aws

User Guide

Amazon Managed Workflows for Apache Airflow

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon Managed Workflows for Apache Airflow: User Guide

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

What Is Amazon MWAA? 1
Features 1
Architecture 2
Integration 4
Supported versions 4
What's next? 4
Quick start 5
In this tutorial 5
Prerequisites 6
Step one: Save the AWS CloudFormation template locally 6
Step two: Create the stack using the AWS CLI 16
Step three: Upload a DAG to Amazon S3 and run in the Apache Airflow UI 17
Step four: View logs in CloudWatch Logs 18
What's next? 18
Get started 19
Prerequisites 19
About this guide 19
Before you begin 20
Available regions 20
Create a bucket 22
Before you begin 22
Create the bucket 22
What's next? 24
Create the VPC network 24
Prerequisites 25
Before you begin 25
Options to create the Amazon VPC network 25
What's next? 37
Create an environment 38
Before you begin 38
Apache Airflow versions 38
Create an environment 40
What's next? 24
Managing access 45

Accessing an Amazon MWAA environment ... 45
How it works ... 46
Full console access ... 47
Full API access ... 54
Read-only console access ... 58
Apache Airflow UI access ... 58
Apache Airflow Rest API access ... 59
Apache Airflow CLI access ... 60
Creating a JSON policy ... 61
Example use case ... 61
What's next? ... 63
Service-linked role ... 64
Service-linked role permissions for Amazon MWAA ... 64
Creating a service-linked role for Amazon MWAA ... 67
Editing a service-linked role for Amazon MWAA ... 68
Deleting a service-linked role for Amazon MWAA ... 68
Supported regions for Amazon MWAA service-linked roles ... 68
Policy updates ... 68
Execution role ... 69
Execution role overview ... 70
Create a new role ... 72
View and update an execution role policy ... 72
Grant access to Amazon S3 bucket with account-level public access block ... 74
Use Apache Airflow connections ... 75
Sample policies ... 75
What's next? ... 81
Cross-service confused deputy prevention ... 81
Apache Airflow access modes ... 82
Apache Airflow access modes ... 83
Access modes overview ... 85
Setup for private and public access modes ... 86
Accessing the VPC endpoint for your Apache Airflow Web server (private network access) ... 88
Accessing Apache Airflow ... 89
Prerequisites ... 89
Access ... 89

AWS CLI ... 90
Open the Apache Airflow UI ... 90
Logging into Apache Airflow ... 90
Create a web server access token ... 90
Prerequisites ... 91
Using the AWS CLI ... 91
Using a bash script ... 92
Using a Python script ... 92
What's next? ... 93
Setting up a custom domain ... 93
Configure the custom domain ... 94
Set up the networking infrastructure ... 95
Apache Airflow CLI token ... 99
Prerequisites ... 100
Using the AWS CLI ... 101
Using a curl script ... 101
Using a bash script ... 103
Using a Python script ... 104
What's next? ... 107
Using the Apache Airflow REST API ... 107
Granting access to the Apache Airflow REST API: `airflow:InvokeRestApi` ... 109
Calling the Apache Airflow REST API ... 110
Creating a web server session token and calling the Apache Airflow REST API ... 111
Apache Airflow CLI command reference ... 114
Prerequisites ... 114
What's changed in v2 ... 115
Supported CLI commands ... 115
Sample code ... 118
Managing connections ... 121
Overview ... 121
Apache Airflow packages ... 121
Provider packages for Apache Airflow v2.10.3 connections ... 122
Provider packages for Apache Airflow v2.10.1 connections ... 123
Provider packages for Apache Airflow v2.9.2 connections ... 124
Provider packages for Apache Airflow v2.8.1 connections ... 125
Provider packages for Apache Airflow v2.7.2 connections ... 126

Provider packages for Apache Airflow v2.6.3 connections ... 127
Provider packages for Apache Airflow v2.5.1 connections ... 128
Provider packages for Apache Airflow v2.4.3 connections ... 129
Provider packages for Apache Airflow v2.2.2 connections ... 129
Provider packages for Apache Airflow v2.0.2 connections ... 130
Specifying newer provider packages ... 130
Connection types ... 131
Example connection URI string ... 132
Example connection template ... 132
Example using an HTTP connection template for a Jdbc connection ... 134
Configuring Secrets Manager ... 135
Step one: Provide Amazon MWAA with permission to access Secrets Manager secret keys ... 136
Step two: Create the Secrets Manager backend as an Apache Airflow configuration option ... 137
Step three: Generate an Apache Airflow AWS connection URI string ... 139
Step four: Add the variables in Secrets Manager ... 141
Step five: Add the connection in Secrets Manager ... 142
Sample code ... 144
Resources ... 144
What's next? ... 144
Managing environments ... 145
Configuring the environment class ... 145
Environment capabilities ... 145
Apache Airflow Schedulers ... 148
Configuring worker auto scaling ... 148
How worker scaling works ... 149
Using the Amazon MWAA console ... 149
Example high performance use case ... 150
Troubleshooting tasks stuck in the running state ... 151
What's next? ... 152
Configuring web server auto scaling ... 152
How web server scaling works ... 152
Using the Amazon MWAA console ... 152
Using configuration options ... 153
Prerequisites ... 154

How it works ... 154
Using configuration options to load plugins in Apache Airflow v2 ... 155
Configuration options overview ... 155
Configuration reference ... 156
Examples and sample code ... 162
What's next? ... 164
Upgrading the version ... 164
Upgrade your workflow resources ... 165
Specify the new version ... 166
Using a startup script ... 167
Configure a startup script ... 167
Install Linux runtimes ... 171
Set environment variables ... 172
Working with DAGs ... 176
Amazon S3 bucket overview ... 176
Adding or updating DAGs ... 177
Prerequisites ... 177
How it works ... 178
What's changed in v2 ... 178
Testing DAGs using the Amazon MWAA CLI utility ... 179
Uploading DAG code to Amazon S3 ... 179
Specifying the path to a DAGs folder ... 180
Viewing changes on your Apache Airflow UI ... 181
What's next? ... 181
Installing custom plugins ... 181
Prerequisites ... 182
How it works ... 183
When to use the plugins ... 183
Custom plugins overview ... 184
Examples of custom plugins ... 184
Creating a plugins.zip file ... 194
Uploading `plugins.zip` to Amazon S3 ... 195
Installing custom plugins on your environment ... 196
Example use cases for plugins.zip ... 197
What's next? ... 197
Installing Python dependencies ... 197

Prerequisites ... 198
How it works ... 199
Python dependencies overview ... 199
Creating a requirements.txt file ... 200
Uploading `requirements.txt` to Amazon S3 ... 202
Installing Python dependencies on your environment ... 203
Viewing logs for your `requirements.txt` ... 204
What's next? ... 205
Deleting files on Amazon S3 ... 205
Prerequisites ... 205
Versioning overview ... 206
How it works ... 206
Deleting a DAG on Amazon S3 ... 207
Removing "current" plugins.zip or requirements.txt ... 207
Delete "non-current" plugins.zip or requirements.txt ... 207
Deleting files with lifecycles ... 208
Example lifecycle policy ... 208
What's next? ... 209
Networking ... 210
About networking ... 210
Terms ... 211
What's supported ... 211
VPC infrastructure overview ... 211
Example use cases for an Amazon VPC and Apache Airflow access mode ... 214
Security in your VPC ... 216
Terms ... 217
Security overview ... 217
Network access control lists (ACLs) ... 218
VPC security groups ... 218
VPC endpoint policies (private routing only) ... 220
Managing access to VPC endpoints ... 221
Pricing ... 222
VPC endpoint overview ... 222
Permission to use other AWS services ... 223
Viewing VPC endpoints ... 223

Accessing the VPC endpoint for your Apache Airflow Web server (private network access) ... 225
VPC service endpoints in private Amazon VPCs ... 227
Pricing ... 227
Private network and private routing ... 228
(Required) VPC endpoints ... 229
Attaching the required VPC endpoints ... 229
(Optional) Enable private IP addresses for your Amazon S3 VPC interface endpoint ... 233
Managing your own Amazon VPC endpoints ... 234
Creating an environment in a shared Amazon VPC ... 234
Tutorials ... 244
Tutorial: AWS Client VPN ... 244
Private network ... 245
Use cases ... 246
Before you begin ... 246
Objectives ... 246
(Optional) Step one: Identify your VPC, CIDR rules, and VPC security(s) ... 247
Step two: Create the server and client certificates ... 248
Step three: Save the AWS CloudFormation template locally ... 249
Step four: Create the Client VPN AWS CloudFormation stack ... 251
Step five: Associate subnets to your Client VPN ... 251
Step six: Add an authorization ingress rule to your Client VPN ... 252
Step seven: Download the Client VPN endpoint configuration file ... 252
Step eight: Connect to the AWS Client VPN ... 254
What's next? ... 255
Tutorial: Linux Bastion Host ... 255
Private network ... 255
Use cases ... 256
Before you begin ... 257
Objectives ... 257
Step one: Create the bastion instance ... 257
Step two: Create the ssh tunnel ... 259
Step three: Configure the bastion security group as an inbound rule ... 260
Step four: Copy the Apache Airflow URL ... 261
Step five: Configure proxy settings ... 261
Step six: Open the Apache Airflow UI ... 264

What's next? ... 264
Tutorial: Restricting users to a subset of DAGs ... 264
Prerequisites ... 265
Step one: Provide Amazon MWAA web server access to your IAM principal with the default `Public` Apache Airflow role. ... 265
Step two: Create a new Apache Airflow custom role ... 266
Step three: Assign the role you created to your Amazon MWAA user ... 267
Next steps ... 268
Related resources ... 268
Tutorial: Automate managing your own environment endpoints ... 268
Prerequisites ... 269
Create the Amazon VPC ... 269
Create the Lambda function ... 270
Create the EventBridge rule ... 270
Create the environment ... 271
Code examples ... 273
Import variables DAG ... 274
Version ... 274
Prerequisites ... 274
Permissions ... 274
Dependencies ... 274
Code sample ... 275
What's next? ... 276
Using the `SSHOperator` ... 276
Version ... 277
Prerequisites ... 277
Permissions ... 277
Requirements ... 278
Copy your secret key to Amazon S3 ... 278
Create a new Apache Airflow connection ... 278
Code sample ... 279
Apache Airflow Snowflake connection in Secrets Manager ... 281
Version ... 281
Prerequisites ... 281
Permissions ... 281
Requirements ... 282

