限故障排除）
- 服务管理员：确定用户访问权限并提交权限请求（请参阅Amazon MWAA 如何与 IAM 协同工作）
- IAM 管理员：编写用于管理访问权限的策略（请参阅Amazon MWAA 基于身份的策略示例）

使用身份进行身份验证

身份验证是您 AWS 使用身份凭证登录的方式。您必须以 IAM 用户身份进行身份验证 AWS 账户根用户，或者通过担任 IAM 角色进行身份验证。

您可以使用来自身份源的证书 AWS IAM Identity Center（例如（IAM Identity Center）、单点登录身份验证或 Google/Facebook 证书，以联合身份登录。有关登录的更多信息，请参阅《AWS 登录 用户指南》中的如何登录您的 AWS 账户。

对于编程访问，AWS 提供 SDK 和 CLI 来对请求进行加密签名。有关更多信息，请参阅《IAM 用户指南》中的适用于 API 请求的AWS 签名版本 4。

AWS 账户 根用户

创建时 AWS 账户，首先会有一个名为 AWS 账户 root 用户的登录身份，该身份可以完全访问所有资源 AWS 服务 和资源。我们强烈建议不要使用根用户进行日常任务。有关要求根用户凭证的任务，请参阅《IAM 用户指南》中的需要根用户凭证的任务。

IAM 用户和群组

IAM 用户是对某个人员或应用程序具有特定权限的一个身份。建议使用临时凭证，而非具有长期凭证的 IAM 用户。有关更多信息，请参阅 IAM 用户指南中的要求人类用户使用身份提供商的联合身份验证才能 AWS 使用临时证书进行访问。

IAM 组指定一组 IAM 用户，便于更轻松地对大量用户进行权限管理。有关更多信息，请参阅《IAM 用户指南》中的 IAM 用户使用案例。

IAM 角色

IAM 角色是具有特定权限的身份，可提供临时凭证。您可以通过从用户切换到 IAM 角色（控制台）或调用 AWS CLI 或 AWS API 操作来代入角色。有关更多信息，请参阅《IAM 用户指南》中的担任角色的方法。

IAM 角色对于联合用户访问、临时 IAM 用户权限、跨账户访问、跨服务访问以及在 Amazon EC2 上运行的应用程序非常有用。有关更多信息，请参阅《IAM 用户指南》中的 IAM 中的跨账户资源访问。

使用策略管理访问

您可以 AWS 通过创建策略并将其附加到 AWS 身份或资源来控制中的访问权限。策略定义了与身份或资源关联时的权限。 AWS 在委托人提出请求时评估这些政策。大多数策略都以 JSON 文档的 AWS 形式存储在中。有关 JSON 策略文档的更多信息，请参阅《IAM 用户指南》中的 JSON 策略概述。

管理员使用策略，通过定义哪个主体可以在什么条件下对哪些资源执行哪些操作来指定谁有权访问什么。

默认情况下，用户和角色没有权限。IAM 管理员创建 IAM 策略并将其添加到角色中，然后用户可以担任这些角色。IAM 策略定义权限，与执行操作所用的方法无关。

Identity-Based 政策

Identity-based 策略是您附加到身份（用户、组或角色）的 JSON 权限策略文档。这些策略控制身份可以执行什么操作、对哪些资源执行以及在什么条件下执行。要了解如何创建基于身份的策略，请参阅《IAM 用户指南》中的使用客户管理型策略定义自定义 IAM 权限。

Identity-based 策略可以是内联策略（直接嵌入到单个身份中）或托管策略（附加到多个身份的独立策略）。要了解如何在托管策略和内联策略之间进行选择，请参阅《IAM 用户指南》中的在托管策略与内联策略之间进行选择。

Resource-Based 政策

Resource-based 策略是您附加到资源的 JSON 策略文档。示例包括 IAM 角色信任策略和 Amazon S3 存储桶策略。在支持基于资源的策略的服务中，服务管理员可以使用它们来控制对特定资源的访问。您必须在基于资源的策略中指定主体。

Resource-based 策略是位于该服务中的内联策略。您不能在基于资源的策略中使用 IAM 中的 AWS 托管策略。

访问控制列表（ACL）

访问控制列表（ACL）控制哪些主体（账户成员、用户或角色）有权访问资源。ACL 与基于资源的策略类似，但它们不使用 JSON 策略文档格式。

Amazon S3 和 Amazon VPC 就是支持 ACL 的服务示例。 AWS WAF要了解有关 ACL 的更多信息，请参阅《Amazon Simple Storage Service 开发人员指南》中的访问控制列表（ACL）概览。

其他策略类型

AWS 支持其他策略类型，这些策略类型可以设置更常见的策略类型授予的最大权限：

- 权限边界 – 设置基于身份的策略可以授予 IAM 实体的最大权限。有关更多信息，请参阅《 IAM 用户指南》中的 IAM 实体的权限边界。
- 服务控制策略（SCP）– 指定 AWS Organizations中组织或组织单元的最大权限。有关更多信息，请参阅《AWS Organizations 用户指南》中的服务控制策略。
- 资源控制策略（RCP）– 设置对账户中资源的最大可用权限。有关更多信息，请参阅《AWS Organizations 用户指南》中的资源控制策略（RCP）。
- 会话策略 – 在为角色或联合用户创建临时会话时，作为参数传递的高级策略。有关更多信息，请参阅《IAM 用户指南》中的会话策略。

多个策略类型

当多个类型的策略应用于一个请求时，生成的权限更加复杂和难以理解。要了解在涉及多种策略类型时如何 AWS 确定是否允许请求，请参阅 IAM 用户指南中的策略评估逻辑。

允许用户访问他们自己的权限

该示例说明了您如何创建策略，以允许 IAM 用户查看附加到其用户身份的内联和托管式策略。此策略包括在控制台上或使用 AWS CLI 或 AWS API 以编程方式完成此操作的权限。

```
{
    "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": "*"
        }
    ]
}
```

Amazon MWAA 身份和访问权限故障排除

可以使用以下信息，以帮助您诊断和修复在使用 Amazon MWAA 和 IAM 时可能遇到的常见问题。

我无权在 Amazon MWAA 中执行操作

如果 AWS 管理控制台 告诉您您无权执行某项操作，则必须联系管理员寻求帮助。管理员是指提供用户名和密码的人员。

我无权执行 iam：PassRole

如果您收到错误，指明您无权执行 `iam:PassRole` 操作，则必须更新策略以允许您将角色传递给 Amazon MWAA。

有些 AWS 服务 允许您将现有角色传递给该服务，而不是创建新的服务角色或服务相关角色。为此，您必须具有将角色传递到服务的权限。

当名为 marymajor 的 IAM 用户尝试使用控制台在 Amazon MWAA 中执行操作时，会发生以下示例错误。但是，服务必须具有服务角色所授予的权限才可执行此操作。Mary 不具有将角色传递到服务的权限。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

在这种情况下，必须更新 Mary 的策略以允许她执行 iam:PassRole 操作。

如果您需要帮助，请联系您的 AWS 管理员。您的管理员是提供登录凭证的人。

我想允许我以外的人进入 AWS 账户 访问我的 Amazon MWAA 资源

您可以创建一个角色，以便其他账户中的用户或您组织外的人员可以使用该角色来访问您的资源。您可以指定谁值得信赖，可以代入角色。对于支持基于资源的策略或访问控制列表（ACL）的服务，您可以使用这些策略向人员授予对您的资源的访问权。

要了解更多信息，请参阅以下内容：

- 要了解 Amazon MWAA 是否支持这些功能，请参阅 Amazon MWAA 如何与 IAM 协同工作。
- 要了解如何提供对您拥有的资源的访问权限 AWS 账户 ，请参阅 IAM 用户指南中的向您拥有 AWS 账户 的另一个 IAM 用户提供访问权限。
- 要了解如何向第三方提供对您的资源的访问权限 AWS 账户，请参阅 IAM 用户指南中的向第三方提供访问权限。 AWS 账户
- 要了解如何通过身份联合验证提供访问权限，请参阅《IAM 用户指南》中的为经过外部身份验证的用户（身份联合验证）提供访问权限。
- 要了解使用角色和基于资源的策略进行跨账户访问之间的差别，请参阅《IAM 用户指南》中的 IAM 中的跨账户资源访问。

Amazon MWAA 如何与 IAM 协同工作

Amazon MWAA 使用基于 IAM 身份的策略来授予对 Amazon MWAA 操作和资源的权限。有关您可用于控制对 Amazon MWAA 资源访问权限的自定义 IAM 策略 的推荐示例，请参阅 the section called “访问 Amazon MWAA 环境”。

要全面了解 Amazon MWAA 和其他 AWS 服务如何与 IAM 配合使用，请参阅 IAM 用户指南中的与 IAM 配合使用的AWS 服务。

Amazon MWAA 基于身份的策略

通过使用 IAM 基于身份的策略，您可以指定允许或拒绝的操作和资源以及允许或拒绝操作的条件。Amazon MWAA 支持特定的操作、资源和条件键。

以下步骤展示了如何使用 IAM 控制台创建新的 JSON 策略。此策略提供对 Amazon MWAA 资源的只读访问权限。

使用 JSON 策略编辑器创建策略

1. 登录 AWS 管理控制台 并打开 IAM 控制台，网址为https://console.aws.amazon.com/iam/。
2. 在左侧的导航窗格中，选择策略。

   如果这是您首次选择策略，则会显示欢迎访问托管式策略页面。选择开始使用。
3. 在页面的顶部，选择创建策略。
4. 在策略编辑器部分，选择 JSON 选项。
5. 输入以下 JSON 策略文档：

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

6. 选择下一步。

> **Note**
>
> 您可以随时在可视化和 JSON 编辑器选项卡之间切换。不过，如果您进行更改或在可视化编辑器中选择下一步，IAM 可能会调整策略结构以针对可视化编辑器进行优化。有关更多信息，请参阅《IAM 用户指南》中的调整策略结构。

7. 在查看并创建页面上，为您要创建的策略输入策略名称和描述（可选）。查看此策略中定义的权限以查看策略授予的权限。
8. 选择创建策略可保存新策略。

要了解在 JSON 策略中使用的所有元素，请参阅《IAM 用户指南》 中的 IAM JSON 策略元素参考。

操作

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

JSON 策略的 `Action` 元素描述可用于在策略中允许或拒绝访问的操作。在策略中包含操作以授予执行关联操作的权限。

策略语句必须包含 `Action` 或 `NotAction` 元素。`Action` 元素列出了策略允许的操作。`NotAction` 元素列出了不允许的操作。

为 Amazon MWAA 定义的操作反映了您可以使用 Amazon MWAA 执行的任务。Amazon Detective 中的策略操作具有以下前缀：`airflow:`。

您可以使用通配符（*）来指定多个操作。您可以授予对所有以单词（例如 `environment`）结尾的操作的访问权限，而不必单独列出这些操作。

要获得 Amazon MWAA 操作的列表，请参阅《IAM 用户指南》中的 Amazon MWAA 定义的操作。

Amazon MWAA 基于身份的策略示例

要访问 Amazon MWAA 策略，请参阅 管理对 Amazon MWAA 环境的访问。

默认情况下，IAM 用户和角色没有创建或修改 Amazon MWAA 资源的权限。他们也无法使用 AWS 管理控制台 AWS CLI、或 AWS API 执行任务。

IAM 管理员必须创建 IAM 策略，以便为用户和角色授予权限以对所需的指定资源执行特定的 API 操作。然后，管理员必须将这些策略附加到需要这些权限的 IAM 用户或组。

> **Important**
>
> 我们建议使用 IAM 角色和临时凭证来提供对 Amazon MWAA 资源的访问权限。避免将权限策略直接附加到 IAM 用户。

要了解如何使用这些示例 JSON 策略文档创建 IAM 基于身份的策略，请参阅《IAM 用户指南》中的在 JSON 选项卡上创建策略。

主题

- 策略最佳实践
- 使用 Amazon MWAA 控制台
- 允许用户访问他们自己的权限

策略最佳实践

Identity-based 策略决定是否有人可以在您的账户中创建、访问或删除亚马逊 MWAA 资源。这些操作可能会使 AWS 账户产生成本。创建或编辑基于身份的策略时，请遵循以下指南和建议：

- 开始使用 AWS 托管策略并转向最低权限权限 — 要开始向用户和工作负载授予权限，请使用为许多常见用例授予权限的AWS 托管策略。它们在你的版本中可用 AWS 账户。我们建议您通过定义针对您的用例的 AWS 客户托管策略来进一步减少权限。有关更多信息，请参阅《IAM 用户指南》中的 AWS 托管策略或工作职能的AWS 托管策略。
- 应用最低权限：在使用 IAM 策略设置权限时，请仅授予执行任务所需的权限。为此，您可以定义在特定条件下可以对特定资源执行的操作，也称为最低权限许可。有关使用 IAM 应用权限的更多信息，请参阅《IAM 用户指南》中的 IAM 中的策略和权限。
- 使用 IAM 策略中的条件进一步限制访问权限：您可以向策略添加条件来限制对操作和资源的访问。例如，您可以编写策略条件来指定必须使用 SSL 发送所有请求。如果服务操作是通过特定的方式使用的，则也可以使用条件来授予对服务操作的访问权限 AWS 服务，例如 CloudFormation。有关更多信息，请参阅《IAM 用户指南》中的 IAM JSON 策略元素：条件。
- 使用 IAM Access Analyzer 验证您的 IAM 策略，以确保权限的安全性和功能性：IAM Access Analyzer 会验证新策略和现有策略，以确保策略符合 IAM 策略语言（JSON）和 IAM 最佳实践。IAM Access Analyzer 提供 100 多项策略检查和可操作的建议，以帮助您制定安全且功能性强的策略。有关更多信息，请参阅《IAM 用户指南》中的使用 IAM Access Analyzer 验证策略。
- 需要多重身份验证 (MFA)-如果 AWS 账户您的场景需要 IAM 用户或根用户，请启用 MFA 以提高安全性。若要在调用 API 操作时需要 MFA，请将 MFA 条件添加到您的策略中。有关更多信息，请参阅《IAM 用户指南》中的使用 MFA 保护 API 访问。