Code sample ... 282
What's next? ... 283
Using a DAG to write custom metrics ... 283
Version ... 284
Prerequisites ... 284
Permissions ... 284
Dependencies ... 284
Code example ... 284
Aurora PostgreSQL database cleanup ... 287
Version ... 288
Prerequisites ... 288
Dependencies ... 288
Code sample ... 288
Exporting environment metadata to Amazon S3 ... 291
Version ... 292
Prerequisites ... 292
Permissions ... 292
Requirements ... 293
Code sample ... 293
Using an Apache Airflow variable in Secrets Manager ... 295
Version ... 296
Prerequisites ... 296
Permissions ... 296
Requirements ... 296
Code sample ... 297
What's next? ... 298
Using an Apache Airflow connection in Secrets Manager ... 298
Version ... 298
Prerequisites ... 298
Permissions ... 299
Requirements ... 296
Code sample ... 299
What's next? ... 302
Custom plugin with Oracle ... 302
Version ... 303
Prerequisites ... 303

Permissions ... 303
Requirements ... 303
Code sample ... 304
Create the custom plugin ... 305
Airflow configuration options ... 308
What's next? ... 308
Custom plugin with environment variables ... 308
Version ... 309
Prerequisites ... 309
Permissions ... 309
Requirements ... 309
Custom plugin ... 309
Plugins.zip ... 310
Airflow configuration options ... 310
What's next? ... 310
Changing a DAG's timezone ... 311
Version ... 311
Prerequisites ... 311
Permissions ... 311
Create a plugin to change the timezone in Airflow logs ... 312
Create a `plugins.zip` ... 312
Code sample ... 313
What's next? ... 314
Refreshing an AWS CodeArtifact token at runtime ... 314
Version ... 315
Prerequisites ... 315
Permissions ... 315
Code sample ... 316
What's next? ... 317
Custom plugin with Apache Hive and Hadoop ... 317
Version ... 318
Prerequisites ... 318
Permissions ... 318
Requirements ... 296
Download dependencies ... 319
Custom plugin ... 320

Plugins.zip ... 320
Code sample ... 321
Airflow configuration options ... 321
What's next? ... 321
Custom plugin to patch PythonVirtualenvOperator ... 322
Version ... 322
Prerequisites ... 322
Permissions ... 323
Requirements ... 323
Custom plugin sample code ... 323
Plugins.zip ... 325
Code sample ... 325
Airflow configuration options ... 327
What's next? ... 328
Invoking DAGs with Lambda ... 328
Version ... 328
Prerequisites ... 328
Permissions ... 329
Dependencies ... 329
Code example ... 329
Invoking DAGs in different environments ... 331
Version ... 331
Prerequisites ... 331
Permissions ... 332
Dependencies ... 332
Code example ... 332
Amazon RDS server ... 334
Version ... 334
Prerequisites ... 335
Dependencies ... 288
Apache Airflow v2 connection ... 335
Code sample ... 336
What's next? ... 338
Amazon EMR integration ... 339
Version ... 339
Code sample ... 339

Amazon EKS (eksctl) ... 342
Version ... 342
Prerequisites ... 342
Create a public key for Amazon EC2 ... 343
Create the cluster ... 343
Create a `mwaa` namespace ... 344
Create a role for the `mwaa` namespace ... 344
Create and attach an IAM role for the Amazon EKS cluster ... 345
Create the requirements.txt file ... 349
Create an identity mapping for Amazon EKS ... 349
Create the `kubeconfig` ... 349
Create a DAG ... 350
Add the DAG and `kube_config.yaml` to the Amazon S3 bucket ... 352
Enable and trigger the example ... 352
Using the `ECSOperator` ... 353
Version ... 353
Prerequisites ... 353
Permissions ... 354
Create an Amazon ECS cluster ... 355
Code sample ... 360
Using dbt with Amazon MWAA ... 363
Version ... 363
Prerequisites ... 363
Dependencies ... 364
Upload a dbt project to Amazon S3 ... 365
Use a DAG to verify dbt dependency installation ... 366
Use a DAG to run a dbt project ... 366
AWS blogs and tutorials ... 367
Best practices ... 368
Performance tuning for Apache Airflow ... 368
Adding an Apache Airflow configuration option ... 368
Apache Airflow scheduler ... 369
DAG folders ... 374
DAG files ... 376
Tasks ... 380
Managing Python dependencies ... 385

Testing DAGs using the Amazon MWAA CLI utility ... 385
Installing Python dependencies using PyPi.org Requirements File Format ... 386
Enabling logs on the Amazon MWAA console ... 391
Viewing logs on the CloudWatch Logs console ... 392
Viewing errors in the Apache Airflow UI ... 393
Example `requirements.txt` scenarios ... 393
Monitoring and metrics ... 395
Overview ... 395
Amazon CloudWatch overview ... 396
AWS CloudTrail overview ... 396
Viewing audit logs ... 396
Creating a trail in CloudTrail ... 397
Viewing events with CloudTrail Event History ... 397
Example trail for `CreateEnvironment` ... 397
What's next? ... 399
Viewing Airflow logs ... 399
Pricing ... 399
Before you begin ... 400
Log types ... 400
Enabling Apache Airflow logs ... 400
Viewing Apache Airflow logs ... 401
Example scheduler logs ... 401
What's next? ... 402
Monitoring dashboards and alarms ... 402
Metrics ... 403
Alarm states overview ... 403
Example custom dashboards and alarms ... 403
Deleting metrics and dashboards ... 409
What's next? ... 409
Apache Airflow v2 environment metrics ... 409
Terms ... 410
Dimensions ... 410
Accessing metrics in the CloudWatch console ... 411
Apache Airflow metrics available in CloudWatch ... 412
Choosing which metrics are reported ... 428
What's next? ... 429

Container, queue, and database metrics ... 429
Terms ... 430
Dimensions ... 430
Accessing metrics ... 431
List of metrics ... 432
Security ... 435
Data Protection ... 436
Encryption ... 436
Using customer managed keys ... 438
AWS Identity and Access Management ... 442
Audience ... 443
Authenticating With Identities ... 443
Managing Access Using Policies ... 446
Allowing users to view their own permissions ... 449
Troubleshooting Amazon Managed Workflows for Apache Airflow identity and access ... 450
How Amazon MWAA works with IAM ... 451
Compliance Validation ... 456
Resilience ... 457
Infrastructure Security ... 457
Configuration and Vulnerability Analysis ... 458
Best practices ... 458
Security best practices in Apache Airflow ... 459
Versions ... 461
About Amazon MWAA versions ... 461
Latest version ... 461
Apache Airflow versions ... 461
Apache Airflow components ... 463
Schedulers ... 463
Workers ... 464
Upgrading the Apache Airflow version ... 464
Apache Airflow deprecated versions ... 464
Apache Airflow version support and FAQ ... 465
Frequently asked questions ... 465
Endpoints and quotas ... 467
Service endpoints ... 467
Service quotas ... 467

Increasing quotas ... 468
FAQs ... 469
Supported versions ... 470
Apache Airflow support ... 470
Apache Airflow versions ... 470
Python version ... 470
Pip version ... 472
Use cases ... 472
When should I use AWS Step Functions vs. Amazon MWAA? ... 472
Environment specifications ... 472
How much task storage is available to each environment? ... 472
Default OS ... 473
Custom images ... 473
HIPAA compliance ... 473
Does Amazon MWAA support Spot Instances? ... 473
Custom domain ... 473
SSH access ... 474
Self-referencing rule ... 474
Custom metrics ... 474
Store data ... 474
Worker quota ... 475
Shared Amazon VPCs ... 475
Shared Amazon VPCs ... 475
Metrics ... 475
Worker metrics ... 475
Custom metrics ... 475
DAGs, Operators, Connections, and other questions ... 476
PythonVirtualenvOperator ... 476
How long does it take Amazon MWAA to recognize a new DAG file? ... 476
Why is my DAG file not picked up by Apache Airflow? ... 476
Remove plugins.zip or requirements.txt ... 476
Remove plugins.zip or requirements.txt ... 477
Can I use AWS Database Migration Service (DMS) Operators? ... 477
When I access the Airflow REST API using the AWS credentials, can I increase the throttling limit to more than 10 transactions per second (TPS)? ... 477
Troubleshooting ... 478

Apache Airflow v2 480
Connections 480
Web server 483
Tasks 484
CLI 487
Operators 488
Amazon MWAA Create/Update 489
Updating `requirements.txt` 490
Plugins 491
Create bucket 491
Create environment 492
Update environment 495
Access environment 495
CloudWatch Logs and CloudTrail 496
Logs 496
Document History 502

What Is Amazon Managed Workflows for Apache Airflow?

Use Amazon Managed Workflows for Apache Airflow, a managed orchestration service for Apache Airflow, to setup and operate data pipelines in the cloud at scale. Apache Airflow is an open-source tool used to programmatically author, schedule, and monitor sequences of processes and tasks referred to as *workflows*.

With Amazon MWAA, you can use Apache Airflow and Python to create workflows without having to manage the underlying infrastructure for scalability, availability, and security. Amazon MWAA automatically scales its workflow execution capacity to meet your needs, and integrates with AWS security services to help provide you with fast and secure access to your data.

Content

- Features
- Architecture
- Integration
- Supported versions
- What's next?

Features

Review the following features to learn how Amazon MWAA can simplify the management of your Apache Airflow workflows.

- **Automatic Airflow setup** – Quickly setup Apache Airflow by choosing an Apache Airflow version when you create an Amazon MWAA environment. Amazon MWAA sets up Apache Airflow for you using the same Apache Airflow user interface and open-source code that you can download on the Internet.
- **Automatic scaling** – Automatically scale Apache Airflow *Workers* by setting the minimum and maximum number of *Workers* that run in your environment. Amazon MWAA monitors the *Workers* in your environment and uses its autoscaling component to add *Workers* to meet demand, up to and until it reaches the maximum number of *Workers* you defined.
- **Built-in authentication** – Enable role-based authentication and authorization for your Apache Airflow *Web server* by defining the access control policies in AWS Identity and Access

Management (IAM). The Apache Airflow *Workers* assume these policies for secure access to AWS services.

- **Built-in security** – The Apache Airflow *Workers* and *Schedulers* run in Amazon MWAA's Amazon VPC. Data is also automatically encrypted using AWS Key Management Service, so your environment is secure by default.
- **Public or private access modes** – Access your Apache Airflow *Web server* using a private, or public access mode. The **Public network** access mode uses a VPC endpoint for your Apache Airflow *Web server* that is accessible *over the Internet*. The **Private network** access mode uses a VPC endpoint for your Apache Airflow *Web server* that is accessible *in your VPC*. In both cases, access for your Apache Airflow users is controlled by the access control policy you define in AWS Identity and Access Management (IAM), and AWS SSO.
- **Streamlined upgrades and patches** – Amazon MWAA provides new versions of Apache Airflow periodically. The Amazon MWAA team will update and patch the images for these versions.
- **Workflow monitoring** – View Apache Airflow logs and Apache Airflow metrics in Amazon CloudWatch to identify Apache Airflow task delays or workflow errors without the need for additional third-party tools. Amazon MWAA automatically sends environment metrics—and if enabled—Apache Airflow logs to CloudWatch.
- **AWS integration** – Amazon MWAA supports open-source integrations with Amazon Athena, AWS Batch, Amazon CloudWatch, Amazon DynamoDB, AWS DataSync, Amazon EMR, AWS Fargate, Amazon EKS, Amazon Data Firehose, AWS Glue, AWS Lambda, Amazon Redshift, Amazon SQS, Amazon SNS, Amazon SageMaker AI, and Amazon S3, as well as hundreds of built-in and community-created operators and sensors.
- **Worker fleets** – Amazon MWAA offers support for using containers to scale the worker fleet on demand and reduce scheduler outages using Amazon ECS on AWS Fargate. Operators that invoke tasks on Amazon ECS containers, and Kubernetes operators that create and run pods on a Kubernetes cluster are supported.

Architecture

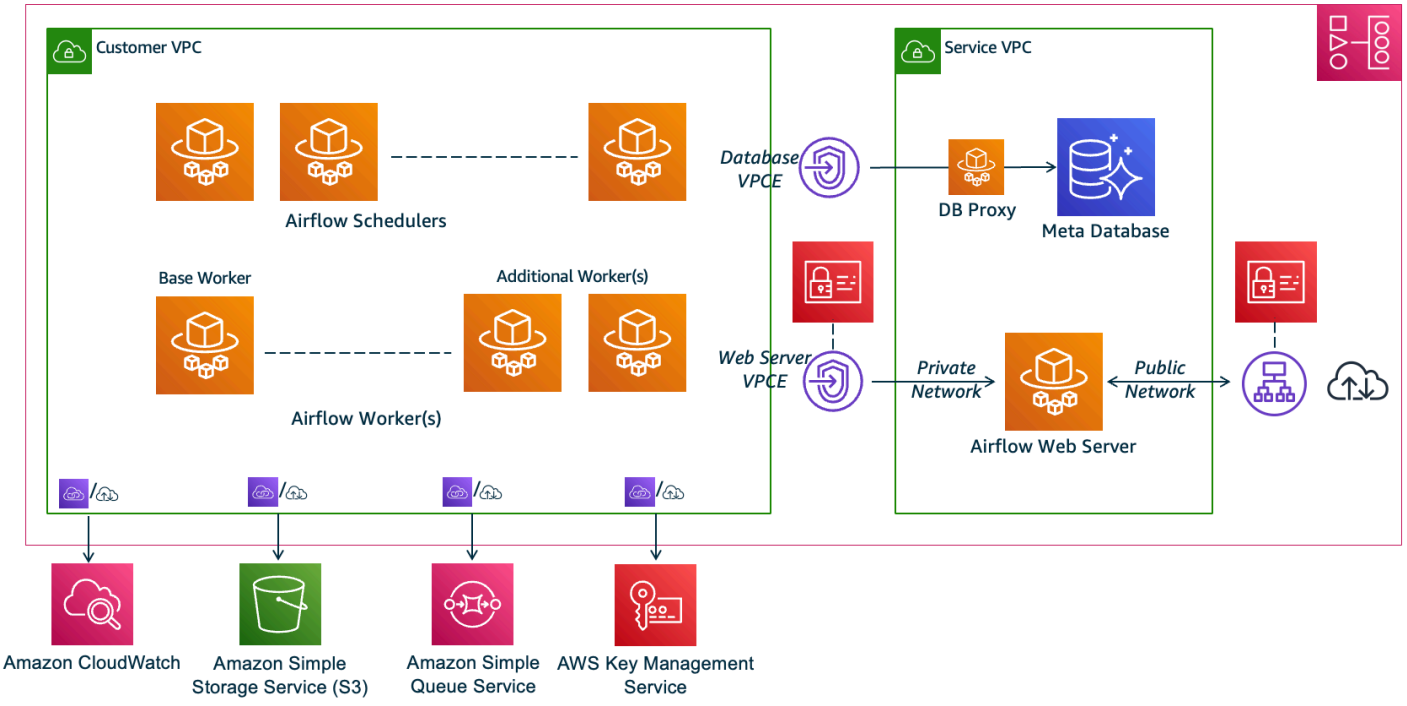
All of the components contained in the outer box (in the image below) appear as a single Amazon MWAA environment in your account. The Apache Airflow *Scheduler* and Workers are AWS Fargate containers that connect to the private subnets in the Amazon VPC for your environment. Each environment has its own Apache Airflow metadatabase managed by AWS that is accessible to the *Scheduler* and Workers Fargate containers via a privately-secured VPC endpoint.

Amazon CloudWatch, Amazon S3, Amazon SQS, and AWS KMS are separate from Amazon MWAA and need to be accessible from the Apache Airflow *Scheduler(s)* and Workers in the Fargate containers.

The Apache Airflow *Web server* can be accessed either *over the Internet* by selecting the **Public network** Apache Airflow access mode, or *within your VPC* by selecting the **Private network** Apache Airflow access mode. In both cases, access for your Apache Airflow users is controlled by the access control policy you define in AWS Identity and Access Management (IAM).

> **Note**
>
> Multiple Apache Airflow *Schedulers* are only available with Apache Airflow v2 and above. Learn more about the Apache Airflow task lifecycle at Concepts in the *Apache Airflow reference guide*.

Amazon MWAA Architecture



Integration

The active and growing Apache Airflow open-source community provides operators (plugins that simplify connections to services) for Apache Airflow to integrate with AWS services. This includes services such as Amazon S3, Amazon Redshift, Amazon EMR, AWS Batch, and Amazon SageMaker AI, as well as services on other cloud platforms.

Using Apache Airflow with Amazon MWAA fully supports integration with AWS services and popular third-party tools such as Apache Hadoop, Presto, Hive, and Spark to perform data processing tasks. Amazon MWAA is committed to maintaining compatibility with the Apache Airflow API, and Amazon MWAA intends to provide reliable integrations to AWS services and make them available to the community, and be involved in community feature development.

For sample code, see Code examples for Amazon Managed Workflows for Apache Airflow.

Supported versions

Amazon MWAA supports multiple versions of Apache Airflow. For more information about the Apache Airflow versions we support and the Apache Airflow components included with each version, see Apache Airflow versions on Amazon Managed Workflows for Apache Airflow.

What's next?

- Get started with a single AWS CloudFormation template that creates an Amazon S3 bucket for your Airflow DAGs and supporting files, an Amazon VPC with public routing, and an Amazon MWAA environment in Quick start tutorial for Amazon Managed Workflows for Apache Airflow.
- Get started incrementally by creating an Amazon S3 bucket for your Airflow DAGs and supporting files, choosing from one of three Amazon VPC networking options, and creating an Amazon MWAA environment in Get started with Amazon Managed Workflows for Apache Airflow.

Quick start tutorial for Amazon Managed Workflows for Apache Airflow

This quick start tutorial uses an AWS CloudFormation template that creates the Amazon VPC infrastructure, an Amazon S3 bucket with a dags folder, and an Amazon Managed Workflows for Apache Airflow environment at the same time.

Topics

- In this tutorial
- Prerequisites
- Step one: Save the AWS CloudFormation template locally
- Step two: Create the stack using the AWS CLI
- Step three: Upload a DAG to Amazon S3 and run in the Apache Airflow UI
- Step four: View logs in CloudWatch Logs
- What's next?

In this tutorial

This tutorial walks you through three AWS Command Line Interface (AWS CLI) commands to upload a DAG to Amazon S3, run the DAG in Apache Airflow, and view logs in CloudWatch. It concludes by walking you through the steps to create an IAM policy for an Apache Airflow development team.

> **Note**
>
> The AWS CloudFormation template on this page creates an Amazon Managed Workflows for Apache Airflow environment for the latest version of Apache Airflow available in AWS CloudFormation. The latest version available is Apache Airflow v2.10.3.

The AWS CloudFormation template on this page creates the following:

- **VPC infrastructure**. The template uses Public routing over the Internet. It uses the Public network access mode for the Apache Airflow *Web server* in `WebserverAccessMode: PUBLIC_ONLY`.

- **Amazon S3 bucket**. The template creates an Amazon S3 bucket with a `dags` folder. It's configured to **Block all public access**, with **Bucket Versioning** enabled, as defined in Create an Amazon S3 bucket for Amazon MWAA.
- **Amazon MWAA environment**. The template creates an Amazon MWAA environment that's associated to the `dags` folder on the Amazon S3 bucket, an execution role with permission to AWS services used by Amazon MWAA, and the default for encryption using an AWS owned key, as defined in Create an Amazon MWAA environment.
- **CloudWatch Logs**. The template enables Apache Airflow logs in CloudWatch at the "INFO" level and up for the *Airflow scheduler log group*, *Airflow web server log group*, *Airflow worker log group*, *Airflow DAG processing log group*, and the *Airflow task log group*, as defined in Viewing Airflow logs in Amazon CloudWatch.