有关 IAM 中的最佳实操的更多信息，请参阅《IAM 用户指南》中的 IAM 中的安全最佳实践。

使用 Amazon MWAA 控制台

要使用 Amazon MWAA 控制台，用户或角色必须有权访问与 API 中的相应操作相匹配的相关操作。

要访问 Amazon MWAA 策略，请参阅 管理对 Amazon MWAA 环境的访问。

允许用户访问他们自己的权限

该示例说明了您如何创建策略，以允许 IAM 用户查看附加到其用户身份的内联和托管式策略。此策略包括在控制台上或使用 AWS CLI 或 AWS API 以编程方式完成此操作的权限。

```
{
    "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": "*"
        }
    ]
}
```

Amazon MWAA 的合规性验证

要了解是否属于特定合规计划的范围，请参阅AWS 服务 “按合规计划划分的范围” ”，然后选择您感兴趣的合规计划。 AWS 服务 有关一般信息，请参阅AWS 合规计划AWS。

您可以使用下载第三方审计报告 AWS Artifact。有关更多信息，请参阅中的 “下载报告” 中的 “ AWS Artifact。

您在使用 AWS 服务 时的合规责任取决于您的数据的敏感性、贵公司的合规目标以及适用的法律和法规。有关您在使用时的合规责任的更多信息 AWS 服务，请参阅AWS 安全文档。

Amazon MWAA 的弹性

AWS 全球基础设施是围绕 AWS 区域 可用区构建的。各区域提供多个在物理上独立且隔离的可用区，这些可用区通过延迟低、吞吐量高且冗余性高的网络连接在一起。利用可用区，您可以设计和操作在可用区之间无中断地自动实现故障转移的应用程序和数据库。与传统的单个或多个数据中心基础设施相比，可用区具有更高的可用性、容错能力和可扩展性。

有关 AWS 区域 和可用区的更多信息，请参阅AWS 全球基础设施。

Amazon MWAA 中的基础设施安全性

作为一项托管服务，适用于 Apache Airflow 的亚马逊托管工作流程受全球网络安全的保护 AWS 。有关 AWS 安全服务以及如何 AWS 保护基础设施的信息，请参阅AWS 云安全。要使用基础设施安全的最佳实践来设计您的 AWS 环境，请参阅 S AWS ecurity Pillar Well-Architected Fram ework 中的基础设施保护。

您可以使用 AWS 已发布的 API 调用通过网络访问亚马逊 MWAA。客户端必须支持以下内容：

- 传输层安全性协议（TLS）。我们要求使用 TLS 1.2，建议使用 TLS 1.3。
- 具有完全向前保密 (PFS) 的密码套件，例如 DHE（短暂的）或 ECDHE（椭圆曲线短暂的 Diffie-Hellman）。 Diffie-Hellman大多数现代系统（如 Java 7 及更高版本）都支持这些模式。

Amazon MWAA 中的配置和脆弱性分析

配置和 IT 控制由您（我们的客户）共同 AWS 负责。

Amazon MWAA 定期修补和升级环境中的 Apache Airflow。确保对 VPC 使用适当的访问策略。

有关更多详细信息，请参阅以下资源：

- Amazon MWAA 的合规性验证
- 责任共担模式
- Amazon Web Services：安全过程概述
- Amazon MWAA 中的基础设施安全性
- Amazon MWAA 的安全最佳实践

Amazon MWAA 的安全最佳实践

Amazon MWA3 提供了在您开发和实施自己的安全策略时需要考虑的大量安全特征。以下最佳实践是一般指导原则，并不代表完整安全解决方案。这些最佳实践可能不适合环境或不满足环境要求，请将其视为有用的考虑因素而不是惯例。

- 使用最低许可的权限策略。仅向用户执行任务所需的资源或操作授予权限。
- AWS CloudTrail 用于监控您账户中的用户活动。
- 确保 Amazon S3 存储桶策略和对象 ACL 向关联的 Amazon MWAA 环境中的用户授予权限，以将对象放入存储桶。这样可以确保有权向存储桶添加工作流程的用户也拥有在 Airflow 中运行工作流程的权限。
- 使用与 Amazon MWAA 环境关联的 Amazon S3 存储桶。Amazon S3 存储桶可以是任何名称。请勿在存储桶中存储其他对象，也不要将该存储桶与其他服务一起使用。

Apache Airflow 中的安全最佳实践

Apache Airflow 不是多租户的。虽然有一些访问控制措施可以将某些功能限制为特定用户，而 Amazon MWAA 实施了这些措施，但 DAG 创建者确实能够编写 DAG 来更改 Apache Airflow 用户权限并与底层元数据库进行交互。

在 Amazon MWAA 上使用 Apache Airflow 时，我们建议您执行以下步骤，以确保环境的元数据库和 DAG 的安全。

- 为具有 DAG 写入权限或能够将文件添加到 Amazon S3 `/dags` 文件夹的不同团队使用不同的环境，前提是可以写入环境的用户也可以访问 Amazon MWAA 执行角色或 Apache Airflow 连接访问的任何内容。
- 请勿直接提供 Amazon S3 DAG 文件夹访问权限。取而代之的是，使用 CI/CD 工具将 DAG 写入 Amazon S3，并通过验证步骤确保 DAG 代码符合团队的安全准则。

- 阻止用户访问环境的 Amazon S3 存储桶。取而代之的是，使用 DAG 工厂，该工厂基于 YAML、JSON 或其他定义文件生成 DAG，该文件存储在与存储 DAG 的 Amazon MWAA Amazon S3 存储桶不同的位置。
- 在 Secrets Manager 中管理密钥 虽然这不会阻止可以编写 DAG 的用户读取密钥，但会阻止他们修改环境使用的密钥。

检测 Apache Airflow 用户权限的更改

你可以使用 CloudWatch Logs Insights 来检测发生的更改 Apache Airflow 用户权限的 DAG 的情况。为此，只要您的某个 DAG 更改 Apache Airfl CloudWatch ow 用户权限，您就可以使用 EventBridge 计划规则、Lambda 函数和 Logs Insights 向 CloudWatch 指标发送通知。

先决条件

要完成本节中的步骤，您需要以下满足以下条件：

- Amazon MWAA 环境启用 `INFO` 日志级别的所有 Apache Airflow 日志类型。有关更多信息，请参阅the section called “访问 Airflow 日志”。

要配置有关 Apache Airflow 用户权限更改的通知，请执行以下操作

1. 创建一个 Lambda 函数，该函数针对五个 Amazon MWAA 环境 CloudWatch 日志组（`DAGProcessing`、、、`Scheduler`和）运行以下 Logs Insights 查询字符串。`Task` `WebServer` `Worker`

   ```
   fields @log, @timestamp, @message | filter @message like "add-role" | stats count()
    by @log
   ```

2. 使用您在@@ 上一步中创建的 Lambda 函数作为 EventBridge 规则的目标，创建按计划运行的规则。使用 cron 或 rate 表达式配置计划程序，使其定期运行。

Amazon MWAA 上的 Apache Airflow 版本

本主题介绍 Amazon MWAA 支持的 Apache Airflow 版本，以及升级到最新版本的最佳实践。

主题

关于 Amazon MWAA 版本

Amazon MWAA 构建的容器镜像将 Apache Airflow 版本与其他常见的二进制文件和 Python 库捆绑在一起。该镜像使用您指定的版本的 Apache Airflow 基础版安装。创建环境时，需要指定要使用的镜像版本。环境创建后会一直使用指定的镜像版本，直到您将其升级到更高版本。

最新版本

Amazon MWAA 支持多个 Apache Airflow 版本。如果您在创建环境时未指定镜像版本，则 Amazon MWAA 会使用支持的最新版本的 Apache Airflow 创建环境。

Apache Airflow 版本

Amazon MWAA 上支持以下 Apache Airflow 版本。

> **Note**
>
> - 自 2025 年 12 月 30 日起，Amazon MWAA 将终止对 Apache Airflow 版本 v2.4.3、v2.5.1 和 v2.6.3 的支持。有关更多信息，请参阅Apache Airflow 版本支持和常见问题。

- 从 Apache Airflow v2.2.2 开始，Amazon MWAA 支持直接在 Apache Airflow Web 服务器上安装 Python 要求、提供程序包和自定义插件。
- 从 Apache Airflow v2.7.2 开始，要求文件必须包含一条 `--constraint` 语句。如果您未提供约束条件，Amazon MWAA 将为您指定一个约束条件，以确保您的要求中列出的程序包与您正在使用的 Apache Airflow 版本兼容。

  有关在需求文件中设置约束条件的更多信息，请参阅安装 Python 依赖项。

Apache Airflow 版本	Apache Airflow 发布日期	Amazon MWAA 上市日期	Apache Airflow 约束条件	Python 版本
v3.2.1	2026年4月22日	2026年5月19日	v3.2.1 约束文件	Python 3.12
v2.11.0	2025年5月20日	2026年1月7日	v2.11.0 约束文件	Python 3.12
v3.0.6	2025 年 8 月 29 日	2025 年 10 月 1 日	v3.0.6 约束文件	Python 3.12
v2.10.3	2024 年 11 月 4 日	2024 年 12 月 18 日	v2.10.3 约束文件	Python 3.11
v2.10.1	2024 年 9 月 5 日	2024 年 9 月 26 日	v2.10.1 约束文件	Python 3.11
v2.9.2	2024 年 6 月 10 日	2024 年 7 月 9 日	v2.9.2 约束文件	Python 3.11
v2.8.1	2024 年 1 月 19 日	2024 年 2 月 23 日	v2.8.1 约束文件	Python 3.11
v2.7.2	2023 年 10 月 12 日	2023 年 11 月 6 日	v2.7.2 约束文件	Python 3.11

有关迁移自管理的 Apache Airflow 部署或迁移现有 Amazon MWAA 环境的更多信息，包括备份元数据数据库的说明，请参阅 Amazon MWAA 迁移指南。

Apache Airflow 组件

本节描述了 Amazon MWAA 上每个 Apache Airflow 版本可用的 Apache Airflow 计划程序和工作线程的数量，并提供了 Apache Airflow 的关键功能列表，指出了支持每项功能的版本。

调度器

Apache Airflow v2 及更高版本的计划程序：

计划程序（默认值）	计划程序（最小值）	计划程序（最大值）
2	2	5

工作线程

Apache Airflow v2 及更高版本的工作线程：

工作线程（默认值）	工作线程（最小值）	工作线程（最大值）
10	1	25

升级 Apache Airflow 版本

Amazon MWAA 支持次要版本升级。这意味着您可以将环境从版本 `x.1.z` 升级到 `x.2.z`，但不能升级到新的主要版本，例如，从 `1.y.z` 升级到 `2.y.z`。

有关更多信息以及关于更新工作流程资源和将环境升级到新版本的详细说明，请参阅 the section called “更改版本”。

降级 Apache Airflow 版本

Amazon MWAA 支持将次要版本降级到在降级时仍受支持的较早版本。这意味着您可以将环境从版本 `x.2.z` 降级到 `x.1.z`，但不能降级到以前的主要版本，例如，从 `2.y.z` 降级到 `1.y.z`。

有关更多信息以及关于更新工作流程资源和将环境升级到新版本的详细说明，请参阅 the section called “更改版本”。

Apache Airflow 已弃用版本

下表列出了 Amazon MWAA 中已弃用的 Apache Airflow 版本，以及每个版本的初始发布日期和支持终止日期。有关迁移到更新版本的更多信息，请参阅 Amazon MWAA 迁移指南。

Apache Airflow 版本	Apache Airflow 发布日期	Amazon MWAA 上市日期	Amazon MWAA 支持终止日期
v1.10.12	2020 年 8 月 25 日	2020 年 11 月 24 日	2024 年 2 月 21 日
v2.0.2	2021 年 4 月 19 日	2021 年 5 月 25 日	2024 年 4 月 29 日
v2.2.2	2021 年 11 月 15 日	2022 年 1 月 27 日	2024 年 6 月 27 日
v2.4.3	2022 年 11 月 14 日	2023 年 1 月 5 日	2025 年 12 月 30 日
v2.5.1	2023 年 1 月 20 日	2023 年 4 月 11 日	2025 年 12 月 30 日
v2.6.3	2023 年 7 月 10 日	2023 年 8 月 9 日	2025 年 12 月 30 日

Apache Airflow 版本支持和常见问题

根据 Apache Airflow 社区的发布流程和版本政策，Amazon MWAA 承诺在任何给定时间至少支持三个 Apache Airflow 次要版本。我们将在支持终止日期前至少 180 天宣布给定 Apache Airflow 次要版本的终止支持日期。

常见问题

问：Amazon MWAA 支持 Apache Airflow 版本多长时间？

答：Amazon MWAA 在 Apache Airflow 补丁版本上市后支持至少 12 个月。

问：当对 Amazon MWAA 上的 Apache Airflow 版本的支持结束时，我是否会收到通知？

答：能。如果您账户中的任何 Amazon MWAA 环境在支持快要结束时运行该版本，则 Amazon MWAA 会在支持结束日期之前发出通知。 Health Dashboard

问：支持结束之日会发生什么？

答：在支持日期结束后，您将无法再使用弃用版本创建新的 Amazon MWAA 环境。您能够继续访问运行关联的已弃用 Apache Airflow 版本的现有 Amazon MWAA 环境，但相关风险由您自行承担。要在 Amazon MWAA 上升级到更新版本的 Apache Airflow，请参阅《Amazon MWAA 迁移指南》。

Important

您有责任保持您的 Amazon MWAA 版本为最新版本。 AWS 敦促所有客户将其的 Amazon MWAA 环境升级到最新版本，以便从最新的安全、隐私和可用性保护措施中受益。如果您在弃用日期之后在不受支持的版本或软件（简称旧版本）上运行环境，则更有可能面临安全、隐私和运营风险，包括停机事件。在旧版本上运行您的 Amazon MWAA 环境，即表示您确认自己了解并在知情的情况下承担这些风险，并且您同意尽快完成到最新版本的升级。在旧版本上继续运行您的环境需遵守管理您使用 AWS 服务的协议。

旧版本不被视为普遍可用， AWS 也不再为旧版本提供支持。因此， AWS 如果 AWS 确定旧版本对服务、其关联公司或任何其他第三方构成安全或责任风险或损害风险，则可以随时限制访问或使用任何旧版本。 AWS如果您决定继续在旧版本上中运行工作负载，可能会导致您的内容不可用、损坏或无法恢复。在旧版本上运行的环境受服务水平协议（SLA）例外条款的约束。