In this tutorial, you'll complete the following tasks:

- **Upload and run a DAG**. Upload Apache Airflow's tutorial DAG for the latest Amazon MWAA supported Apache Airflow version to Amazon S3, and then run in the Apache Airflow UI, as defined in Adding or updating DAGs.
- **View logs**. View the *Airflow web server log group* in CloudWatch Logs, as defined in Viewing Airflow logs in Amazon CloudWatch.
- **Create an access control policy**. Create an access control policy in IAM for your Apache Airflow development team, as defined in Accessing an Amazon MWAA environment.

Prerequisites

The AWS Command Line Interface (AWS CLI) is an open source tool that enables you to interact with AWS services using commands in your command-line shell. To complete the steps on this page, you need the following:

- AWS CLI – Install version 2.
- AWS CLI – Quick configuration with `aws configure`.

Step one: Save the AWS CloudFormation template locally

- Copy the contents of the following template and save locally as `mwaa-public-network.yml`. You can also download the template.

```
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
        Statement:
          - Effect: Allow
            Principal:
              Service:
                - airflow-env.amazonaws.com
                - airflow.amazonaws.com
            Action:
             - "sts:AssumeRole"
```

```
      Path: "/service-role/"

  MwaaExecutionPolicy:
    DependsOn: EnvironmentBucket
    Type: AWS::IAM::ManagedPolicy
    Properties:
      Roles:
        - !Ref MwaaExecutionRole
      PolicyDocument:
        Version: 2012-10-17
        Statement:
          - Effect: Allow
            Action: airflow:PublishMetrics
            Resource:
              - !Sub "arn:aws:airflow:${AWS::Region}:${AWS::AccountId}:environment/
${EnvironmentName}"
          - Effect: Deny
            Action: s3:ListAllMyBuckets
            Resource:
              - !Sub "${EnvironmentBucket.Arn}"
              - !Sub "${EnvironmentBucket.Arn}/*"

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

          - Effect: Allow
            Action:
              - logs:CreateLogStream
              - logs:CreateLogGroup
              - logs:PutLogEvents
              - logs:GetLogEvents
              - logs:GetLogRecord
              - logs:GetLogGroupFields
              - logs:GetQueryResults
```

```
              - logs:DescribeLogGroups
            Resource:
              - !Sub "arn:aws:logs:${AWS::Region}:${AWS::AccountId}:log-
group:airflow-${AWS::StackName}*"
          - Effect: Allow
            Action: cloudwatch:PutMetricData
            Resource: "*"
          - Effect: Allow
            Action:
              - sqs:ChangeMessageVisibility
              - sqs:DeleteMessage
              - sqs:GetQueueAttributes
              - sqs:GetQueueUrl
              - sqs:ReceiveMessage
              - sqs:SendMessage
            Resource:
              - !Sub "arn:aws:sqs:${AWS::Region}:*:airflow-celery-*"
          - Effect: Allow
            Action:
              - kms:Decrypt
              - kms:DescribeKey
              - "kms:GenerateDataKey*"
              - kms:Encrypt
            NotResource: !Sub "arn:aws:kms:*:${AWS::AccountId}:key/*"
            Condition:
              StringLike:
                "kms:ViaService":
                  - !Sub "sqs.${AWS::Region}.amazonaws.com"
Outputs:
  VPC:
    Description: A reference to the created VPC
    Value: !Ref VPC

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

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

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

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

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

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

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

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

Step two: Create the stack using the AWS CLI

1. In your command prompt, navigate to the directory where `mwaa-public-network.yml` is stored. For example:

```
cd mwaaproject
```

2. Use the `aws cloudformation create-stack` command to create the stack using the AWS CLI.

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

> **Note**
>
> It takes over 30 minutes to create the Amazon VPC infrastructure, Amazon S3 bucket, and Amazon MWAA environment.

Step three: Upload a DAG to Amazon S3 and run in the Apache Airflow UI

1. Copy the contents of the `tutorial.py` file for the latest supported Apache Airflow version and save locally as `tutorial.py`.
2. In your command prompt, navigate to the directory where `tutorial.py` is stored. For example:

```
cd mwaaproject
```

3. Use the following command to list all of your Amazon S3 buckets.

```
aws s3 ls
```

4. Use the following command to list the files and folders in the Amazon S3 bucket for your environment.

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

5. Use the following script to upload the `tutorial.py` file to your `dags` folder. Substitute the sample value in *YOUR_S3_BUCKET_NAME*.

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

6. Open the Environments page on the Amazon MWAA console.
7. Choose an environment.
8. Choose **Open Airflow UI**.
9. On the Apache Airflow UI, from the list of available DAGs, choose the **tutorial** DAG.
10. On the DAG details page, choose the **Pause/Unpause DAG** toggle next to your DAG name to unpause the DAG.
11. Choose **Trigger DAG**.

Step four: View logs in CloudWatch Logs

You can view Apache Airflow logs in the CloudWatch console for all of the Apache Airflow logs that were enabled by the AWS CloudFormation stack. The following section shows how to view logs for the *Airflow web server log group*.

1. Open the Environments page on the Amazon MWAA console.
2. Choose an environment.
3. Choose the **Airflow web server log group** on the **Monitoring** pane.
4. Choose the `webserver_console_ip` log in **Log streams**.

What's next?

- Learn more about how to upload DAGs, specify Python dependencies in a `requirements.txt` and custom plugins in a `plugins.zip` in Working with DAGs on Amazon MWAA.
- Learn more about the best practices we recommend to tune the performance of your environment in Performance tuning for Apache Airflow on Amazon MWAA.
- Create a monitoring dashboard for your environment in Monitoring dashboards and alarms on Amazon MWAA.
- Run some of the DAG code samples in Code examples for Amazon Managed Workflows for Apache Airflow.

Get started with Amazon Managed Workflows for Apache Airflow

Amazon Managed Workflows for Apache Airflow uses the Amazon VPC, DAG code and supporting files in your Amazon S3 storage bucket to create an environment. This chapter describes the prerequisites and AWS resources needed to get started with Amazon MWAA.

Topics

- Prerequisites
- About this guide
- Before you begin
- Available regions
- Create an Amazon S3 bucket for Amazon MWAA
- Create the VPC network
- Create an Amazon MWAA environment
- What's next?

Prerequisites

To create an Amazon MWAA environment, you may want to take additional steps to ensure you have permission to the AWS resources you need to create.

- **AWS account** – An AWS account with permission to use Amazon MWAA and the AWS services and resources used by your environment.

About this guide

This section describes the AWS infrastructure and resources you'll create in this guide.

- **Amazon VPC** – The Amazon VPC networking components required by an Amazon MWAA environment. You can configure an existing VPC that meets these requirements (advanced) as seen in About networking on Amazon MWAA, or create the VPC and networking components, as defined in the section called “Create the VPC network”.

- **Amazon S3 bucket** – An Amazon S3 bucket to store your DAGs and associated files, such as `plugins.zip` and `requirements.txt`. Your Amazon S3 bucket must be configured to **Block all public access**, with **Bucket Versioning** enabled, as defined in Create an Amazon S3 bucket for Amazon MWAA.
- **Amazon MWAA environment** – An Amazon MWAA environment configured with the location of your Amazon S3 bucket, the path to your DAG code and any custom plugins or Python dependencies, and your Amazon VPC and its security group, as defined in Create an Amazon MWAA environment.

Before you begin

To create an Amazon MWAA environment, you may want to take additional steps to create and configure other AWS resources before you create your environment.

To create an environment, you need the following:

- **AWS KMS key** – An AWS KMS key for data encryption on your environment. You can choose the default option on the Amazon MWAA console to create an AWS owned key when you create an environment, or specify an existing Customer managed key with permissions to other AWS services used by your environment configured (advanced). To learn more, see Using customer managed keys for encryption.
- **Execution role** – An execution role that allows Amazon MWAA to access AWS resources in your environment. You can choose the default option on the Amazon MWAA console to create an execution role when you create an environment. To learn more, see Amazon MWAA execution role.
- **VPC security group** – A VPC security group that allows Amazon MWAA to access other AWS resources in your VPC network. You can choose the default option on the Amazon MWAA console to create a security group when you create an environment, or provide a security group with the appropriate inbound and outbound rules (advanced). To learn more, see Security in your VPC on Amazon MWAA.

Available regions

Amazon MWAA is available in the following AWS Regions.

- Europe (Stockholm) - eu-north-1

- Europe (Frankfurt) - eu-central-1
- Europe (Zurich) - eu-central-2
- Europe (Ireland) - eu-west-1
- Europe (London) - eu-west-2
- Europe (Paris) - eu-west-3
- Europe (Milan)- eu-south-1
- Europe (Spain) - eu-south-2
- Asia Pacific (Mumbai) - ap-south-1
- Asia Pacific (Hyderabad) - ap-south-2
- Asia Pacific (Singapore) - ap-southeast-1
- Asia Pacific (Sydney) - ap-southeast-2
- Asia Pacific (Jakarta) - ap-southeast-3
- Asia Pacific (Melbourne) - ap-southeast-4
- Asia Pacific (Tokyo) - ap-northeast-1
- Asia Pacific (Seoul) - ap-northeast-2
- Asia Pacific (Osaka) - ap-northeast-3
- Asia Pacific (Hong Kong) - ap-east-1
- US East (N. Virginia) - us-east-1
- US East (Ohio) - us-east-2
- US West (N. California) - us-west-1
- US West (Oregon) - us-west-2
- Canada (Central) - ca-central-1
- Canada West (Calgary) - ca-west-1
- South America (São Paulo) - sa-east-1
- Africa (Cape Town) - af-south-1
- Israel (Tel Aviv) - il-central-1
- Middle East (Bahrain) - me-south-1
- Middle East (UAE) - me-central-1

Create an Amazon S3 bucket for Amazon MWAA

This guide describes the steps to create an Amazon S3 bucket to store your Apache Airflow Directed Acyclic Graphs (DAGs), custom plugins in a `plugins.zip` file, and Python dependencies in a `requirements.txt` file.

Contents

- Before you begin
- Create the bucket
- What's next?

Before you begin

- The Amazon S3 bucket name can't be changed after you create the bucket. To learn more, see Rules for bucket naming in the *Amazon Simple Storage Service User Guide*.
- An Amazon S3 bucket used for an Amazon MWAA environment must be configured to **Block all public access**, with **Bucket Versioning** enabled.
- An Amazon S3 bucket used for an Amazon MWAA environment must be located in the same AWS Region as an Amazon MWAA environment. To view a list of AWS Regions for Amazon MWAA, see Amazon MWAA endpoints and quotas in the *AWS General Reference*.

Create the bucket

This section describes the steps to create the Amazon S3 bucket for your environment.

To create a bucket

1. Sign in to the AWS Management Console and open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. Choose **Create bucket**.
3. In **Bucket name**, enter a DNS-compliant name for your bucket.

   The bucket name must:

   - Be unique across all of Amazon S3.
   - Be between 3 and 63 characters long.

- Not contain uppercase characters.
- Start with a lowercase letter or number.

> **Important**
>
> Avoid including sensitive information, such as account numbers, in the bucket name. The bucket name is visible in the URLs that point to the objects in the bucket.

4. Choose an AWS Region in **Region**. This must be the same AWS Region as your Amazon MWAA environment.
   - We recommend choosing a region close to you to minimize latency and costs and address regulatory requirements.
5. Choose **Block all public access**.
6. Choose **Enable** in **Bucket Versioning**.
7. **Optional** - *Tags*. Add key-value tag pairs to identify your Amazon S3 bucket in **Tags**. For example, `Bucket : Staging`.
8. **Optional** - *Server-side encryption*. You can optionally **Enable** one of the following encryption options on your Amazon S3 bucket.
   a. Choose **Amazon S3 key (SSE-S3)** in **Server-side encryption** to enable server-side encryption for the bucket.
   b. Choose **AWS Key Management Service key (SSE-KMS)** to use an AWS KMS key for encryption on your Amazon S3 bucket:
      i. **AWS managed key (aws/s3)** - If you choose this option, you can either use an AWS owned key managed by Amazon MWAA, or specify a Customer managed key for encryption of your Amazon MWAA environment.
      ii. **Choose from your AWS KMS keys** or **Enter AWS KMS key ARN** - If you choose to specify a Customer managed key in this step, you must specify an AWS KMS key ID or ARN. AWS KMS aliases and multi-region keys are not supported by Amazon MWAA. The AWS KMS key you specify must also be used for encryption on your Amazon MWAA environment.
9. **Optional** - *Advanced settings*. If you want to enable Amazon S3 Object Lock:
   a. Choose **Advanced settings**, **Enable**.

> **Important**
>
> Enabling Object Lock will permanently allow objects in this bucket to be locked. To learn more, see Locking Objects Using Amazon S3 Object Lock in the *Amazon Simple Storage Service User Guide*.

b. Choose the acknowledgement.

10. Choose **Create bucket**.

What's next?

- Learn how to create the required Amazon VPC network for an environment in Create the VPC network.
- Learn how to how to manage access permissions in How do I set ACL bucket permissions?
- Learn how to delete a storage bucket in How do I delete an S3 Bucket?.

Create the VPC network

Amazon Managed Workflows for Apache Airflow requires an Amazon VPC and specific networking components to support an environment. This guide describes the different options to create the Amazon VPC network for an Amazon Managed Workflows for Apache Airflow environment.

> **Note**
>
> Apache Airflow works best in a low-latency network environment. If you are using an existing Amazon VPC which routes traffic to another region or to an on-premise environment, we recommended adding AWS PrivateLink endpoints for Amazon SQS, CloudWatch, Amazon S3, and AWS KMS. For more information about configuring AWS PrivateLink for Amazon MWAA, see Creating an Amazon VPC network without internet access.

Contents

- Prerequisites
- Before you begin

- Options to create the Amazon VPC network
  - Option one: Creating the VPC network on the Amazon MWAA console
  - Option two: Creating an Amazon VPC network with Internet access
  - Option three: Creating an Amazon VPC network without Internet access
- What's next?

Prerequisites

The AWS Command Line Interface (AWS CLI) is an open source tool that enables you to interact with AWS services using commands in your command-line shell. To complete the steps on this page, you need the following:

- AWS CLI – Install version 2.
- AWS CLI – Quick configuration with `aws configure`.

Before you begin

- The VPC network you specify for your environment can't be changed after the environment is created.
- You can use private or public routing for your Amazon VPC and Apache Airflow *Web server*. To view a list of options, see the section called "Example use cases for an Amazon VPC and Apache Airflow access mode".

Options to create the Amazon VPC network

The following section describes the options available to create the Amazon VPC network for an environment.

> **Note**
>
> Amazon MWAA does not support the use of `use1-az3` Availability Zone (AZ) in the US East (N. Virginia) Region. When creating the VPC for Amazon MWAA in the US East (N. Virginia) region, you must explicitly assign the `AvailabilityZone` in the AWS CloudFormation (CFN) template. The assigned availability zone name must not be mapped to `use1-az3`.

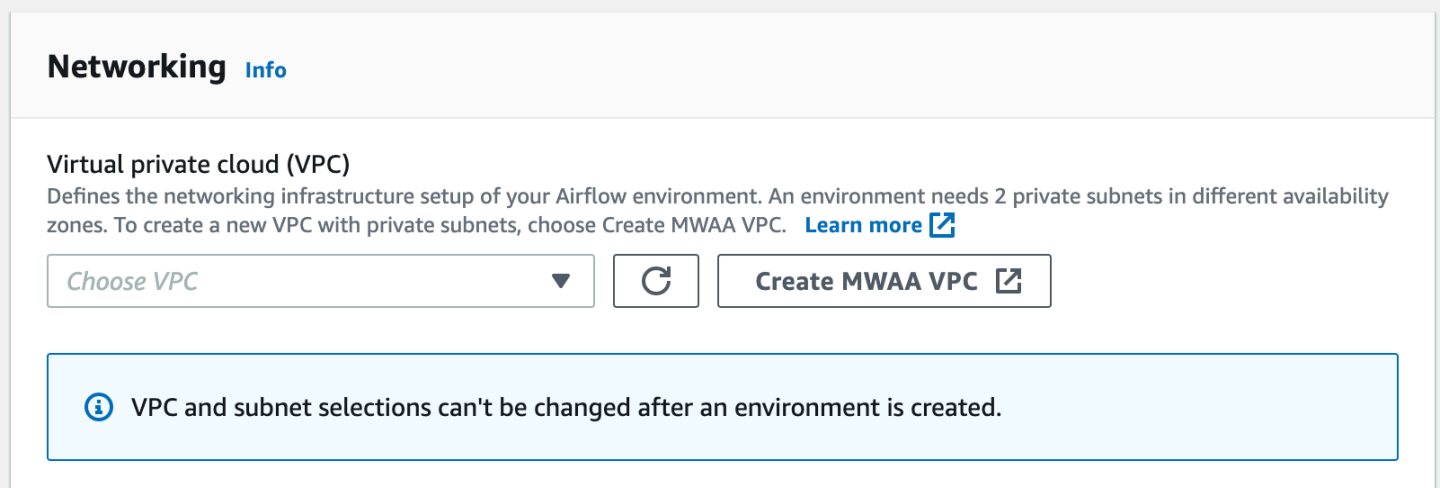
You can retrieve the detailed mapping of AZ names to their corresponding AZ IDs by running the following command:

```
aws ec2 describe-availability-zones --region us-east-1
```

Option one: Creating the VPC network on the Amazon MWAA console

The following section shows how to create an Amazon VPC network on the Amazon MWAA console. This option uses Public routing over the Internet. It can be used for an Apache Airflow *Web server* with the **Private network** or **Public network** access modes.

The following image shows where you can find the **Create MWAA VPC** button on the Amazon MWAA console.



Option two: Creating an Amazon VPC network *with* Internet access

The following AWS CloudFormation template creates an Amazon VPC network *with Internet access* in your default AWS Region. This option uses Public routing over the Internet. This template can be used for an Apache Airflow *Web server* with the **Private network** or **Public network** access modes.

1. Copy the contents of the following template and save locally as `cfn-vpc-public-private.yaml`. You can also download the template.

```
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. In your command prompt, navigate to the directory where `cfn-vpc-public-private.yaml` is stored. For example:

```
cd mwaaproject
```

3. Use the `aws cloudformation create-stack` command to create the stack using the AWS CLI.

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

> **Note**
>
> It takes about 30 minutes to create the Amazon VPC infrastructure.

Option three: Creating an Amazon VPC network *without* Internet access

The following AWS CloudFormation template creates an Amazon VPC network *without Internet access* in your default AWS region.

This option uses Private routing without Internet access. This template can be used for an Apache Airflow *Web server* with the **Private network** access mode only. It creates the required VPC endpoints for the AWS services used by an environment.