在旧版本上运行的环境和相关软件可能包含漏洞、错误、缺陷和有害组件。因此，尽管协议或服务条款中有任何相反的信息，但仍按原样 AWS 提供旧版本。

有关分担责任模型 AWS的更多信息，请参阅AWS Well-Architected 框架中的分担责任。

Amazon MWAA 服务端点和限额

Amazon MWAA 拥有以下服务限额和端点。服务限额（也称为限制）是 使用的服务资源或操作的最大数量AWS 账户

目录

服务端点

要访问 Amazon MWAA 的端点列表，请参阅 Amazon MWAA 端点和限额。

服务配额

限额名称	描述	默认限额	可调整
环境	每个区域每个账户的 Amazon MWAA 环境的最大数量。	10	是
每个环境的工件数	每个 Amazon MWAA 环境的最大工作线程数。	25	是
每个环境 Web 服务器数	每个 Amazon MWAA 环境的最大 Web 服务器数。	5	是

增加限额

您可以通过提交限额增加请求来申请增加可调整的限额。

Amazon MWAA 常见问题解答

本页描述了您在使用 Amazon MWAA 时可能遇到的常见问题。

目录

- Amazon MWAA 需要多长时间才能识别新的 DAG 文件？
- 为什么 Apache Airflow 没有采集我的 DAG 文件？
- 我能否从环境中删除 plugins.zip 或 requirements.txt？
- 为什么我的插件不出现在 Apache Airflow v2.0.2 管理员插件菜单中？
- 我能用吗 AWS Database Migration Service (DMS) 操作员？
- 当我使用 Airflow REST API 访问时 AWS 凭证，我能否将限制限制提高到每秒 10 笔交易以上 (TPS)？
- Airflow 任务执行 API 服务器在 Amazon MWAA 中的哪个位置运行？

支持的版本

Amazon MWAA 对 Apache Airflow v2 支持什么？

要了解 Amazon MWAA 支持的内容，请参阅 Amazon MWAA 上的 Apache Airflow 版本。

我可以使用何种版本的 Python？

Amazon MWAA 上支持以下 Apache Airflow 版本。

> **Note**
>
> - 自 2025 年 12 月 30 日起，Amazon MWAA 将终止对 Apache Airflow 版本 v2.4.3、v2.5.1 和 v2.6.3 的支持。有关更多信息，请参阅Apache Airflow 版本支持和常见问题。
> - 从 Apache Airflow v2.2.2 开始，Amazon MWAA 支持直接在 Apache Airflow Web 服务器上安装 Python 要求、提供程序包和自定义插件。
> - 从 Apache Airflow v2.7.2 开始，要求文件必须包含一条 `--constraint` 语句。如果您未提供约束条件，Amazon MWAA 将为您指定一个约束条件，以确保您的要求中列出的程序包与您正在使用的 Apache Airflow 版本兼容。
>
>   有关在需求文件中设置约束条件的更多信息，请参阅安装 Python 依赖项。

Apache Airflow 版本	Apache Airflow 发布日期	Amazon MWAA 上市日期	Apache Airflow 约束条件	Python 版本
v3.2.1	2026年4月22日	2026年5月19日	v3.2.1 约束文件	Python 3.12
v2.11.0	2025年5月20日	2026年1月7日	v2.11.0 约束文件	Python 3.12
v3.0.6	2025 年 8 月 29 日	2025 年 10 月 1 日	v3.0.6 约束文件	Python 3.12
v2.10.3	2024 年 11 月 4 日	2024 年 12 月 18 日	v2.10.3 约束文件	Python 3.11
v2.10.1	2024 年 9 月 5 日	2024 年 9 月 26 日	v2.10.1 约束文件	Python 3.11
v2.9.2	2024 年 6 月 10 日	2024 年 7 月 9 日	v2.9.2 约束文件	Python 3.11
v2.8.1	2024 年 1 月 19 日	2024 年 2 月 23 日	v2.8.1 约束文件	Python 3.11
v2.7.2	2023 年 10 月 12 日	2023 年 11 月 6 日	v2.7.2 约束文件	Python 3.11

有关迁移自管理的 Apache Airflow 部署或迁移现有 Amazon MWAA 环境的更多信息，包括备份元数据数据库的说明，请参阅 Amazon MWAA 迁移指南。

使用案例

我可以在亚马逊 Uni SageMaker fied Studio 中使用亚马逊 MWAA 吗？

可以。通过亚马逊 SageMaker Unified Studio 工作流程，您可以在亚马逊 U SageMaker nified Studio 中设置和运行一系列任务。亚马逊 SageMaker Unified Studio 工作流程使用 Apache Airflow 对数据处理程序进行建模并编排您的 SageMaker 亚马逊 Unified Studio 代码工件。有关更多信息，请参阅工作流程部分。要了解有关亚马逊的更多信息 SageMaker，请参阅什么是亚马逊 SageMaker？

我什么时候可以使用 AWS Step Functions 对比亚马逊 MWAA？

1. 您可以使用 Step Functions 来处理个人客户订单，因为 Step Functions 可以扩展以满足对一个订单或一百万个订单的需求。
2. 如果您运行的是夜间工作流程来处理前一天的订单，则可以使用 Step Functions 或 Amazon MWAA。Amazon MWAA 为您提供了一个开源选项，可以将工作流程从您正在使用的 AWS 资源中抽象出来。

环境通知

每个环境有多少任务存储空间可用？

任务存储空间限制为 20 GB，由 Amazon ECS Fargate 1.4 指定。RAM 量由您指定的环境类决定。有关环境类的更多信息，请参阅 配置 Amazon MWAA 环境类。

Amazon MWAA 环境使用的默认操作系统是什么？

Amazon MWAA 环境是在运行 2.6 及更早版本的 Amazon Linux 2 的实例上创建的，以及在运行 2.7 及更高版本的 Amazon Linux 2023 的实例上创建的。

我能否为我的 Amazon MWAA 环境使用自定义镜像？

不支持自定义镜像。Amazon MWAA 使用基于 Amazon Linux AMI 构建的镜像。对于您添加到环境的 Amazon S3 存储桶中的 requirements.txt 文件中指定的要求，Amazon MWAA 通过运行 `pip3 -r install` 来安装额外要求。

Amazon MWAA 是否符合 HIPAA 标准？

Amazon MWAA 符合《健康保险流通与责任法案（HIPAA）》的要求。如果您有 HIPAA 商业伙伴附录 (BAA)，则可以使用 Amazon MWAA 处理 AWS在 2022 年 11 月 14 日或之后创建的环境中处理受保护的健康信息 (PHI) 的工作流程。

Amazon MWAA 是否支持竞价型实例？

Amazon MWAA 目前不支持 Apache Airflow 的按需 Amazon EC2 竞价型实例类型。但是，Amazon MWAA 环境可以在 Amazon EMR 和 Amazon EC2 上触发竞价型实例。

Amazon MWAA 是否支持自定义域？

要使用自定义域作为 Amazon MWAA 主机名，请执行以下操作之一：

- 对于具有公共 Web 服务器访问权限的 Amazon MWAA 部署，您可以使用带有 CloudFront Lambda @Edge 的亚马逊将流量引导到您的环境，并将自定义域名映射到。 CloudFront有关更多信息以及为公共环境设置自定义域的示例，请参阅 Amazon MWAA 示例存储库中的公有 Web 服务器的 Amazon MWAA 自定义域示例。 GitHub
- 有关具有私有 Web 服务器访问权限的 Amazon MWAA 部署，请参阅 the section called “设置自定义域”。

我能否通过 SSH 进入我的环境？

虽然 Amazon MWAA 环境不支持 SSH，但可以使用 DAG 通过使用 `BashOperator` 来运行 bash 命令。例如：

```
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'] }}"
    )
```

要在 Apache Airflow UI 中触发 DAG，请使用：

```
{ "command" : "your bash command"}
```

为什么 VPC 安全组需要自引用规则？

通过创建自引用规则，您可以将源限制为 VPC 中的同一安全组，而不将其对所有网络开放。要了解更多信息，请参阅 the section called “VPC 安全”。

我能否在 IAM 中向不同的群组隐藏环境？

您可以通过在中指定环境名称来限制访问权限 AWS Identity and Access Management，但是， AWS 控制台中不提供访问筛选功能，如果用户可以访问一个环境，他们就可以访问所有环境。

我能否在 Apache Airflow 工作线程上存储临时数据？

Apache Airflow 运算符可以在工作线程上存储临时数据。Apache Airflow 工作线程可以访问环境的 Fargate 容器上的 `/tmp` 中的临时文件。

> **Note**
>
> 根据 Amazon ECS Fargate 1.4，任务总存储空间限制在 20 GB 以内。无法保证后续任务将在同一 Fargate 容器实例上运行，该实例可以使用不同的 `/tmp` 文件夹。

我能否指定超过 25 个 Apache Airflow 工作线程？

可以。尽管您可以在 Amazon MWAA 控制台上指定最多 25 个 Apache Airflow 工作线程，但您可以通过请求增加配额在环境中配置多达 50 个 Apache Airflow 工作线程。有关更多信息，请参阅请求增加限额。

Amazon MWAA 是否支持共享 Amazon VPC 或共享子网？

Amazon MWAA 不支持共享 Amazon VPC 或共享子网。您在创建环境时选择的 Amazon VPC 必须由尝试创建环境的账户拥有。但是，您可以将流量从 Amazon MWAA 账户中的 Amazon VPC 路由到共享 VPC。有关更多信息以及将流量路由到共享 Amazon VPC 的示例，请参阅《Amazon VPC 中转网关指南》中的集中出站路由到互联网。

我能否创建或集成自定义 Amazon SQS 队列来管理 Apache Airflow 中的任务执行和工作流程编排？

不可以，您不能在 Amazon MWAA 中创建、修改或使用自定义 Amazon SQS 队列。这是因为 Amazon MWAA 会自动为每个 Amazon MWAA 环境预置和管理自己的 Amazon SQS 队列。

指标

使用哪些指标来确定是否扩展工作线程？

亚马逊 MWAA 会监视QueuedTasks并RunningTasks进入， CloudWatch 以确定是否在您的环境中扩展 Apache Airflow 工作程序。要了解更多信息，请参阅 监控和指标。

我可以在中创建自定义指标 CloudWatch吗？

不在 CloudWatch 主机上。但是，您可以创建在中写入自定义指标的 DAG CloudWatch。有关更多信息，请参阅the section called “使用 DAG 编写自定义指标”。

DAG、运算符、连接和其他问题

我能否使用 `PythonVirtualenvOperator`？

Amazon MWAA 未明确支持 `PythonVirtualenvOperator`，但您可以创建一个使用 `PythonVirtualenvOperator` 的自定义插件。有关示例代码，请参阅 the section called “修补 PythonVirtualenvOperator 的自定义插件 ”。

Amazon MWAA 需要多长时间才能识别新的 DAG 文件？

DAG 会定期从 Amazon S3 存储桶同步到环境。如果您添加新的 DAG 文件，Amazon MWAA 大约需要 300 秒才能开始使用新文件。如果您更新现有的 DAG，Amazon MWAA 大约需要 30 秒才能识别更新。

这些值（新 DAG 为 300 秒，现有 DAG 更新为 30 秒）分别对应于 Apache Airflow 配置选项 `dag_dir_list_interval` 和 `min_file_process_interval`。

为什么 Apache Airflow 没有采集我的 DAG 文件？

以下是此问题的可能解决方案：

1. 检查执行角色是否具有对 Amazon S3 存储桶的足够权限。要了解更多信息，请参阅 Amazon MWAA 执行角色。
2. 检查 Amazon S3 存储桶是否已配置阻止公共访问并启用版本控制。要了解更多信息，请参阅 为 Amazon MWAA 创建 Amazon S3 存储桶。 。
3. 验证 DAG 文件本身。例如，请确保每个 DAG 都有一个唯一的 DAG ID。

我能否从环境中删除 `plugins.zip` 或 `requirements.txt`？

目前，没有办法在添加 plugins.zip 或 requirements.txt 后将其从环境中删除，但我们正在努力解决这个问题。在此期间，变通方法是分别指向空文本或 zip 文件。要了解更多信息，请参阅 删除 Amazon S3 上的文件。

为什么我的插件不出现在 Apache Airflow v2.0.2 管理员插件菜单中？

出于安全原因，Amazon MWAA 上的 Apache Airflow Web 服务器的网络出口流量有限，并且不会直接在 2.0.2 版本环境的 Apache Airflow Web 服务器上安装插件或 Python 依赖项。列出的插件允许亚马逊 MWAA 在 (IAM) 中对你的 Apache Airflow 用户进行身份验证。 AWS Identity and Access Management

为了能够直接在 Web 服务器上安装插件和 Python 依赖项，我们建议使用 Apache Airflow v2.2 及更高版本创建一个新环境。Amazon MWAA 直接在 Apache Airflow v2.2 及更高版本的 Web 服务器上安装 Python 依赖项和自定义插件。

我能用吗 AWS Database Migration Service (DMS) 操作员？

Amazon MWAA 支持 DMS 运算符。但是，此运算符不能用于对与 Amazon MWAA 环境关联的 Amazon Aurora PostgreSQL 元数据数据库执行操作。

当我使用 Airflow REST API 访问时 AWS 凭证，我能否将限制限制提高到每秒 10 笔交易以上 (TPS)？

是的，可以。要提高节流限制，请联系 AWS 客户支持。

Airflow 任务执行 API 服务器在 Amazon MWAA 中的哪个位置运行？

Amazon MWAA 在 Web 服务器组件中运行 Airflow 任务执行 API 服务器。任务执行 API 仅在 Apache Airflow v3 及更高版本中可用。了解有关 Amazon MWAA 架构的更多信息，请参阅 架构。

Amazon MWAA 故障排除

本章介绍在 Amazon MWAA 上使用 Apache Airflow 时可能遇到的常见问题和错误，以及解决这些错误的推荐步骤。

目录

故障排除： DAGs、操作员、连接和其他问题

本页的主题介绍了 Apache Airflow v2 和 v3 Python 依赖项、自定义 DAGs插件、操作员、连接、任务以及您在适用于 Apache Airflow 的亚马逊托管工作流程环境中可能遇到的网络服务器问题的解决方案。

目录

Connections

以下主题描述了您在使用 Apache Airflow 连接或其他数据库时可能收到的错误。 AWS

我无法连接 Secrets Manager

我们建议您完成以下步骤：

1. 了解如何为 Apache Airflow 连接和变量创建密钥，请参阅 the section called “配置 Secrets Manager”。
2. 要了解如何使用 Apache Airflow 变量 (`test-variable`) 的密钥，请参阅 为 Apache Airflow 变量使用 AWS Secrets Manager 中的密钥。
3. 要了解如何使用密钥进行 Apache Airflow 连接 (`myconn`)，请参阅 使用 AWS Secrets Manager 中的密钥进行 Apache Airflow 连接。

如何在我的执行角色策略中配置 secretsmanager:ResourceTag/<tag-key> Secrets Manager 条件或资源限制？

> **Note**
>
> 适用于 Apache Airflow 版本 2.0 及更早版本。

目前，由于 Apache Airflow 中存在已知问题，您无法通过在环境的执行角色中使用条件密钥或其他资源限制来限制对 Secrets Manager 密钥的访问。

我无法连接 Snowflake

我们建议您完成以下步骤：

1. 使用aws-mwaa-docker-images在本地测试你的 DAGs自定义插件和 Python 依赖关系 GitHub。
2. 将以下条目添加到适合环境的 requirements.txt 中。

```
apache-airflow-providers-snowflake==1.3.0
```

3. 将以下导入添加至 DAG：

```
from airflow.providers.snowflake.operators.snowflake import SnowflakeOperator
```

确保 Apache Airflow 连接对象包含以下键值对：

1. 连接 ID：snowflake_conn
2. 连接类型：Snowflake
3. 主机：<my account>.<my region if not us-west-2>.snowflakecomputing.com
4. Schema：<my schema>
5. 登录：<my user name>
6. 密码：********
7. 端口：<port, if any>
8. 附加依赖项：

```
{
```

```
      "account": "<my account>",
      "warehouse": "<my warehouse>",
      "database": "<my database>",
      "region": "<my region if not using us-west-2 otherwise omit this line>"
      }
```

例如：

```
>>> import json
    >>> from airflow.models.connection import Connection
    >>> myconn = Connection(
    ...    conn_id='snowflake_conn',
    ...    conn_type='Snowflake',
    ...    host='123456789012.us-east-1.snowflakecomputing.com',
    ...    schema='YOUR_SCHEMA'
    ...    login='YOUR_USERNAME',
    ...    password='YOUR_PASSWORD',
    ...    port='YOUR_PORT'
    ...    extra=json.dumps(dict(account='123456789012', warehouse='YOUR_WAREHOUSE',
 database='YOUR_DB_OPTION', region='us-east-1')),
    ... )
```

我无法在 Airflow UI 中找到我的连接

Apache Airflow 在 Apache Airflow UI 中提供了连接模板。无论连接类型如何，它都使用此模板来生成连接 URI 字符串。如果 Apache Airflow UI 中没有连接模板，则可以使用备用连接模板来生成连接 URI 字符串，例如使用 HTTP 连接模板。

我们建议您完成以下步骤：

1. 在 Apache Airflow UI 中访问 Amazon MWAA 提供的连接类型，请参阅 安装在 Amazon MWAA 环境中的 Apache Airflow 提供程序包。
2. 在 CLI 中访问创建 Apache Airflow 连接的命令，请参阅 Apache Airflow CLI 命令参考。
3. 要了解如何交替使用 Apache Airflow UI 中的连接模板来处理 Amazon MWAA 上的 Apache Airflow UI 中没有的连接类型，请参阅 连接类型概述。

Webserver

以下主题描述了您在 Amazon MWAA 上的 Apache Airflow Web 服务器上可能收到的错误。

我在访问 Web 服务器时收到 5xx 错误

我们建议您完成以下步骤：

1. 检查 Apache Airflow 配置选项。验证您指定为 Apache Airflow 配置选项的键值对（例如 AWS Secrets Manager）是否已正确配置。要了解更多信息，请参阅 the section called “我无法连接 Secrets Manager”。
2. 查看 `requirements.txt`。验证在 `requirements.txt` 中列出的 Airflow “Extras”程序包和其他库是否与 Apache Airflow 版本兼容。
3. 探索在 `requirements.txt` 文件中指定 Python 依赖项的方法，请参阅 在 requirements.txt 中管理 Python 依赖项。

我收到 **`The scheduler does not seem to be running`** 错误

如果调度程序似乎没有运行，或者最后一次 “心跳” 是在几个小时前收到的， DAGs 那么你可能不会在 Apache Airflow 中列出，也不会安排新任务。

我们建议您完成以下步骤：

1. 确认 VPC 安全组允许入站访问端口 5432。需要使用此端口才能连接到环境的 Amazon Aurora PostgreSQL 元数据数据库。添加此规则后，给 Amazon MWAA 几分钟，错误就会消失。要了解更多信息，请参阅 the section called “VPC 安全”。

   > **Note**
   >
   > - Aurora PostgreSQL 元数据库是亚马逊 MWAA 服务架构的一部分，在您的中不可用。AWS 账户
   > - 与数据库相关的错误通常是计划程序失败的症状，而不是根本原因。

2. 如果计划程序未运行，则可能是由于多种因素造成的，例如依赖项安装失败或计划程序过载。通过访问日志中的 CloudWatch 相应日志组，确认您的 DAGs插件和要求是否正常运行。要了解更多信息，请参阅 监控和指标。

任务

以下主题描述了在环境中执行 Apache Airflow 任务时可能收到的错误。

我的任务卡顿或者没有完成

如果 Apache Airflow 任务 “卡顿” 或未完成，我们建议您执行以下步骤：

1. 可能有大量 DAGs 已定义的。减少环境的数量 DAGs 并执行环境更新（例如更改日志级别）以强制重置。

   a. Airflow 会解析它们 DAGs 是否已启用。如果您使用的容量超过环境容量的 50%，则可能会开始让 Apache Airflow 计划程序不堪重负。这会导致 CloudWatch 指标中的总解析时间过长，或者 CloudWatch 日志中的 DAG 处理时间过长。还有其他优化 Apache Airflow 配置的方法，这些方法不在本指南的讨论范围之内。

   b. 要详细了解调整环境性能我们建议的最佳实践，请参阅 the section called “Apache Airflow 的性能调整”。

2. 队列中可能有大量任务。这通常表现为该None州大量且不断增长的任务，或者显示为大量任务。`Queued Tasks` and/or `Tasks Pending` CloudWatch出现此错误的原因如下：

   a. 如果要运行的任务多于环境的运行能力， and/or 则在自动缩放之前排队的大量任务有时间检测任务并部署更多工作人员。

   b. 如果要运行的任务多于环境的运行能力，我们建议减少同时运行的任务数量， and/or 增加 Apache DAGs Airflow 工作线程的最低数量。

   c. 如果在自动缩放有时间检测和部署更多工作人员之前有大量任务排队，我们建议错开任务部署，以 and/or 增加最少 Apache Airflow 工作人员。

   d. 您可以使用 AWS Command Line Interface (AWS CLI) 中的 update-environment 命令来更改在您的环境中运行的工作器的最小或最大数量。

   ```
   aws mwaa update-environment --name MyEnvironmentName --min-workers 2 --max-
   workers 10
   ```

   e. 要详细了解调整环境性能我们建议的最佳实践，请参阅 the section called “Apache Airflow 的性能调整”。

3. 如果任务停留在“正在运行”状态，也可以清除任务或将其标记为成功或失败。这允许环境的自动扩缩组件缩减运行在环境上的工作线程的数量。下图显示了滞留任务的示例。

   - 选择滞留任务的圆圈，然后选择清除（如图所示）。这允许 Amazon MWAA 缩减员工规模；否则，如果仍有排队的任务，Amazon MWAA 无法确定 DAGs 哪些已启用或禁用，也无法缩小规模。

4. 要详细了解 Apache Airflow 任务生命周期，请参阅《Apache Airflow 参考指南》的概念。

我在 Airflow v3 中任务失败，但没有日志

如果您的 Apache Airflow 3 任务失败但没有日志，请按照以下步骤操作：

- 如果工作线程日志显示错误（例如任务失败前后的 `Task handler raised error: WorkerLostError('Worker exited prematurely: exitcode 15 Job: 12.'))`，则表示分配给该任务的分支工作进程可能意外终止。

  要解决这个问题，可以考虑将 celery.worker_autoscale 配置为相同的最小值和最大值。例如：

  ```
  celery.worker_autoscale=5,5  # for mw1.small
  celery.worker_autoscale=10,10 # for mw1.medium
  celery.worker_autoscale=20,20 # for mw1.large
  ```

  这样可以确保工作线程池大小保持固定，防止工作线程意外终止。

CLI

以下主题介绍了您在 AWS Command Line Interface中运行 Airflow CLI 命令时可能收到的错误。

在 CLI 中触发 DAG 时我收到“503”错误

Airflow CLI 在 Apache Airflow Web 服务器上运行，该服务器的并发性有限。通常，它最多可以同时运行 4 个 CLI 命令。

为什么 **`dags backfill`** Apache Airflow CLI 命令会失败？ 是否有解决方法？

> **Note**
>
> 以下内容仅适用于 Apache Airflow v2.0.2 环境。

与`backfill`其他 Apache Airflow CLI 命令一样，该命令在处理 DAGs 任何命令之前都会在本地解析 DAGs 所有命令，无论该 CLI 操作适用于哪个 DAG。在使用 Apache Airflow v2.0.2 的 Amazon MWAA 环境中，由于在 CLI 命令运行时，Web 服务器上尚未安装插件和要求，因此解析操作将失败，并且不会调用 `backfill` 操作。如果环境中没有任何要求或插件，则 `backfill` 操作将成功。

为了能够运行 backfill CLI 命令，我们建议在 bash 运算符中调用它。在 bash 运算符中，backfill是从工作程序启动的，允许成功解析，因为所有必要的需求和插件都可用并已安装。DAGs 按照以下示例使用 BashOperator 创建 DAG 来运行 backfill。

```
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"
    )
```

运算符

以下主题描述了您在使用运算符时可能收到的错误。

我在使用 S3Transform 运算符时遇到了 **PermissionError: [Errno 13] Permission denied** 错误

如果您尝试使用 S3Transform 运算符运行 shell 脚本并收到 PermissionError: [Errno 13] Permission denied 错误，我们建议您执行以下步骤。以下步骤假设您已有一个 plugins.zip 文件。如果您要创建新的 plugins.zip，请参阅 安装自定义插件。

1. 使用aws-mwaa-docker-images在本地测试你的 DAGs自定义插件和 Python 依赖关系 GitHub。
2. 创建“转换”脚本。

```
#!/bin/bash
cp $1 $2
```

3. （可选）macOS 和 Linux 用户可能需要运行以下命令以确保脚本可执行。

```
chmod 777 transform_test.sh
```

4. 将脚本添加到 plugins.zip。

```
zip plugins.zip transform_test.sh
```

5. 按照将 plugins.zip 上传到 Amazon S3 中的步骤进行操作。
6. 按照在 Amazon MWAA 控制台上指定 plugins.zip 版本中的步骤操作。
7. 创建以下 DAG 文件。

```
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://amzn-s3-demo-bucket/files/input.txt',
      dest_s3_key='s3://amzn-s3-demo-bucket/files/output.txt'
      )
```

8. 按照将 DAG 代码上传到 Amazon S3 中的步骤进行操作。

故障排除：创建和更新 Amazon MWAA 环境

本页主题包含您在创建和更新 Amazon MWAA 环境时可能遇到的错误以及如何解决这些错误。

目录

更新 `requirements.txt`

以下主题描述了您在更新 `requirements.txt` 时可能收到的错误。

我指定了 `requirements.txt` 的新版本，更新环境花了 20 多分钟

如果环境安装 `requirements.txt` 文件的新版本花费的时间超过二十分钟，则环境更新失败，Amazon MWAA 正在回滚到容器镜像的最后一个稳定版本。

1. 检查程序包版本。我们建议您始终为 `requirements.txt` 中的 Python 依赖项指定特定版本 (==) 或最高版本 (<=) 。
2. 查看 Apache Airflow 日志。如果您启用了 Apache Airflow 日志，请在 CloudWatch 控制台的日志组页面上验证日志组是否已成功创建。如果您收到空白日志，最常见的原因是执行角色中缺少写入日志的 CloudWatch 或 Amazon S3 的权限。要了解更多信息，请参阅 执行角色。
3. 检查 Apache Airflow 配置选项。如果您使用的是 Secrets Manager，请验证您指定为 Apache Airflow 配置选项的键值对是否配置正确。要了解更多信息，请参阅 the section called “配置 Secrets Manager”。
4. 检查 VPC 网络配置。要了解更多信息，请参阅 the section called “环境状态”。
5. 检查执行角色权限。执行角色是 AWS Identity and Access Management (IAM) 角色，其权限策略允许 Amazon MWAA 代表您调用其他 AWS 服务（例如 Amazon S3、CloudWatch、Amazon SQS、Amazon ECR）的资源。还需要允许访问由客户托管的密钥或 AWS 自有密钥。要了解更多信息，请参阅 执行角色。
6. 要运行故障排除脚本来检查 Amazon MWAA 环境的 Amazon VPC 网络设置和配置，请参阅 GitHub AWS 支持工具中的验证环境脚本。

插件

以下主题描述了您在配置或更新 Apache Airflow 插件时可能遇到的问题。

Amazon MWAA 是否支持实现自定义 UI？

从 Apache Airflow v2.2.2 开始，Amazon MWAA 支持在 Apache Airflow Web 服务器上安装插件和实现自定义 UI。如果 Amazon MWAA 环境运行的是 Apache Airflow v2.0.2 或更早版本，则您将无法实现自定义 UI。

有关版本管理和升级现有环境的更多信息，请参阅 版本。

创建存储桶

以下主题描述了您在创建 Amazon S3 存储桶时可能收到的错误。

我无法选择 S3 阻止公共访问设置的选项

Amazon MWAA 环境的执行角色需要获得对 Amazon S3 存储桶执行 `GetBucketPublicAccessBlock` 操作的权限，以验证存储桶已阻止公共访问。我们建议您完成以下步骤：

1. 按照步骤将 JSON 策略附加到执行角色。
2. 附加以下 JSON 策略：

```
{
  "Effect":"Allow",
  "Action":[
    "s3:GetObject*",
    "s3:GetBucket*",
    "s3:List*"
  ],
  "Resource":[
    "arn:aws:s3:::amzn-s3-demo-bucket",
    "arn:aws:s3:::amzn-s3-demo-bucket/*"
  ]
}
```

   将 *amzn-s3-demo-bucket* 中的示例占位符替换为您的 Amazon S3 存储桶名称。
3. 要运行故障排除脚本来检查 Amazon MWAA 环境的 Amazon VPC 网络设置和配置，请参阅 GitHub AWS 支持工具中的验证环境脚本。