1. Copy the contents of the following template and save locally as `cfn-vpc-private.yaml`. You can also download the template.

```
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. In your command prompt, navigate to the directory where `cfn-vpc-private.yml` is stored. For example:

```
cd mwaaproject
```

3. Use the `aws cloudformation create-stack` command to create the stack using the AWS CLI.

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

> **Note**
>
> It takes about 30 minutes to create the Amazon VPC infrastructure.

4. You'll need to create a mechanism to access these VPC endpoints from your computer. To learn more, see Managing access to service-specific Amazon VPC endpoints on Amazon MWAA.

> **Note**
>
> You can further restrict outbound access in the CIDR of your Amazon MWAA security group. For example, you can restrict to itself by adding a self-referencing outbound rule, the prefix list for Amazon S3, and the CIDR of your Amazon VPC.

What's next?

- Learn how to create an Amazon MWAA environment in Create an Amazon MWAA environment.
- Learn how to create a VPN tunnel from your computer to your Amazon VPC with private routing in Tutorial: Configuring private network access using an AWS Client VPN.

Create an Amazon MWAA environment

Amazon Managed Workflows for Apache Airflow sets up Apache Airflow on an environment in your chosen version using the same open-source Apache Airflow and user interface available from Apache. This guide describes the steps to create an Amazon MWAA environment.

Contents

- Before you begin
- Apache Airflow versions
- Create an environment
  - Step one: Specify details
  - Step two: Configure advanced settings
  - Step three: Review and create

Before you begin

- The VPC network you specify for your environment cannot be modified after the environment is created.
- You need an Amazon S3 bucket configured to **Block all public access**, with **Bucket Versioning** enabled.
- You need an AWS account with permissions to use Amazon MWAA, and permission in AWS Identity and Access Management (IAM) to create IAM roles. If you choose the **Private network** access mode for the Apache Airflow *web server*, which limits Apache Airflow access within your Amazon VPC, you'll need permission in IAM to create Amazon VPC endpoints.

Apache Airflow versions

The following Apache Airflow versions are supported on Amazon Managed Workflows for Apache Airflow.

Note

- Beginning with Apache Airflow v2.2.2, Amazon MWAA supports installing Python requirements, provider packages, and custom plugins directly on the Apache Airflow web server.

- Beginning with Apache Airflow v2.7.2, your requirements file must include a `--constraint` statement. If you do not provide a constraint, Amazon MWAA will specify one for you to ensure the packages listed in your requirements are compatible with the version of Apache Airflow you are using.

  For more information on setting up constraints in your requirements file, see Installing Python dependencies.

Apache Airflow version	Apache Airflow guide	Apache Airflow constraints	Python version
2.10.3;	Apache Airflow v2.10.3 reference guide	Apache Airflow v2.10.3 constraints file	Python 3.11
2.10.1;	Apache Airflow v2.10.1 reference guide	Apache Airflow v2.10.1 constraints file	Python 3.11
v2.9.2	Apache Airflow v2.9.2 reference guide	Apache Airflow v2.9.2 constraints file	Python 3.11
v2.8.1	Apache Airflow v2.8.1 reference guide	Apache Airflow v2.8.1 constraints file	Python 3.11
v2.7.2	Apache Airflow v2.7.2 reference guide	Apache Airflow v2.7.2 constraints file	Python 3.11
v2.6.3	Apache Airflow v2.6.3 reference guide	Apache Airflow v2.6.3 constraints file	Python 3.10
v2.5.1	Apache Airflow v2.5.1 reference guide	Apache Airflow v2.5.1 constraints file	Python 3.10
v2.4.3	Apache Airflow v2.4.3 reference guide	Apache Airflow v2.4.3 constraints file	Python 3.10

Apache Airflow version	Apache Airflow guide	Apache Airflow constraints	Python version
v2.2.2	Apache Airflow v2.2.2 reference guide	Apache Airflow v2.2.2 constraints file	Python 3.7
v2.0.2	Apache Airflow v2.0.2 reference guide	Apache Airflow v2.0.2 constraints file	Python 3.7

For more information about migrating your self-managed Apache Airflow deployments, or migrating an existing Amazon MWAA environment, including instructions for backing up your metadata database, see the Amazon MWAA Migration Guide.

Create an environment

The following section describes the steps to create an Amazon MWAA environment.

Step one: Specify details

To specify details for the environment

1. Open the Amazon MWAA console.
2. Use the AWS Region selector to select your region.
3. Choose **Create environment**.
4. On the **Specify details** page, under **Environment details**:
   a. Type a unique name for your environment in **Name**.
   b. Choose the Apache Airflow version in **Airflow version**.

> **Note**
>
> If no value is specified, defaults to the latest Apache Airflow version. The latest version available is Apache Airflow v2.10.3.

5. Under **DAG code in Amazon S3** specify the following:
   a. **S3 Bucket**. Choose **Browse S3** and select your Amazon S3 bucket, or enter the Amazon S3 URI.

b. **DAGs folder**. Choose **Browse S3** and select the `dags` folder in your Amazon S3 bucket, or enter the Amazon S3 URI.

c. **Plugins file - *optional***. Choose **Browse S3** and select the `plugins.zip` file on your Amazon S3 bucket, or enter the Amazon S3 URI.

d. **Requirements file - *optional***. Choose **Browse S3** and select the `requirements.txt` file on your Amazon S3 bucket, or enter the Amazon S3 URI.

e. **Startup script file - *optional***, Choose **Browse S3** and select the script file on your Amazon S3 bucket, or enter the Amazon S3 URI.

6. Choose **Next**.

Step two: Configure advanced settings

To configure advanced settings

1. On the **Configure advanced settings** page, under **Networking**:

   - Choose your Amazon VPC.

     This step populates two of the private subnets in your Amazon VPC.

2. Under **Web server access**, select your preferred Apache Airflow access mode:

   a. **Private network**. This limits access of the Apache Airflow UI to users *within your Amazon VPC* that have been granted access to the IAM policy for your environment. You need permission to create Amazon VPC endpoints for this step.

   > **Note**
   >
   > Choose the **Private network** option if your Apache Airflow UI is only accessed within a corporate network, and you do not require access to public repositories for web server requirements installation. If you choose this access mode option, you need to create a mechanism to access your Apache Airflow *Web server* in your Amazon VPC. For more information, see Accessing the VPC endpoint for your Apache Airflow Web server (private network access).

   b. **Public network**. This allows the Apache Airflow UI to be accessed *over the Internet* by users granted access to the IAM policy for your environment.

3. Under **Security group(s)**, choose the security group used to secure your Amazon VPC:

a. By default, Amazon MWAA creates a security group in your Amazon VPC with specific inbound and outbound rules in **Create new security group**.

b. **Optional**. Deselect the check box in **Create new security group** to select up to 5 security groups.

> **Note**
>
> An existing Amazon VPC security group must be configured with specific inbound and outbound rules to allow network traffic. To learn more, see Security in your VPC on Amazon MWAA.

4. Under **Environment class**, choose an environment class.

   We recommend choosing the smallest size necessary to support your workload. You can change the environment class at any time.

5. For **Maximum worker count**, specify the maximum number of Apache Airflow workers to run in the environment.

   For more information, see Example high performance use case.

6. Specify the **Maximum web server count** and **Minimum web server count** to configure how Amazon MWAA scales the Apache Airflow web servers in your environment.

   For more information about web server automatic scaling, see the section called “Configuring web server auto scaling”.

7. Under **Encryption**, choose a data encryption option:

   a. By default, Amazon MWAA uses an AWS owned key to encrypt your data.

   b. **Optional**. Choose **Customize encryption settings (advanced)** to choose a different AWS KMS key. If you choose to specify a Customer managed key in this step, you must specify an AWS KMS key ID or ARN. AWS KMS aliases and multi-region keys are not supported by Amazon MWAA. If you specified an Amazon S3 key for server-side encryption on your Amazon S3 bucket, you must specify the same key for your Amazon MWAA environment.

> **Note**
>
> You must have permissions to the key to select it on the Amazon MWAA console. You must also grant permissions for Amazon MWAA to use the key by attaching the policy described in Attach key policy.

8. **Recommended**. Under **Monitoring**, choose one or more log categories for **Airflow logging configuration** to send Apache Airflow logs to CloudWatch Logs:

   a. **Airflow task logs**. Choose the type of Apache Airflow task logs to send to CloudWatch Logs in **Log level**.

   b. **Airflow web server logs**. Choose the type of Apache Airflow web server logs to send to CloudWatch Logs in **Log level**.

   c. **Airflow scheduler logs**. Choose the type of Apache Airflow scheduler logs to send to CloudWatch Logs in **Log level**.

   d. **Airflow worker logs**. Choose the type of Apache Airflow worker logs to send to CloudWatch Logs in **Log level**.

   e. **Airflow DAG processing logs**. Choose the type of Apache Airflow DAG processing logs to send to CloudWatch Logs in **Log level**.

9. **Optional**. For **Airflow configuration options**, choose **Add custom configuration option**.

   You can choose from the suggested dropdown list of Apache Airflow configuration options for your Apache Airflow version, or specify custom configuration options. For example, `core.default_task_retries : 3`.

10. **Optional**. Under **Tags**, choose **Add new tag** to associate tags to your environment. For example, `Environment: Staging`.

11. Under **Permissions**, choose an execution role:

    a. By default, Amazon MWAA creates an execution role in **Create a new role**. You must have permission to create IAM roles to use this option.

    b. **Optional**. Choose **Enter role ARN** to enter the Amazon Resource Name (ARN) of an existing execution role.

12. Choose **Next**.

Step three: Review and create

To review an environment summary

- Review the environment summary, choose **Create environment**.

> **Note**
>
> It takes about twenty to thirty minutes to create an environment.

What's next?

- Learn how to create an Amazon S3 bucket in Create an Amazon S3 bucket for Amazon MWAA.

Managing access to an Amazon MWAA environment

Amazon Managed Workflows for Apache Airflow needs to be permitted to use other AWS services and resources used by an environment. You also need to be granted permission to access an Amazon MWAA environment and your Apache Airflow UI in AWS Identity and Access Management (IAM). This section describes the execution role used to grant access to the AWS resources for your environment and how to add permissions, and the AWS account permissions you need to access your Amazon MWAA environment and Apache Airflow UI.

Topics

- Accessing an Amazon MWAA environment
- Service-linked role for Amazon MWAA
- Amazon MWAA execution role
- Cross-service confused deputy prevention
- Apache Airflow access modes

Accessing an Amazon MWAA environment

To use Amazon Managed Workflows for Apache Airflow, you must use an account, and IAM entities with the necessary permissions. This topic describes the access policies you can attach to your Apache Airflow development team and Apache Airflow users for your Amazon Managed Workflows for Apache Airflow environment.

We recommend using temporary credentials and configuring federated identities with groups and roles, to access your Amazon MWAA resources. As a best practice, avoid attaching policies directly to your IAM users, and instead define groups or roles to provide temporary access to AWS resources.

An IAM role is an IAM identity that you can create in your account that has specific permissions. An IAM role is similar to an IAM user in that it is an AWS identity with permissions policies that determine what the identity can and cannot do in AWS. However, instead of being uniquely associated with one person, a role is intended to be assumable by anyone who needs it. Also, a role does not have standard long-term credentials such as a password or access keys associated with it. Instead, when you assume a role, it provides you with temporary security credentials for your role session.

To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see Create a role for a third-party identity provider (federation) in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see Permission sets in the *AWS IAM Identity Center User Guide*.

You can use an IAM role in your account to grant another AWS account permissions to access your account's resources. For an example, see Tutorial: Delegate access across AWS accounts using IAM roles in the *IAM User Guide*.

Sections

- How it works
- Full console access policy: AmazonMWAAFullConsoleAccess
- Full API and console access policy: AmazonMWAAFullApiAccess
- Read-only console access policy: AmazonMWAAReadOnlyAccess
- Apache Airflow UI access policy: AmazonMWAAWebServerAccess
- Apache Airflow Rest API access policy: AmazonMWAARestAPIAccess
- Apache Airflow CLI policy: AmazonMWAAAirflowCliAccess
- Creating a JSON policy
- Example use case to attach policies to a developer group
- What's next?

How it works

The resources and services used in an Amazon MWAA environment are not accessible to all AWS Identity and Access Management (IAM) entities. You must create a policy that grants Apache Airflow users permission to access these resources. For example, you need to grant access to your Apache Airflow development team.

Amazon MWAA uses these policies to validate whether a user has the permissions needed to perform an action on the AWS console or via the APIs used by an environment.

You can use the JSON policies in this topic to create a policy for your Apache Airflow users in IAM, and then attach the policy to a user, group, or role in IAM.

- AmazonMWAAFullConsoleAccess – Use this policy to grant permission to configure an environment on the Amazon MWAA console.
- AmazonMWAAFullApiAccess – Use this policy to grant access to all Amazon MWAA APIs used to manage an environment.
- AmazonMWAAReadOnlyAccess – Use this policy to grant access to to view the resources used by an environment on the Amazon MWAA console.
- AmazonMWAAWebServerAccess – Use this policy to grant access to the Apache Airflow web server.
- AmazonMWAAAirflowCliAccess – Use this policy to grant access to run Apache Airflow CLI commands.

To provide access, add permissions to your users, groups, or roles:

- Users and groups in AWS IAM Identity Center:

  Create a permission set. Follow the instructions in Create a permission set in the *AWS IAM Identity Center User Guide*.
- Users managed in IAM through an identity provider:

  Create a role for identity federation. Follow the instructions in Create a role for a third-party identity provider (federation) in the *IAM User Guide*.
- IAM users:
  - Create a role that your user can assume. Follow the instructions in Create a role for an IAM user in the *IAM User Guide*.
  - (Not recommended) Attach a policy directly to a user or add a user to a user group. Follow the instructions in Adding permissions to a user (console) in the *IAM User Guide*.

Full console access policy: AmazonMWAAFullConsoleAccess

A user may need access to the `AmazonMWAAFullConsoleAccess` permissions policy if they need to configure an environment on the Amazon MWAA console.

> **Note**
>
> Your full console access policy must include permissions to perform `iam:PassRole`. This allows the user to pass service-linked roles, and execution roles, to Amazon MWAA. Amazon

MWAA assumes each role in order to call other AWS services on your behalf. The following example uses the `iam:PassedToService` condition key to specify the Amazon MWAA service principal (`airflow.amazonaws.com`) as the service to which a role can be passed. For more information about `iam:PassRole`, see Granting a user permissions to pass a role to an AWS service in the *IAM User Guide*.

Use the following policy if you want to create, and manage, your Amazon MWAA environments using an AWS owned key for encryption at-rest.

Using an AWS owned key

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

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

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

Use the following policy if you want to create, and manage, your Amazon MWAA environments using a customer managed key for encryption at-rest. To use a customer managed key, the IAM principal must have permission to access AWS KMS resources using the key stored in your account.

Using a customer managed key

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":"airflow:*",
         "Resource":"*"
      },
      {
         "Effect":"Allow",
         "Action":[
            "iam:PassRole"
         ],
         "Resource":"*",
         "Condition":{
            "StringLike":{
               "iam:PassedToService":"airflow.amazonaws.com"
            }
         }
      },
      {
         "Effect":"Allow",
         "Action":[
            "iam:ListRoles"
         ],
         "Resource":"*"
      },
      {
         "Effect":"Allow",
         "Action":[
            "iam:CreatePolicy"
         ],
         "Resource":"arn:aws:iam::YOUR_ACCOUNT_ID:policy/service-role/MWAA-Execution-
Policy*"
      },
            {
         "Effect":"Allow",
```