创建环境。

以下主题描述了您在创建环境时可能收到的错误。

我尝试创建环境，但它一直处于 **Creating** 状态

我们建议您完成以下步骤：

1. 使用公有路由检查 VPC 网络。如果您使用的是可访问互联网的 Amazon VPC，请验证以下内容：
    - Amazon VPC 已配置为允许 Amazon MWAA 环境使用的不同 AWS 资源之间的网络流量，如 the section called “关于联网” 中所定义。例如，VPC 安全组必须允许自引用规则中的所有流量，或者可以选择指定 HTTPS 端口范围 443 和 TCP 端口范围 5432 的端口范围。
2. 使用私有路由检查 VPC 网络。如果您使用的是不可访问互联网的 Amazon VPC，请验证以下内容：
    - Amazon VPC 已配置为允许 Amazon MWAA 环境使用的不同 AWS 资源之间的网络流量，如 the section called “关于联网” 中所定义。例如，两个私有子网不得有通往 NAT 网关（或 NAT 实例）的路由表，也不能有互联网网关。
3. 要运行故障排除脚本来检查 Amazon MWAA 环境的 Amazon VPC 网络设置和配置，请参阅 GitHub AWS 支持工具中的验证环境脚本。

我尝试创建环境，但它的状态显示为 **Create failed**

我们建议您完成以下步骤：

1. 检查 VPC 网络配置。要了解更多信息，请参阅 the section called “环境状态”。
2. 检查用户权限 在创建环境之前，Amazon MWAA 会根据用户的凭证进行试运行。AWS 账户 账户可能无权在 AWS Identity and Access Management (IAM) 中为环境创建某些资源。例如，如果您选择私有网络 Apache Airflow 访问模式，则 AWS 账户 必须已获得管理员的授权，访问适用于环境的 AmazonMWAAFullConsoleAccess 访问控制策略，该策略允许您的账户创建 VPC 端点。
3. 检查执行角色权限。执行角色是 AWS Identity and Access Management (IAM) 角色，其权限策略允许 Amazon MWAA 代表您调用其他 AWS 服务（例如 Amazon S3、CloudWatch、Amazon SQS、Amazon ECR）的资源。还需要允许访问由客户托管的密钥或 AWS 自有密钥。要了解更多信息，请参阅 执行角色。
4. 查看 Apache Airflow 日志。如果您启用了 Apache Airflow 日志，请在 CloudWatch 控制台的日志组页面上验证日志组是否已成功创建。如果您收到空白日志，最常见的原因是执行角色中缺少写入日志的 CloudWatch 或 Amazon S3 的权限。要了解更多信息，请参阅 执行角色。

5. 要运行故障排除脚本来检查 Amazon MWAA 环境的 Amazon VPC 网络设置和配置，请参阅 GitHub AWS 支持工具中的验证环境脚本。
6. 如果您使用的是无法访问互联网的 Amazon VPC，请确保您已创建一个 Amazon S3 网关端点，并授予 Amazon ECR 访问 Amazon S3 所需的最低权限。要了解有关创建 Amazon S3 网关端点的更多信息，请参阅以下内容：
   - 创建没有互联网访问权限的 Amazon VPC 网络
   - 在《Amazon Elastic Container Registry 用户指南》中的创建 Amazon S3 网关端点。

我尝试选择 VPC 但收到 **Network Failure** 错误

我们建议您完成以下步骤：

- 如果您在创建环境时尝试选择 Amazon VPC 时看到 `Network Failure` 错误，请关闭所有正在运行的浏览器内代理，然后重试。

我尝试创建环境但收到服务、分区或资源“必须传递”错误

我们建议您完成以下步骤：

- 您之所以收到此错误，可能是因为您为 Amazon S3 存储桶指定的 URI 的 URI 末尾包含一个“/”。我们建议删除路径中的“/”。该值必须为以下格式：

```
s3://amzn-s3-demo-bucket
```

我尝试创建环境，它的状态显示为 **Available**，但是当我尝试访问 Airflow UI 时，会显示 **Empty Reply from Server** 或 **502 Bad Gateway** 错误

我们建议您完成以下步骤：

1. 检查 VPC 安全组配置。要了解更多信息，请参阅 the section called “环境状态”。
2. 确认您在 `requirements.txt` 中列出的任何 Apache Airflow 程序包都对应于您在 Amazon MWAA 上运行的 Apache Airflow 版本。要了解更多信息，请参阅 安装 Python 依赖项。
3. 要运行故障排除脚本来检查 Amazon MWAA 环境的 Amazon VPC 网络设置和配置，请参阅 GitHub AWS 支持工具中的验证环境脚本。

我尝试创建一个环境，我的用户名是一堆随机的字符名称

- Apache Airflow 的用户名最大长度为 64 个字符。如果 AWS Identity and Access Management (IAM) 角色超过此长度，则使用哈希算法来缩短该长度，同时保持其唯一性。

Update environment

以下主题描述了您在更新环境时可能收到的错误。

我尝试更改环境类，但更新失败了

如果您将环境更新为其他环境类（例如将 `mw1.medium` 更改为 `mw1.small`），并且更新环境的请求失败，则环境状态将变为 `UPDATE_FAILED` 状态，环境将回滚到之前的稳定版本并根据环境的先前稳定版本进行计费。

我们建议您完成以下步骤：

1. 使用 GitHub 上的 aws-mwaa-docker-images 在本地测试 DAG、自定义插件和 Python 依赖项。
2. 要运行故障排除脚本来检查 Amazon MWAA 环境的 Amazon VPC 网络设置和配置，请参阅 GitHub AWS 支持工具中的验证环境脚本。

访问环境

以下主题描述了您在访问环境时可能收到的错误。

我无法访问 Apache Airflow UI

我们建议您完成以下步骤：

1. 检查用户权限 您可能未被授予可用来访问 Apache Airflow UI 的权限策略的访问权限。要了解更多信息，请参阅 the section called “访问 Amazon MWAA 环境”。
2. 检查网络访问。这可能是因为您选择了私有网络访问模式。如果 Apache Airflow UI 的 URL 采用以下格式 `387fbcn-8dh4-9hfj-0dnd-834jhdfb-vpce.c10.us-west-2.airflow.amazonaws.com`，则表示您在 Apache Airflow Web 服务器上使用私有路由。您可以将 Apache Airflow 访问模式更新为公有网络访问模式，也可以创建一种机制来访问 Apache Airflow Web 服务器的 VPC 端点。要了解更多信息，请参阅 the section called “管理 VPC 端点访问”。

故障排除：CloudWatch Logs 和 CloudTrail 错误

本页主题包含您在 Amazon MWAA 环境中可能遇到的 Amazon CloudWatch Logs 和 AWS CloudTrail 错误的解决方法。

目录

日志

以下主题描述了您在访问 Apache Airflow 日志时可能收到的错误。

我找不到我的任务日志，或者我收到了 **Reading remote log from Cloudwatch log_group** 错误

Amazon MWAA 已将 Apache Airflow 配置为直接从 Amazon CloudWatch Logs 读取和写入日志。如果工作线程无法启动任务或未能写入任何日志，您将看到错误：

```
*** 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.
```

- 我们建议您完成以下步骤：

a. 确认您已在环境 INFO 级别上启用任务日志。有关更多信息，请参阅访问 Amazon 中的 Airflow 日志 CloudWatch。

b. 验证环境执行角色的权限策略是否正确。

c. 验证运算符或任务是否正常运行，是否有足够的资源来解析 DAG，以及是否有相应的 Python 库可供加载。要验证依赖项是否正确，请尝试取消导入，直到找到导致问题的依赖项。我们建议使用 aws-mwaa-docker-images 测试 Python 依赖项。

任务在没有任何日志的情况下失败

如果工作流程中的任务失败并且您找不到失败任务的任何日志，请检查是否在默认参数中设置了 queue 参数，如下所列。

```
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'] }}"
    )
```

要解决此问题，请从代码中删除 queue，然后再次调用 DAG。

我在 CloudTrail 中遇到了 **ResourceAlreadyExistsException** 错误

```
"errorCode": "ResourceAlreadyExistsException",
    "errorMessage": "The specified log stream already exists",
    "requestParameters": {
        "logGroupName": "airflow-MyAirflowEnvironment-DAGProcessing",
        "logStreamName": "scheduler_cross-account-eks.py.log"
    }
```

某些 Python 要求，例如 apache-airflow-backport-providers-amazon 将 Amazon MWAA 用于与 CloudWatch 通信的 watchtower 库回滚到旧版本。我们建议您完成以下步骤：

- 将以下库添加到 requirements.txt。

```
watchtower==1.0.6
```

我在 CloudTrail 中看到了 **Invalid request** 错误

```
Invalid request provided: Provided role does not have sufficient permissions for s3
 location airflow-xxx-xxx/dags
```

如果您使用相同的 CloudFormation 模板创建 Amazon MWAA 环境和 Amazon S3 存储桶，则需要在 CloudFormation 模板中添加 DependsOn 部分。这两个资源（MWAA 环境和 MWAA 执行策略）在 CloudFormation 中有依赖关系。我们建议您完成以下步骤：

- 将以下 **DependsOn** 语句添加到 CloudFormation 模板中。

```
...
  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:
...
```

有关具体示例，请参阅 Amazon MWAA 的快速入门教程。

我在 Apache Airflow 日志中看到了 **Cannot locate a 64-bit Oracle Client library: "libclntsh.so: cannot open shared object file: No such file or directory**

- 我们建议您完成以下步骤：
  - 如果您使用的是 Apache Airflow v2，请添加 `core.lazy_load_plugins : False` 为 Apache Airflow 配置选项。要了解更多信息，请参阅 2 中的使用配置选项加载插件。

我在我的计划程序日志中看到了 psycopg2“服务器意外关闭了连接”

如果您看到类似于以下内容的错误，则说明 Apache Airflow 计划程序可能已耗尽资源。

```
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.
```

我们建议您完成以下步骤：

- 考虑升级到 Apache Airflow v2.0.2，此版本允许您指定多达 5 个计划程序。

我在我的 DAG 处理日志中看到了 **Executor reports task instance %s finished (%s) although the task says its %s**

如果您看到类似于以下内容的错误，则说明长时间运行的任务可能已达到 Amazon MWAA 上的任务时间限制。Amazon MWAA 对任何一个 Airflow 任务的限制为 12 小时，以防止任务卡在队列中并阻止自动扩缩等活动。

```
Executor reports task instance %s finished (%s) although the task says its %s. (Info:
 %s) Was the task killed externally
```

我们建议您完成以下步骤：

- 考虑将任务分解为多个运行时间较短的任务。Airflow 通常有运算符异步模型。它调用外部系统上的活动，Apache Airflow 传感器会进行轮询以检查其何时完成。如果传感器出现故障，则可以在不影响运算符功能的情况下安全地重试。

我在我的任务日志中看到了 **Could not read remote logs from log_group: airflow-*{*environmentName}-Task log_stream:* {*DAG_ID}/*{*TASK_ID}/*{*time}/*{*n}.log.**

如果您看到类似于以下内容的错误，则环境的执行角色可能不包含为任务日志创建日志流的权限策略。

```
Could not read remote logs from log_group: airflow-*{*environmentName}-Task
 log_stream:* {*DAG_ID}/*{*TASK_ID}/*{*time}/*{*n}.log.
```

我们建议您完成以下步骤：

- 使用 the section called “执行角色” 中的示例策略之一修改环境的执行角色。

您可能还在 `requirements.txt` 文件中指定了与 Apache Airflow 版本不兼容的提供程序包。例如，如果您使用的是 Apache Airflow v2.0.2，则可能已经指定了程序包，例如 apache-airflow-providers-databricks 程序包，它仅与 Airflow 2.1+ 兼容。

我们建议您完成以下步骤：

1. 如果您使用的是 Apache Airflow v2.0.2，请修改 `requirements.txt` 文件并添加 `apache-airflow[databricks]`。这将安装与 Apache Airflow v2.0.2 兼容的 Databricks 程序包的正确版本。
2. 使用 GitHub 上的 aws-mwaa-docker-images 在本地测试 DAG、自定义插件和 Python 依赖项。

Amazon MWAA 用户指南历史记录

下表介绍了自 2020 年 11 月以来对 Amazon MWAA 服务文档的重要新增。如需有关本文档更新的通知，您可以订阅 RSS 源。

变更	说明	日期
新的 Web 服务器访问模式	对于运行 Apache Airflow v3.2.1 及更高版本的环境，亚马逊 MWAA 现在支持第三种网络服务器访问模式，包括公网和私有网络访问。此模式既创建了用于浏览器访问 Apache Airflow UI 的公共网络负载均衡器，又创建了用于工作人员到 Web 服务器通信的私有 VPC 端点，允许工作人员在没有互联网访问的情况下通过 VPC 访问网络服务器。 • the section called “Apache Airflow 访问模式” • the section called “关于联网”	2026年5月26日
全新 Apache Airflow 版本	亚马逊 MWAA 现在支持 Apache Airflow v3.2.1。此更新包括有关更新的提供程序包的信息、有关在亚马逊 MWAA 上使用 Apache Airflow v3.2.1 的详细信息、关于不支持的 Apache Airflow 配置选项的新章节以及 Apache Airflow v3 的新数据库清理代码示例。 • 版本 • the section called “Version-specific 提供者套餐”	2026年5月19日

	• the section called “不支持的配置” • the section called “Aurora PostgreSQL 数据库清理”	
全新 Apache Airflow 版本	亚马逊 MWAA 现在支持 Apache Airflow v2.11.0。此更新包括有关更新的提供程序包的信息，以及有关在亚马逊 MWAA 上使用 Apache Airflow v2.11.0 的详细信息。 • 版本 • the section called “Version-specific 提供者套餐”	2026年1月7日
添加了 Apache Airflow v3 支持	更新了文档以包括对 Apache Airflow v3 的支持 • 代码示例 • 版本	2025 年 10 月 1 日
IPv6 更新	添加了有关 IPv6 支持的信息。 • the section called “关于联网”	2025 年 8 月 26 日
环境更新	添加了有关在环境处于 `MAINTENANCE` 状态的情况下执行更新时 `workerReplacementStrategy` 会从 `GRACEFUL` 变为 `FORCED` 的注释。 • the section called “更新环境”	2025 年 8 月 6 日

版本弃用信息	已更新有关版本弃用的主题，包括 Apache Airflow v2.4.3、Apache Airflow v2.5.1 和 Apache Airflow v2.6.3 的弃用通知和时间表。 • the section called “Apache Airflow 已弃用版本”	2025 年 6 月 24 日
添加了一个新的环境类：mw1.micro	Amazon MWAA 现在提供了一个新的环境类：mw1.micro。 • the section called “配置环境类” • the section called “Apache Airflow 的性能调整”	2024 年 11 月 19 日
支持使用更简单的方法来访问 Apache Airflow REST API	亚马逊 MWAA 现在提供了一种使用凭证与 Apache Airflow REST API 进行交互的简化方法。 AWS • the section called “使用 Apache Airflow REST API” • the section called “Apache Airflow Rest API 访问”	2024 年 10 月 23 日
全新 Apache Airflow 版本	Amazon MWAA 现在支持 Apache Airflow v2.10.1。此更新包括有关更新的提供程序包的信息，以及有关在 Amazon MWAA 上使用 Apache Airflow v2.10.1 的详细信息。 • 版本 • the section called “Version-specific 提供者套餐”	2024 年 9 月 26 日

全新 Apache Airflow 版本	Amazon MWAA 现在支持 Apache Airflow v2.9.2。此更新包括有关更新的提供程序包的信息，以及有关在 Amazon MWAA 上使用 Apache Airflow v2.9.2 的详细信息。 • 版本 • the section called “Version-specific 提供者套餐”	2024 年 7 月 9 日
Amazon MWAA 支持配置自定义 Web 服务器域名	Amazon MWAA 支持为无互联网访问权限的私有环境配置自定义 Web 服务器域名。此更新包括以下新主题，其中说明了设置新自定义域的信息。 • the section called “设置自定义域”	2024 年 6 月 18 日
Amazon MWAA 支持 Web 服务器自动扩缩和 Apache Airflow REST API	Amazon MWAA 现在支持 Web 服务器自动扩缩以及访问和使用 Apache Airflow REST API 的功能。 • the section called “配置 Web 服务器自动扩缩” • the section called “使用 Apache Airflow REST API”	2024 年 5 月 16 日
完善了有关自动扩缩行为的说明	更新了以下主题，以反映 Worker 节点在 Fargate Worker 节点缩减过程中接到新任务时新的 Amazon MWAA 自动扩缩行为。 • the section called “配置 Worker 节点自动扩缩”	2024 年 5 月 10 日

支持更大的实例大小	Amazon MWAA 现在支持两个更大的实例大小选项，以满足较大工作负载的需要：`mw1.xlarge` 和 `mw1.2xlarge` • the section called “环境功能”	2024 年 4 月 16 日
全新 Apache Airflow 版本	Amazon MWAA 现在支持 Apache Airflow v2.8.1。此更新包括有关更新的提供程序包的信息，以及有关在 Amazon MWAA 上使用 Apache Airflow v2.8.1 的详细信息。 • 版本 • the section called “Version-specific 提供者套餐”	2024 年 2 月 22 日
支持共享 Amazon VPC	Amazon MWAA 支持为使用亚马逊 OpenSearch 服务的组织创建跨账户环境，使用所有者账户中的中央共享亚马逊 VPC 来管理亚马逊 MWAA 资源。作为此次发布的一部分，Amazon MWAA 允许您选择创建和管理自己的 Amazon VPC 端点。 • the section called “管理您自己的 Amazon VPC 端点”	2023 年 11 月 15 日

全新 Apache Airflow 版本	Amazon MWAA 现在支持 Apache Airflow v2.7.2。此更新包括有关更新的提供程序包的信息，以及有关在 Amazon MWAA 上使用 Apache Airflow v2.7.2 的详细信息。 • 版本 • the section called “Version-specific 提供者套餐”	2023 年 11 月 6 日
全新 Apache Airflow 版本	Amazon MWAA 现在支持 Apache Airflow v2.6.3。此更新包括有关更新的提供程序包的信息，以及有关在 Amazon MWAA 上使用 Apache Airflow v2.6.3 的详细信息。 • 版本	2023 年 8 月 9 日
版本弃用信息	已更新有关版本弃用的主题，包括 Apache Airflow v2.0.2 和 Apache Airflow v2.2.2 的弃用通知和时间表。 • the section called “Apache Airflow 已弃用版本”	2023 年 7 月 31 日
新主题和用例	Amazon MWAA 支持次要版本升级。此更新包括以下新主题，该主题描述了如何升级环境并确保您的工作流程资源与要升级到的 Apache Airflow 版本兼容： • the section called “更改版本”	2023 年 6 月 5 日

已更新主题	已更新由客户托管的 IAM 策略，该策略授予用户 Amazon MWAA 的完整控制台和 API 访问权限。此更新描述了为什么您必须为提供 `iam:PassRole` 权限以允许用户将角色传递给 Amazon MWAA。Amazon MWAA 使用这些权限代表用户执行操作。 • the section called “访问 Amazon MWAA 环境”	2023 年 4 月 12 日
新指南	更新了有关配置 AWS Secrets Manager 为 Amazon MWAA 后端的主题，以提供有关使用查找模式的指导。使用查找模式可以收窄 Apache Airflow 搜索的密钥范围，并减少 Amazon MWAA 为检索连接和变量而向 Secrets Manager 调用 API 的次数。这样可以降低与使用 Secrets Manager 作为后端相关的成本。 • 创建 Secrets Manager 后端作为 Apache Airflow 配置选项	2023 年 4 月 12 日
全新 Apache Airflow 版本	Amazon MWAA 现在支持 Apache Airflow v2.5.1。此更新包括有关更新的提供程序包的信息，以及有关在 Amazon MWAA 上使用 Apache Airflow v2.5.1 的详细信息。 • 版本	2023 年 4 月 11 日

新主题和用例	已添加有关在 Amazon MWAA 环境中使用启动脚本的新主题。本主题介绍了为现有环境配置启动脚本、使用它安装 Linux 运行时以及设置环境变量。 • the section called “使用启动脚本”	2023 年 4 月 3 日
已更新有关私有 Web 服务器访问权限的章节	已更新以下有关私有 Web 服务器访问权限的主题。该更新澄清说，在具有私有 Web 服务器访问权限的环境中，您必须使用 Python wheel 档案 (`.whl`) 来打包和安装依赖项。 • 私有 Web 服务器访问模式	2023 年 2 月 24 日
已添加有关已弃用的 Apache Airflow 版本的信息	已更新版本主题，提供了有关 Amazon MWAA 如何管理正在弃用的 Apache Airflow 版本的新信息。已删除关于升级到更新版本的 Apache Airflow 的章节，以及介绍 Apache Airflow v1 和 Apache Airflow v2 之间变更的章节。有关迁移到更新版本的 Apache Airflow 的更多信息，请参阅《Amazon MWAA 迁移指南》。 • the section called “Apache Airflow 已弃用版本” • the section called “Apache Airflow 版本支持和常见问题”	2023 年 2 月 17 日

已修复 Amazon MWAA 容器指标中的问题	已更新容器指标主题，并已删除 `Cluster` 维度中不存在的一组错误指标。已添加一个章节，介绍如何通过绘制 `AdditionalWorker` 组件的 `CPUUtilization` 或 `MemoryUtilization` 指标并将统计数据类型设置为 `Sample Count`，来评估环境在给定时间使用的额外工作线程数。 • the section called “评估额外工作线程和 Web 服务器容器的数量”	2023 年 1 月 20 日
全新 Apache Airflow 版本	Amazon MWAA 现在支持 Apache Airflow v2.4.3。此更新包括有关更新的提供程序包的信息、有关在 Amazon MWAA 上使用 Apache Airflow v2.4.3 的详细信息，以及有关 Amazon MWAA 上每个 Apache Airflow 版本支持哪些功能的综合信息。 • 版本	2023 年 1 月 5 日

已更新服务相关角色的主题	更新了有关 Amazon MWAA 用来代表您创建和管理 AWS 资源的服务相关角色的信息，包括有关在不再需要服务相关角色时如何删除该角色的信息。这包括更新的服务相关角色权限策略，该策略允许 Amazon MWAA 在命名空间中发布其他 CloudWatch 指标。AWS/MWAA • the section called “Service-linked 角色”	2022 年 11 月 18 日
关于服务指标的新主题	已添加描述 Amazon MWAA 在 AWS/MWAA 命名空间中发出的服务指标的新主题。这些指标包括计划程序、工作线程和 Web 服务器的 Amazon ECS 集群指标、允许 Amazon MWAA 分离计划程序和工作线程的队列的 Amazon SQS 指标，以及元数据数据库的 Amazon RDS 指标。 • the section called “容器、队列和数据库指标”	2022 年 11 月 18 日
新主题	已添加有关修改约束文件的新指南，以指定与 Amazon MWAA 环境配合使用的新版本的提供程序包。 • the section called “约束条件文件”	2022 年 11 月 18 日

已更新常见问题解答条目	已更新与 Amazon MWAA 的 HIPAA 资格相关的信息。 • the section called “HIPAA 合规性”	2022 年 11 月 15 日
新主题	已添加有关在 Amazon MWAA 执行角色信任策略中使用 `aws:SourceArn` 和 `aws:SourceAccount` 全局条件上下文密钥的新主题，以防止跨服务混淆代理。 • the section called “Cross-service 混乱的副手预防”	2022 年 10 月 21 日
新示例代码	添加了更新的说明和向中写入自定义 OS-level 指标的 DAG 代码示例 CloudWatch。 • the section called “使用 DAG 编写自定义指标”	2022 年 9 月 13 日
新示例代码	添加了更新的说明和新的 AWS Lambda Python 代码示例，用于检索 Apache Airflow CLI 令牌，然后在指定的亚马逊 MWAA 环境中调用 DAG。 • the section called “使用 Lambda DAGs 进行调用”	2022 年 9 月 12 日
新架构图	已添加新架构图，演示了带有公有和私有 Web 服务器的 Amazon MWAA 环境。 • the section called “Apache Airflow 访问模式”	2022 年 9 月 12 日

新示例代码	已添加最新说明和新的 DAG 代码示例，用于检索 Apache Airflow CLI 令牌，然后在不同的 Amazon MWAA 环境中调用另一个 DAG。 • the section called “ DAGs 在不同的环境中调用”	2022 年 8 月 16 日
新示例代码	已添加最新说明和新的 DAG，用于查询环境的 Aurora PostgreSQL 以获取元数据信息，将结果写入 CSV 文件并将文件存储在 Amazon S3 中。 • the section called “将环境元数据导出到 Amazon S3 上”	2022 年 8 月 12 日
新示例代码	添加了更新的说明和新的 DAG，可在运行时刷新 AWS CodeArtifact 令牌并将结果存储在 Amazon S3 中。 • the section called “在运行时刷新 AWS CodeArtifact 令牌”	2022 年 8 月 3 日
新示例代码	已添加最新说明和 DAG 代码示例，用于在 Amazon MWAA 中使用 `ECSOperator` 。 • the section called “使用 `ECSOperator` ”	2022 年 7 月 26 日

新示例代码	已添加最新说明和 DAG 代码示例，用于在 Amazon MWAA 中使用 SSHOperator 。 • the section called “使用 SSHOperator ”	2022 年 7 月 15 日
新示例代码	已添加在 Amazon MWAA 中使用 dbt Postgres 的新说明和 DAG 代码示例。 • the section called “在 Amazon MWAA 中使用 dbt”	2022 年 6 月 17 日
新主题和用例	已添加新的说明和 DAG 代码示例，以对具有公有和私有访问权限的 Amazon MWAA 环境使用 Python Wheel 文件安装依赖项。 • 使用 Python Wheel 管理依赖项	2022 年 5 月 13 日
新主题和用例	添加了有关选择亚马逊 MWAA 向哪些 Apache Airflow 指标发送的新指南。 CloudWatch • 选择报告哪些 Apache Airflow 指标	2022 年 4 月 19 日
新指南	Amazon MWAA 提供了迁移指南，用于从自管理部署以及现有 Amazon MWAA 环境中迁移 Apache Airflow 工作流程。 • 《Amazon MWAA 迁移指南》	2022 年 3 月 7 日

新主题和用例	已添加新安全最佳实践，以与 Apache Airflow 配合使用，包括用于检测 Apache Airflow 用户权限更改的解决方案。 • the section called “Apache Airflow 中的安全最佳实践”	2022 年 2 月 18 日
新示例代码	已添加新代码示例，以使用 Pendulum 创建时区感知型 DAG，并已澄清如何使用自定义插件更改创建 Apache Airflow 日志的时区。 • the section called “更改 DAG 的时区”	2022 年 2 月 11 日
Apache Airflow v2.2.2 发布	Amazon Managed Workflows for Apache Airflow 现在支持 Apache Airflow v2.2.2。从 v2.2 开始，Amazon MWAA 将直接在 Apache Airflow Web 服务器上安装 Python 程序包和自定义插件，让您可以更灵活地管理环境。有关更多信息，请参阅以下内容。 • Amazon MWAA 上的 Apache Airflow 版本. • Apache Airflow v2.2.2 变更日志，位于 Apache Airflow 文档网站上。	2022 年 1 月 27 日

新教程	已添加一个新教程，演示如何创建新的自定义 Apache Airflow 角色，并将该角色分配给从 IAM 映射的 Apache Airflow 用户，以限制该用户对指定 DAG 子集的访问权限。 • 教程：限制 Amazon MWAA 用户访问其中的一个子集 DAGs	2021 年 12 月 8 日
修复	已修复设置 `scheduler.min_file_process_interval` 的值以优化 CPU 使用率的最佳实践建议。已添加 IAM 策略示例，以在执行角色中授予对 Secrets Manager 资源的访问权限。已添加有关使用 Secrets Manager 条件密钥的故障排除主题。 • 性能调整计划程序如何解析 DAG • 向 Amazon MWAA 提供访问 Secrets Manager 密钥的权限 • 在 Amazon MWAA 执行角色中配置 Secrets Manager 的条件密钥	2021 年 11 月 22 日