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

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

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

Full API and console access policy: AmazonMWAAFullApiAccess

A user may need access to the `AmazonMWAAFullApiAccess` permissions policy if they need access to all Amazon MWAA APIs used to manage an environment. It does not grant permissions to access the Apache Airflow UI.

> **Note**
>
> A full API access policy must include permissions to perform `iam:PassRole`. This allows the user to pass service-linked roles, and execution roles, to Amazon MWAA. Amazon MWAA assumes each role in order to call other AWS services on your behalf. The following example uses the `iam:PassedToService` condition key to specify the Amazon MWAA service principal (`airflow.amazonaws.com`) as the service to which a role can be passed. For more information about `iam:PassRole`, see Granting a user permissions to pass a role to an AWS service in the *IAM User Guide*.

Use the following policy if you want to create, and manage, your Amazon MWAA environments using an AWS owned key for encryption at-rest.

Using an AWS owned key

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

Use the following policy if you want to create, and manage, your Amazon MWAA environments using a customer managed key for encryption at-rest. To use a customer managed key, the IAM principal must have permission to access AWS KMS resources using the key stored in your account.

Using a customer managed key

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

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

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

Read-only console access policy: AmazonMWAAReadOnlyAccess

A user may need access to the `AmazonMWAAReadOnlyAccess` permissions policy if they need to view the resources used by an environment on the Amazon MWAA console environment details page. It doesn't allow a user to create new environments, edit existing environments, or allow a user to view the Apache Airflow UI.

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

Apache Airflow UI access policy: AmazonMWAAWebServerAccess

A user may need access to the `AmazonMWAAWebServerAccess` permissions policy if they need to access the Apache Airflow UI. It does not allow the user to view environments on the Amazon MWAA console or use the Amazon MWAA APIs to perform any actions. Specify the `Admin`, `Op`, `User`, `Viewer` or the `Public` role in `{airflow-role}` to customize the level of access for the

user of the web token. For more information, see Default Roles in the *Apache Airflow reference guide*.

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

Note

- Amazon MWAA provides IAM integration with the five default Apache Airflow role-based access control (RBAC) roles. For more information on working with custom Apache Airflow roles, see the section called "Tutorial: Restricting users to a subset of DAGs".
- The `Resource` field in this policy could be used to specify the Apache Airflow role-based access control roles for the Amazon MWAA environment. However, it does not support the Amazon MWAA environment ARN (Amazon Resource Name) in the `Resource` field of the policy.

Apache Airflow Rest API access policy: AmazonMWAARestAPIAccess

To access the Apache Airflow REST API, you must grant the `airflow:InvokeRestApi` permission in your IAM policy. In the following policy sample, specify the `Admin`, `Op`, `User`, `Viewer` or the `Public` role in `{airflow-role}` to customize the level of user access. For more information, see Default Roles in the *Apache Airflow reference guide*.

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

```
  "Statement": [
    {
      "Sid": "AllowMwaaRestApiAccess",
      "Effect": "Allow",
      "Action": "airflow:InvokeRestApi",
      "Resource": [
        "arn:aws:airflow:{your-region}:YOUR_ACCOUNT_ID:role/{your-environment-name}/
{airflow-role}"
      ]
    }
  ]
}
```

Note

- While configuring a private web server, the `InvokeRestApi` action cannot be invoked from outside of a Virtual Private Cloud (VPC). You can use the `aws:SourceVpc` key to apply more granular access control for this operation. For more information, see aws:SourceVpc
- The `Resource` field in this policy could be used to specify the Apache Airflow role-based access control roles for the Amazon MWAA environment. However, it does not support the Amazon MWAA environment ARN (Amazon Resource Name) in the `Resource` field of the policy.

Apache Airflow CLI policy: AmazonMWAAAirflowCliAccess

A user may need access to the `AmazonMWAAAirflowCliAccess` permissions policy if they need to run Apache Airflow CLI commands (such as `trigger_dag`). It does not allow the user to view environments on the Amazon MWAA console or use the Amazon MWAA APIs to perform any actions.

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

```
            ],
            "Resource": "arn:aws:airflow:${Region}:${Account}:environment/
${EnvironmentName}"
        }
    ]
}
```

Creating a JSON policy

You can create the JSON policy, and attach the policy to your user, role, or group on the IAM console. The following steps describe how to create a JSON policy in IAM.

To create the JSON policy

1. Open the Policies page on the IAM console.
2. Choose **Create policy**.
3. Choose the **JSON** tab.
4. Add your JSON policy.
5. Choose **Review policy**.
6. Enter a value in the text field for **Name** and **Description** (optional).

   For example, you could name the policy `AmazonMWAAReadOnlyAccess`.
7. Choose **Create policy**.

Example use case to attach policies to a developer group

Let's say you're using a group in IAM named `AirflowDevelopmentGroup` to apply permissions to all of the developers on your Apache Airflow development team. These users need access to the `AmazonMWAAFullConsoleAccess`, `AmazonMWAAAirflowCliAccess`, and `AmazonMWAAWebServerAccess` permission policies. This section describes how to create a group in IAM, create and attach these policies, and associate the group to an IAM user. The steps assume you're using an AWS owned key.