新示例代码	已添加以下用于修改使用自定义插件处理 DAG 的时区的新代码示例，以及用于从 bash 运算符中调用 `dags backfill` Apache Airflow CLI 命令的新故障排除主题。 • the section called “更改 DAG 的时区” • 使用 bash 运算符回填 CLI 命令	2021 年 11 月 1 日
修复	修复了 Amazon ECS 操作员代码示例中的问题，并阐明了 Amazon MWAA 执行角色中允许环境访问日志中的 Amazon ECS 任务日志组所需的额外权限。 CloudWatch • Amazon ECS 运算符权限。	2021 年 10 月 26 日
新示例代码	已添加新的代码示例，用于查询 Aurora PostgreSQL 数据库以获取与 DAG 运行并将结果写入存储在 Amazon S3 上的 CSV 文件相关的信息。 • the section called “将环境元数据导出到 Amazon S3 上”.	2021 年 10 月 1 日
修复	已更正有关 Amazon MWAA 如何自动将新的和已更改的对象从目标 Amazon S3 存储桶同步到您的计划程序和工作线程的信息。 • DAG 文件夹的工作原理。	2021 年 10 月 1 日

现在支持	Amazon MWAA 现在支持 Apache Airflow 2.0+ 的额外提供程序包。要了解有关支持的程序包的更多信息，请参阅以下内容：	2021 年 9 月 24 日
新的命令和程序	添加了其他指导和 AWS CLI 命令示例，用于在使用无法访问互联网的 Amazon VPC 时创建 Amazon S3 网关终端节点： • 创建没有互联网访问权限的 Amazon VPC 网络。	2021 年 9 月 24 日
新主题和用例	已添加以下更改： • 已添加使用 Amazon 弹性容器服务运算符的新代码示例，请参阅 the section called “使用 ECSOperator ”。 • 已在配置 Apache Airflow 插件时出现的问题添加为新的故障排除主题，请参阅 the section called “插件”。	2021 年 9 月 19 日

新增支持区域	Amazon MWAA 现在已在以下区域推出： • 亚太地区（孟买）：ap-south-1 • 亚太地区（首尔）– ap-northeast-2 • 欧洲（伦敦）– eu-west-2 • 欧洲（巴黎）：eu-west-3 • 加拿大（中部）：ca-central-1 • 南美洲（圣保罗）：sa-east-1 有关可用区域和服务端点的更多信息，请参阅如下： • 在《AWS 一般参考》中的 Amazon MWAA 端点和配额	2021 年 8 月 31 日
新主题和用例	已添加以下更改： • 已更新示例策略，以允许 Amazon MWAA 提取账户级别的 Amazon S3 设置 (`s3:GetAccountPublicAccessBlock`)，请参阅 Amazon MWAA 执行角色。	2021 年 8 月 27 日

修复	已添加以下更改： • 修复了 CloudFormation 模板以对中的安全组使用自引用入站规则。创建 VPC 网络 • 修复了 CloudFormation 模板以对中的安全组使用自引用入站规则。Amazon MWAA 的快速入门教程	2021 年 8 月 27 日
新主题和用例	已添加以下更改： • 已将 DAG 修饰器添加到 Apache Airflow v2.0.2 支持的列表中，请参阅 Amazon MWAA 上的 Apache Airflow 版本。	2021 年 8 月 20 日
新主题和用例	已添加以下更改： • 已将 `celery.sync_parallelism` 用例添加到 Amazon MWAA 上的 Apache Airflow 的性能调整。 • 已将服务端点添加到配额页面，并将名称更改为 Amazon MWAA 服务端点和限额。 • 已根据用户反馈澄清网络先决条件，请参阅 开始使用 Amazon MWAA。 • 已将 `dags list-runs` 和 `dags next-execution` 移动至不支持的 Airflow CLI 命令，请参阅 Apache Airflow CLI 命令参考。	2021 年 8 月 13 日

新示例代码	已添加以下更改： • 已添加 bash 示例，用于设置、获取或删除 Apache Airflow v2.0.2 变量，请参阅 Apache Airflow CLI 命令参考。 • 已将 Apache Airflow v2.0.2 依赖项和 Airflow 连接示例添加到 将 Amazon RDS for Microsoft SQL Server 与 Amazon MWAA 一起使用。	2021 年 8 月 13 日
修复	已添加以下更改： • 已根据用户反馈修复 Python 代码示例，请参阅 使用 `SSHOperator` 创建 SSH 连接。	2021 年 8 月 13 日

新主题和用例	已添加以下更改： • 已将 `variables set` 移动至支持的 Airflow CLI 命令，请参阅 Apache Airflow CLI 命令参考。 • 已根据用户反馈将 v2.0.2 中更改的摘要从 Airflow 版本页面添加到 安装 Python 依赖项。 • 已根据用户反馈将 v2.0.2 中更改的摘要从 Airflow 版本页面添加到 Apache Airflow CLI 命令参考。 • 已根据用户反馈将 v2.0.2 中更改的摘要从 Airflow 版本页面添加到 连接类型概述。 • 已根据用户反馈将 v2.0.2 中更改的摘要从 Airflow 版本页面添加到 安装自定义插件。 • 已根据用户反馈将 v2.0.2 中更改的摘要从 Airflow 版本页面添加到 添加或更新 DAG。	2021 年 8 月 6 日
新示例代码	已添加以下更改： • 已将 Apache Airflow v2.0.2 示例代码添加到 使用 DAG 在 CLI 中导入变量。 • 已将 Apache Airflow v2.0.2 示例代码添加到 使用 Lambda DAGs 函数进行调用。	2021 年 8 月 6 日

新主题和用例	已添加以下更改： • 已添加“我在 Airflow UI 中找不到我的连接”的故障排除主题，请参阅 Amazon MWAA 故障排除。 • 已将 Amazon MWAA 支持的 Amazon VPC 列表添加到 关于在 Amazon MWAA 上联网。	2021 年 7 月 29 日
修复	已添加以下更改： • 已根据用户反馈修复 Python 代码示例，以打印 Web 登录令牌，请参阅 创建 Apache Airflow Web 服务器访问令牌。 • 已根据用户反馈修复 Snowflake 连接主题，以便为仓库参数使用单引号，请参阅 Amazon MWAA 故障排除。	2021 年 7 月 29 日
已删除或移动的主题	已添加以下更改： • 已重组现有页面，以包含所有监控和指标文档页面，请参阅 Amazon MWAA 的监控和指标。 • 已将 中的 Apache Airflow 环境指标 CloudWatch 移动至监控和指标导航菜单。	2021 年 7 月 23 日

新指南	已添加以下更改： • 已创建 安装在 Amazon MWAA 环境中的 Apache Airflow 提供程序包。 • 已创建 Amazon MWAA 上的监控概述。 • 已创建 访问审核日志 AWS CloudTrail。 • 已创建 访问 Amazon 中的 Airflow 日志 CloudWatch。	2021 年 7 月 23 日
修复	已添加以下更改： • 已根据用户反馈修复 Python 代码示例，以按正确顺序生成 Airflow 连接字符串，并已添加端口参数，请参阅 使用密钥配置 Apache Airflow 连接 AWS Secrets Manager。 • 已根据用户反馈在添加本地安装解压缩包的步骤，请参阅 使用 Oracle 创建自定义插件。	2021 年 7 月 23 日

新主题和用例	已添加以下更改： • 为 AWS DMS 操作员添加了主题，网址为Amazon MWAA 常见问题解答。 • 已将远程日志错误的故障排除主题添加到 Amazon MWAA 故障排除。 • 已将 `variables set` 移动至不支持的 Airflow CLI 命令，请参阅 Apache Airflow CLI 命令参考。	2021 年 7 月 16 日
新主题和用例	已添加以下更改： • 已根据用户反馈添加创建 requirements.txt 文件的顺序步骤，请参阅 安装 Python 依赖项。 • 已根据用户反馈添加创建 plugins.zip 文件的顺序步骤，请参阅 安装自定义插件。 • 已在《Amazon MWAA API 参考指南》中将整个用户指南中的交叉引用链接添加到 API 参考指南。 • 已添加为何插件未列在 Airflow 2.0 管理员 > 插件菜单中的主题，请参阅 Amazon MWAA 常见问题解答。	2021 年 7 月 9 日

新指南	已添加以下更改： • 已创建 删除 Amazon S3 上的文件。	2021 年 7 月 9 日
新主题和用例	已添加以下更改： • 已在 使用由客户托管的密钥进行加密 中添加支持的值列表。 • 根据用户反馈已更新并澄清私有存储库 URL 的示例，请参阅 在 requirements.txt 中管理 Python 依赖项。	2021 年 7 月 2 日
新示例代码	已添加以下更改： • 在 Apache Airflow v1.10.12 中添加了使用私钥 AWS Secrets Manager 进行 SSH 连接的示例代码。使用 `SSHOperator` 创建 SSH 连接	2021 年 7 月 2 日
新主题和用例	已添加以下更改： • 向中添加了 FinishedTaskInstances 指标 StartedTaskInstances 和指标中的 Apache Airflow 环境指标 CloudWatch。	2021 年 6 月 25 日
新示例代码	已添加以下更改： • 已在 将 Amazon MWAA 与 Amazon EKS 一起使用 中添加 Apache Airflow v2.0.2 示例代码。	2021 年 6 月 25 日

新指南	已添加以下更改：	2021 年 6 月 25 日

- 已创建 Amazon MWAA 上的 Apache Airflow 的性能调整。

新主题和用例	已添加以下更改：	2021 年 6 月 18 日

- 已在支持的 Apache Airflow v2.0.2 CLI 命令中添加 `connections add` 和 `connections delete`，请参阅 Apache Airflow CLI 命令参考。
- 补充说，中可用的最新版本 CloudFormation 是 Apache Airflow v2.0.2，网址为。Amazon MWAA 的快速入门教程
- 已将有关在 Apache Airflow 工作线程上存储临时数据的问题添加到 Amazon MWAA 常见问题解答。
- 已将“执行程序报告任务实例 %s 已完成”错误的主题添加到 Amazon MWAA 故障排除。
- 已将“服务器意外关闭连接”日志的主题添加到 Amazon MWAA 故障排除。
- 已将在通往堡垒主机的 SSH 隧道上运行 CLI 命令的示例添加到 创建 Apache Airflow CLI 令牌。
- 已将随机生成的用户名的主题添加到 Amazon MWAA 故障排除。
- 已将在 CLI 中运行 DAG 时出现 503 错误的主题添加到 Amazon MWAA 故障排除。

- 已添加在 Apache Airflow v2.0.2 中自定义插件的主题，这些插件需要一个 `core.lazy_load_plugins : False` 的 Airflow 配置选项才能在每个 Airflow 进程开始时加载插件，以覆盖该版本的默认设置，请参阅 在 Amazon MWAA 上使用 Apache Airflow 配置选项。
- 已在 Apache Airflow v2.0.2 插件示例代码中添加 Apache Airflow 配置选项步骤，请参阅 使用 Apache Hive 和 Hadoop 创建自定义插件。
- 已在 Apache Airflow v2.0.2 插件示例代码中添加 Apache Airflow 配置选项步骤。
- 已在 Apache Airflow v2.0.2 插件中添加 Apache Airflow 配置选项步骤示例代码，请参阅 为 Apache Airflow PythonVirtualenvOperator 创建自定义插件。
- 已在 Apache Airflow v2.0.2 插件中添加 Apache Airflow 配置选项步骤示例代码，请参阅 使用 Oracle 创建自定义插件。

新示例代码	已添加以下更改： • 已在 使用 AWS Secrets Manager 中的密钥进行 Apache Airflow Snowflake 连接 中添加 Apache Airflow Snowflake 连接的示例代码。	2021 年 6 月 18 日

新主题和用例	已添加以下更改：	2021 年 6 月 2 日

- 已添加服务器端加密指导添加到 为 Amazon MWAA 创建 Amazon S3 存储桶。。
- 已将 Apache Airflow v2.0.2 的密钥后端添加到 使用密钥配置 Apache Airflow 连接 AWS Secrets Manager。
- 已将有关 Apache Airflow 工作线程配额增加请求的问题添加到 Amazon MWAA 常见问题解答。
- 已将使用哪些指标来确定是否扩展 Apache Airflow 工作线程的问题添加到 Amazon MWAA 常见问题解答。
- 在中添加了有关创建自定义指标 CloudWatch 的问题Amazon MWAA 常见问题解答。
- 已将为具有私有路由的 VPC 的 Amazon S3 VPC 接口端点启用私有 IP 地址的步骤添加到 使用私有路由在 Amazon VPC 中创建所需的 VPC 服务端点。
- 已添加使用本地端口转发设置 SSH 隧道的选项，请参阅 教程：使用 Linux 堡垒主机配置私有网络访问权限。

新示例代码	已添加以下更改： • 为查询 Amazon Aurora PostgreSQL 元数据数据库并向亚马逊发布自定义指标的 DAG 添加了示例代码，网址为。 CloudWatch 使用 DAG 在 CloudWatch 中编写自定义指标	2021 年 6 月 2 日
新指南	已添加以下更改： • 已创建有关如何在 Apache Airflow UI 中互换使用连接模板的指南，请参阅 连接类型概述。	2021 年 6 月 2 日
修复	已添加以下更改： • 在选项三：创建无法访问互联网的 VPC 网络的模板中添加了 Apache Airflow VPC 终端节点。 CloudFormation 创建 VPC 网络	2021 年 6 月 2 日

Apache Airflow v2.0.2 发布	Apache Airflow v2.0.2 正式版发布	2021 年 5 月 26 日

- 已创建 Amazon MWAA 上的 Apache Airflow 版本。
- 已创建 中的 Apache Airflow 环境指标 CloudWatch。
- 已将 Apache Airflow v2.0.2 特定版本的链接添加到 在 Amazon MWAA 上使用 Apache Airflow 配置选项。
- 已将 Apache Airflow v2.0.2 特定版本的指导添加到 安装 Python 依赖项。
- 已将 Apache Airflow v2.0.2 特定版本的指导添加到 在 requirements.txt 中管理 Python 依赖项。
- 已将 Apache Airflow v2.0.2 示例插件添加到 安装自定义插件。
- 已将 Apache Airflow v2.0.2 示例代码添加到 在 Amazon MWAA 环境中清理 Aurora PostgreSQL 数据库。
- 已将 Apache Airflow v2.0.2 示例代码添加到 使用 AWS Secrets Manager 中的密钥进行 Apache Airflow 连接。
- 已将 Apache Airflow v2.0.2 示例代码添加到 为 Apache Airflow PythonVirtualenvOperator 创建自定义插件。

	• 已将 Apache Airflow v2.0.2 命令添加到 Apache Airflow CLI 命令参考。 • 已将 Apache Airflow v2.0.2 脚本添加到 创建 Apache Airflow CLI 令牌。 • 已将关于 Amazon MWAA 默认使用最新的 Apache Airflow 版本的说明添加到 创建 Amazon MWAA 环境。	
新主题和用例	已添加以下更改： • 已将排查卡住或未运行的 Airflow 任务的指导添加到 Amazon MWAA 故障排除。	2021 年 5 月 14 日
修复	已添加以下更改： • 我们已经更新了示例插件代码，以使用最新的 Java 版本，请参阅 使用 Apache Hive 和 Hadoop 创建自定义插件。以前，其版本为 `os.environ["JAVA_HOME"]="/usr/lib/jvm/jre-1.8.0-openjdk-1.8.0.272.b10-1.amzn2.0.1.x86_64"` 。	2021 年 5 月 12 日
已删除或移动的主题	已添加以下更改： • 已按类别将 Amazon MWAA 故障排除 中的主题移至新页面。	2021 年 5 月 10 日

新主题和用例	已添加以下更改： • 已将 Amazon S3 存储桶概述添加到 在 Amazon MWAA 上使用 DAG。	2021 年 5 月 10 日
已删除或移动的主题	已添加以下更改： • 已将 访问 Apache Airflow 移动至主级导航，并已添加 创建 Apache Airflow Web 服务器访问令牌、创建 Apache Airflow CLI 令牌 和 Apache Airflow CLI 命令参考 的页面。	2021 年 5 月 7 日
新主题和用例	已添加以下更改： • 已在《Apache Airflow 参考指南》中添加特定版本的链接，其中包含所有支持和不支持的 Airflow CLI 命令，请参阅 Apache Airflow CLI 命令参考。 • 已将特定版本的链接添加到所有配置选项的《Apache Airflow 参考指南》，请参阅 在 Amazon MWAA 上使用 Apache Airflow 配置选项。 • 将 Amazon MWAA CLI 实用工具添加到 在 requirements.txt 中管理 Python 依赖项。	2021 年 5 月 7 日

新主题和用例	已添加以下更改： • 已添加如何构建 plugins.zip 的平面和嵌套示例，请参阅安装自定义插件。 • 将 Amazon MWAA CLI 实用工具添加到 添加或更新 DAG、安装自定义插件 和 安装 Python 依赖项 页面。 • 根据 安装自定义插件 和 安装 Python 依赖项 页面中的用户反馈将内容重组为概述，上传到 Amazon S3，然后在 Amazon MWAA 上安装。 • 已添加一个示例使用案例，用于创建所需的 VPC 端点并将其连接到无法访问互联网的现有 Amazon VPC，请参阅 关于在 Amazon MWAA 上联网。	2021 年 4 月 30 日
新示例代码	已添加以下更改： • 已添加在 Secrets Manager 中为 Apache Airflow 变量使用密钥的示例代码，请参阅为 Apache Airflow 变量使用 AWS Secrets Manager 中的密钥。	2021 年 4 月 30 日
新指南	已添加以下更改： • 已创建 使用私有路由在 Amazon VPC 中创建所需的 VPC 服务端点。	2021 年 4 月 30 日

修复	已添加以下更改： • 我们已经将 `core.default_ui_timezone` 更新为 `webserver.default_ui_timezone`，请参阅在 Amazon MWAA 上使用 Apache Airflow 配置选项。	2021 年 4 月 30 日
新主题和用例	已添加以下更改： • 已将 SSH 隧道的 Windows（Putty）步骤添加到 教程：使用 Linux 堡垒主机配置私有网络访问权限。 • 已将仅与 Apache Airflow 2.0 兼容的 `apache-airflow-providers-amazon` 的主题添加到 Amazon MWAA 故障排除。	2021 年 4 月 23 日
新示例代码	已添加以下更改： • 已添加在 Secrets Manager 中使用密钥进行 Apache Airflow 连接的示例代码，请参阅 使用 AWS Secrets Manager 中的密钥进行 Apache Airflow 连接。	2021 年 4 月 23 日

新指南	已添加以下更改：	2021 年 4 月 23 日

- 已创建 关于在 Amazon MWAA 上联网。
- 已创建 Amazon MWAA 上的 VPC 安全。
- 已创建 在 Amazon MWAA 上管理对服务特定 Amazon VPC 端点的访问。

新主题和用例	已添加以下更改：	2021 年 4 月 16 日

- 添加了一个新 CloudFormation 模板，用于创建无法访问互联网的 Amazon VPC 网络创建 VPC 网络。
- 添加了新教程来创建 AWS Client VPN in 教程：使用配置私有网络访问权限 AWS Client VPN。
- 已根据用户反馈将网络访问页面的名称更改为 Apache Airflow 访问模式，并已简化 Apache Airflow 访问模式 中的文档。
- 已简化文档，仅包含 Amazon VPC 入门信息和基于用户反馈的模板，请参阅创建 VPC 网络。
- 向添加了 BigQuery 操作员解决方法Amazon MWAA 故障排除。
- 已将 Apache Airflow v1.10.12 约束文件最佳实践添加到 安装 Python 依赖项。

新示例代码	已添加以下更改： • 已添加使用 Oracle 创建自定义插件的示例代码，请参阅使用 Oracle 创建自定义插件。 • 已添加示例代码，用于创建生成运行时环境变量的自定义插件。 •	2021 年 4 月 16 日
新主题和用例	已添加以下更改： • 已将在 VPC 安全组上的自引用规则要求的主题添加到 Amazon MWAA 常见问题解答。 • 已将自定义插件目录和大小限制添加到 安装自定义插件。 • 已将要求目录和大小限制添加到 安装 Python 依赖项。 • 已澄清 `foo.user` 和 `foo.pass` 的 Apache Airflow 配置选项，请参阅在 requirements.txt 中管理 Python 依赖项。 • 已将配置选项概述添加到在 Amazon MWAA 上使用 Apache Airflow 配置选项。	2021 年 4 月 9 日

新示例代码	已添加以下更改： • 已添加使用 PythonVirtualenvOperator 创建自定义插件的示例代码，请参阅 为 Apache Airflow PythonVirtualenvOperator 创建自定义插件。 • 已添加创建带有 Apache Hive 和 Hadoop 的自定义插件的示例代码，请参阅 使用 Apache Hive 和 Hadoop 创建自定义插件。	2021 年 4 月 9 日
修复	已添加以下更改： • 我们更新了 requirements.txt 的格式，并添加了一个与 Apache Airflow v1.10.12 兼容的示例，请参阅 安装 Python 依赖项。	2021 年 3 月 31 日
新主题和用例	已添加以下更改： • 已将移除 requirements.txt 或 plugins.zip 的变通方法添加到 Amazon MWAA 常见问题解答。 • 已将环境上的 SSH 的 bash 解决方法添加到 Amazon MWAA 常见问题解答。 • 向中添加了 CloudTrail ResourceAlreadyExistsException 错误主题Amazon MWAA 故障排除。	2021 年 3 月 26 日

新主题和用例	已添加以下更改：	2021 年 3 月 19 日

- 添加了用于的 AWS 服务列表Amazon MWAA 执行角色。
- 添加了用于的 AWS 服务列表Service-linked 亚马逊 MWAA 的角色。
- 已将 Amazon MWAA 的 Python 3.7 版本问题添加到 Amazon MWAA 常见问题解答。
- 已将 `PythonVirtualenvOperator` 的问题添加到 Amazon MWAA 常见问题解答。
- 已添加故障排除脚本，作为与 VPC 和环境配置相关的所有主题的后续步骤，请参阅 Amazon MWAA 故障排除。
- 已澄清 Linux 堡垒机必须与环境位于同一区域的文档，请参阅 教程：使用 Linux 堡垒主机配置私有网络访问权限。

新指南	已添加以下更改： • 为 at 创建了 Apache Airflow 连接指南 AWS Secrets Manager 使用密钥配置 Apache Airflow 连接 AWS Secrets Manager • 使用 CloudFormation 模板创建了快速入门教程，用于在中创建 Amazon VPC 基础设施、Amazon S3 存储桶和 Amazon MWAA 环境。Amazon MWAA 的快速入门教程	2021 年 3 月 19 日
新主题和用例	已添加以下更改： • 已将创建 Amazon S3 存储桶故障排除主题添加到 Amazon MWAA 故障排除。 • 已将创建和附加 JSON 策略的步骤添加到 Amazon MWAA 执行角色。	2021 年 3 月 12 日
新示例代码	已添加以下更改： • 已将在触发 DAG 时添加配置的示例代码添加到 访问 Apache Airflow。	2021 年 3 月 12 日
新指南	已添加以下更改： • 已创建最佳实践指南，请参阅 在 requirements.txt 中管理 Python 依赖项。	2021 年 3 月 12 日

新主题和用例	已添加以下更改： • 向中添加了 Google/GCP/BigQuery 疑难解答主题Amazon MWAA 故障排除。 • 已将 Cython 故障排除主题添加到 Amazon MWAA 故障排除。 • 已将 MySQL 故障排除主题添加到 Amazon MWAA 故障排除。 • 已将 5xx Web 服务器错误故障排除主题添加到 Amazon MWAA 故障排除。	2021 年 3 月 5 日
现在支持	已添加以下更改： • 以前`backend_kwargs` ，不支持，你需要一种变通方法来覆盖 Secrets Manager 函数调用。 AWS Secrets Manager 现在 支持 `backend_kwargs` 。请参阅中的 AWS Secrets Manager 疑难解答主题Amazon MWAA 故障排除。	2021 年 3 月 4 日
修复	已添加以下更改： • 我们更新了每个环境类的大小，以反映实际的 GB，请参阅 配置 Amazon MWAA 环境类。	2021 年 3 月 4 日

新主题和用例	已添加以下更改： • 已将使用 VPC 端点策略访问私有网络的权限添加到 Apache Airflow 访问模式。 • 已将创建环境故障排除主题的其他检查添加到 Amazon MWAA 故障排除。 • 已将访问 `requirements.txt` 的日志的步骤添加到 安装 Python 依赖项。	2021 年 2 月 26 日
新主题和用例	已添加以下更改： • 将 Apache Hive 用例添加到 安装 Python 依赖项。 • 已澄清需要将 Apache Airflow 程序包所需的依赖项包含在 `requirements.txt` 文件中的文档，请参阅 安装 Python 依赖项。 • 已将更新 requirements.txt 故障排除主题添加到 Amazon MWAA 故障排除。	2021 年 2 月 25 日
新教程	已添加以下更改： • 已将私有网络教程添加到 教程：使用 Linux 堡垒主机配置私有网络访问权限。	2021 年 2 月 22 日

新主题和用例	已添加以下更改： • 已将私有和公用网络配置添加到 Apache Airflow 访问模式。 • 已将开发组用例和用户场景添加到 Amazon MWAA 执行角色。	2021 年 2 月 22 日
新示例代码	已添加以下更改： • 已将 Web 登录令牌和 CLI 令牌的示例 Python 脚本添加到访问 Apache Airflow。 • 已将在其他环境中触发 DAG 的示例代码添加到 Amazon MWAA 的代码示例。 • 已将使用 Lambda 函数触发 DAG 的示例代码添加到 使用 Lambda DAGs 函数进行调用。	2021 年 2 月 22 日
新的命令和程序	已添加以下更改： • 在 访问 Apache Airflow 中，将分步过程添加到所有脚本中。	2021 年 2 月 22 日

新示例代码	已添加以下更改： • 已更新 Web 登录令牌的 curl 示例，请参阅 访问 Apache Airflow。 • 已添加用于连接到 Amazon RDS Microsoft SQL Server 的示例代码，请参阅 将 Amazon RDS for Microsoft SQL Server 与 Amazon MWAA 一起使用。	2021 年 2 月 17 日
新的命令和程序	已添加以下更改： • 向在 Amazon MWAA 上使用 DAG页面添加了 AWS CLI 命令。 • Apache Airflow 不支持在 CLI 命令中使用序列化的 DAG。由于 CLI 在 Web 服务器上运行，出于安全考虑，该服务器没有插件或要求，因此任何带有 plugins.zip 或 requirements.txt 的 MWAA 环境都不支持这些命令。已将 Apache Airflow `list_dags` 和 `backfill` 命令移动到不支持的命令，请参阅 访问 Apache Airflow。	2021 年 2 月 17 日
GitHub 发射	用户指南文档现已开源 GitHub。在任意页面 GitHub上选择 “编辑此页面”。	2021 年 2 月 17 日

新主题和用例	已添加以下更改： • 已将 Step Functions v. Amazon MWAA 用例的问题添加到 Amazon MWAA 故障排除。 • 已将访问策略添加到 访问 Amazon MWAA 环境。 • 已澄清可以指定任何支持的 Apache Airflow 配置选项的文档，请参阅 在 Amazon MWAA 上使用 Apache Airflow 配置选项。 • 已澄清文档，即如果一个可用区中的 Fargate 容器出现故障，MWAA 将切换到另一个可用区中的另一个容器，请参阅 创建 VPC 网络。	2021 年 2 月 12 日
新主题和用例	已添加以下更改： • 增加了 配置 Amazon MWAA 环境类。	2021 年 2 月 5 日
已删除或移动的主题	已添加以下更改： • 已删除 Amazon S3 存储桶名称必须以开头`airflow-`的要求，请参阅 开始使用 Amazon MWAA。 • 已将 访问 Amazon MWAA 环境 和 Amazon MWAA 执行角色 移动至 管理对 Amazon MWAA 环境的访问。	2021 年 2 月 4 日

亚马逊 MWAA CloudFormation	更新参数以在 Amazon MWAA CloudFormation 上创建环境。 • 移除 SubnetList。 • 移除 TagList。 • 添加 NetworkConfiguration。 • 添加 TagMap。 • 添加创建环境请求示例。	2021 年 2 月 4 日
新主题和用例	已添加以下更改： • 已将电子邮件配置示例添加到 在 Amazon MWAA 上使用 Apache Airflow 配置选项。 • 向中添加了 PostgresHook 疑难解答主题Amazon MWAA 故障排除。 • 向中添加了 AWS Secrets Manager 疑难解答主题Amazon MWAA 故障排除。 • 已将高性能用例添加到 配置 Amazon MWAA Worker 节点自动扩缩。	2021 年 1 月 29 日
Amazon MWAA 发布	Amazon MWAA 正式版发布。 • 用户指南文档 • CloudFormation 文档	2020 年 11 月 24 日

本文属于机器翻译版本。若本译文内容与英语原文存在差异，则一律以英文原文为准。