To create the AmazonMWAAFullConsoleAccess policy

1. Download the AmazonMWAAFullConsoleAccess access policy.
2. Open the Policies page on the IAM console.

3. Choose **Create policy**.
4. Choose the **JSON** tab.
5. Paste the JSON policy for `AmazonMWAAFullConsoleAccess`.
6. Substitute the following values:
    a. *`{your-account-id}`* – Your AWS account ID (such as `0123456789`)
    b. *`{your-kms-id}`* – The unique identifer for a customer managed key, applicable only if you use a customer managed key for encryption at-rest.
7. Choose the **Review policy**.
8. Type `AmazonMWAAFullConsoleAccess` in **Name**.
9. Choose **Create policy**.

To create the AmazonMWAAWebServerAccess policy

1. Download the AmazonMWAAWebServerAccess access policy.
2. Open the Policies page on the IAM console.
3. Choose **Create policy**.
4. Choose the **JSON** tab.
5. Paste the JSON policy for `AmazonMWAAWebServerAccess`.
6. Substitute the following values:
    a. *`{your-region}`* – the region of your Amazon MWAA environment (such as `us-east-1`)
    b. *`{your-account-id}`* – your AWS account ID (such as `0123456789`)
    c. *`{your-environment-name}`* – your Amazon MWAA environment name (such as `MyAirflowEnvironment`)
    d. *`{airflow-role}`* – the `Admin` Apache Airflow Default Role
7. Choose **Review policy**.
8. Type `AmazonMWAAWebServerAccess` in **Name**.
9. Choose **Create policy**.

To create the AmazonMWAAAirflowCliAccess policy

1. Download the AmazonMWAAAirflowCliAccess access policy.

2. Open the Policies page on the IAM console.
3. Choose **Create policy**.
4. Choose the **JSON** tab.
5. Paste the JSON policy for `AmazonMWAAAirflowCliAccess`.
6. Choose the **Review policy**.
7. Type `AmazonMWAAAirflowCliAccess` in **Name**.
8. Choose **Create policy**.

To create the group

1. Open the Groups page on the IAM console.
2. Type a name of `AirflowDevelopmentGroup`.
3. Choose **Next Step**.
4. Type `AmazonMWAA` to filter results in **Filter**.
5. Select the three policies you created.
6. Choose **Next Step**.
7. Choose **Create Group**.

To associate to a user

1. Open the Users page on the IAM console.
2. Choose a user.
3. Choose **Groups**.
4. Choose **Add user to groups**.
5. Select the **AirflowDevelopmentGroup**.
6. Choose **Add to Groups**.

What's next?

- Learn how to generate a token to access the Apache Airflow UI in Accessing Apache Airflow.
- Learn more about creating IAM policies in Creating IAM policies.

Service-linked role for Amazon MWAA

Amazon Managed Workflows for Apache Airflow uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to Amazon MWAA. Service-linked roles are predefined by Amazon MWAA and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up Amazon MWAA easier because you don't have to manually add the necessary permissions. Amazon MWAA defines the permissions of its service-linked roles, and unless defined otherwise, only Amazon MWAA can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete a service-linked role only after first deleting their related resources. This protects your Amazon MWAA resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see AWS Services That Work with IAM and look for the services that have **Yes** in the **Service-linked roles** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions for Amazon MWAA

Amazon MWAA uses the service-linked role named `AWSServiceRoleForAmazonMWAA` – The service-linked role created in your account grants Amazon MWAA access to the following AWS services:

- Amazon CloudWatch Logs (CloudWatch Logs) – To create log groups for Apache Airflow logs.
- Amazon CloudWatch (CloudWatch) – To publish metrics related to your environment and its underlying components to your account.
- Amazon Elastic Compute Cloud (Amazon EC2) – To create the following resources:
  - An Amazon VPC endpoint in your VPC for an AWS-managed Amazon Aurora PostgreSQL database cluster to be used by the Apache Airflow *Scheduler* and *Worker*.
  - An additional Amazon VPC endpoint to enable network access to the *Web server* if you choose the private network option for your Apache Airflow *Web server*.
  - Elastic Network Interfaces (ENIs) in your Amazon VPC to enable network access to AWS resources hosted in your Amazon VPC.

The following trust policy allows the service principal to assume the service-linked role. The service principal for Amazon MWAA is `airflow.amazonaws.com` as demonstrated by the policy.

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

The role permissions policy named `AmazonMWAAServiceRolePolicy` allows Amazon MWAA to complete the following actions on the specified resources:

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

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see Service-linked role permissions in the *IAM User Guide*.

Creating a service-linked role for Amazon MWAA

You don't need to manually create a service-linked role. When you create a new Amazon MWAA environment using the AWS Management Console, the AWS CLI, or the AWS API, Amazon MWAA creates the service-linked role for you.

If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account. When you create another environment, Amazon MWAA creates the service-linked role for you again.

Editing a service-linked role for Amazon MWAA

Amazon MWAA does not allow you to edit the AWSServiceRoleForAmazonMWAA service-linked role. After you create a service-linked role, you cannot change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see Editing a service-linked role in the *IAM User Guide*.

Deleting a service-linked role for Amazon MWAA

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don't have an unused entity that is not actively monitored or maintained.

When you delete an Amazon MWAA environment, Amazon MWAA deletes all the associated resources it uses as a part of the service. However, you must wait before Amazon MWAA completes deleting your environment, before attempting to delete the service-linked role. If you delete the service-linked role before Amazon MWAA deletes the environment, Amazon MWAA might be unable to delete all of the environment's associated resources.

To manually delete the service-linked role using IAM

Use the IAM console, the AWS CLI, or the AWS API to delete the AWSServiceRoleForAmazonMWAA service-linked role. For more information, see Deleting a service-linked role in the *IAM User Guide*.

Supported regions for Amazon MWAA service-linked roles

Amazon MWAA supports using service-linked roles in all of the regions where the service is available. For more information, see Amazon Managed Workflows for Apache Airflow endpoints and quotas.

Policy updates

Change	Description	Date
Amazon MWAA update its service-linked role permission policy	`AmazonMWAAServiceRolePolicy` – Amazon MWAA updates the permission policy for its service-linked role to grant Amazon	November 18, 2022

Change	Description	Date
	MWAA permission to publish additional metrics related to the service's underlyin g resources to customer accounts. These new metrics are published under the `AWS/MWAA`	
Amazon MWAA started tracking changes	Amazon MWAA started tracking changes for its AWS managed service-linked role permission policy.	November 18, 2022

Amazon MWAA execution role

An execution role is an AWS Identity and Access Management (IAM) role with a permissions policy that grants Amazon Managed Workflows for Apache Airflow permission to invoke the resources of other AWS services on your behalf. This can include resources such as your Amazon S3 bucket, AWS owned key, and CloudWatch Logs. Amazon MWAA environments need one execution role per environment. This topic describes how to use and configure the execution role for your environment to allow Amazon MWAA to access other AWS resources used by your environment.

Contents

- Execution role overview
  - Permissions attached by default
  - How to add permission to use other AWS services
  - How to associate a new execution role
- Create a new role
- View and update an execution role policy
  - Attach a JSON policy to use other AWS services
- Grant access to Amazon S3 bucket with account-level public access block
- Use Apache Airflow connections
- Sample JSON policies for an execution role

- Sample policy for a customer managed key
- Sample policy for an AWS owned key
- What's next?

Execution role overview

Permission for Amazon MWAA to use other AWS services used by your environment are obtained from the execution role. An Amazon MWAA execution role needs permission to the following AWS services used by an environment:

- Amazon CloudWatch (CloudWatch) – to send Apache Airflow metrics and logs.
- Amazon Simple Storage Service (Amazon S3) – to parse your environment's DAG code and supporting files (such as a `requirements.txt`).
- Amazon Simple Queue Service (Amazon SQS) – to queue your environment's Apache Airflow tasks in an Amazon SQS queue owned by Amazon MWAA.
- AWS Key Management Service (AWS KMS) – for your environment's data encryption (using either an AWS owned key or your Customer managed key).

> **Note**
>
> If you have elected for Amazon MWAA to use an AWS owned KMS key to encrypt your data, then you must define permissions in a policy attached to your Amazon MWAA execution role that grant access to arbitrary KMS keys stored outside of your account via Amazon SQS. The following two conditions are required in order for your environment's execution role to access arbitrary KMS keys:
>
> - A KMS key in a third-party account needs to allow this cross account access via its resource policy.
> - Your DAG code needs to access an Amazon SQS queue that starts with `airflow-celery-` in the third-party account and uses the same KMS key for encryption.
>
> In order to mitigate the risks associated with cross-account access to resources, we recommend reviewing the code placed in your DAGs to ensure that your workflows are not accessing arbitrary Amazon SQS queues outside your account. Furthermore, you can use a customer managed KMS key stored in your own account to manage encryption on Amazon MWAA. This limits your environment's execution role to access only the KMS key in your account.

Keep in mind that after you choose an encryption option, you cannot change your selection for an existing environment.

An execution role also needs permission to the following IAM actions:

- `airflow:PublishMetrics` – to allow Amazon MWAA to monitor the health of an environment.

Permissions attached by default

You can use the default options on the Amazon MWAA console to create an execution role and an AWS owned key, then use the steps on this page to add permission policies to your execution role.

- When you choose the **Create new role** option on the console, Amazon MWAA attaches the minimal permissions needed by an environment to your execution role.
- In some cases, Amazon MWAA attaches the maximum permissions. For example, we recommend choosing the option on the Amazon MWAA console to create an execution role when you create an environment. Amazon MWAA adds the permissions policies for all CloudWatch Logs groups automatically by using the regex pattern in the execution role as `"arn:aws:logs:your-region:your-account-id:log-group:airflow-your-environment-name-*"`.

How to add permission to use other AWS services

Amazon MWAA can't add or edit permission policies to an existing execution role after an environment is created. You must update your execution role with additional permission policies needed by your environment. For example, if your DAG requires access to AWS Glue, Amazon MWAA can't automatically detect these permissions are required by your environment, or add the permissions to your execution role.

You can add permissions to an execution role in two ways:

- By modifying the JSON policy for your execution role inline. You can use the sample JSON policy documents on this page to either add to or replace the JSON policy of your execution role on the IAM console.

- By creating a JSON policy for an AWS service and attaching it to your execution role. You can use the steps on this page to associate a new JSON policy document for an AWS service to your execution role on the IAM console.

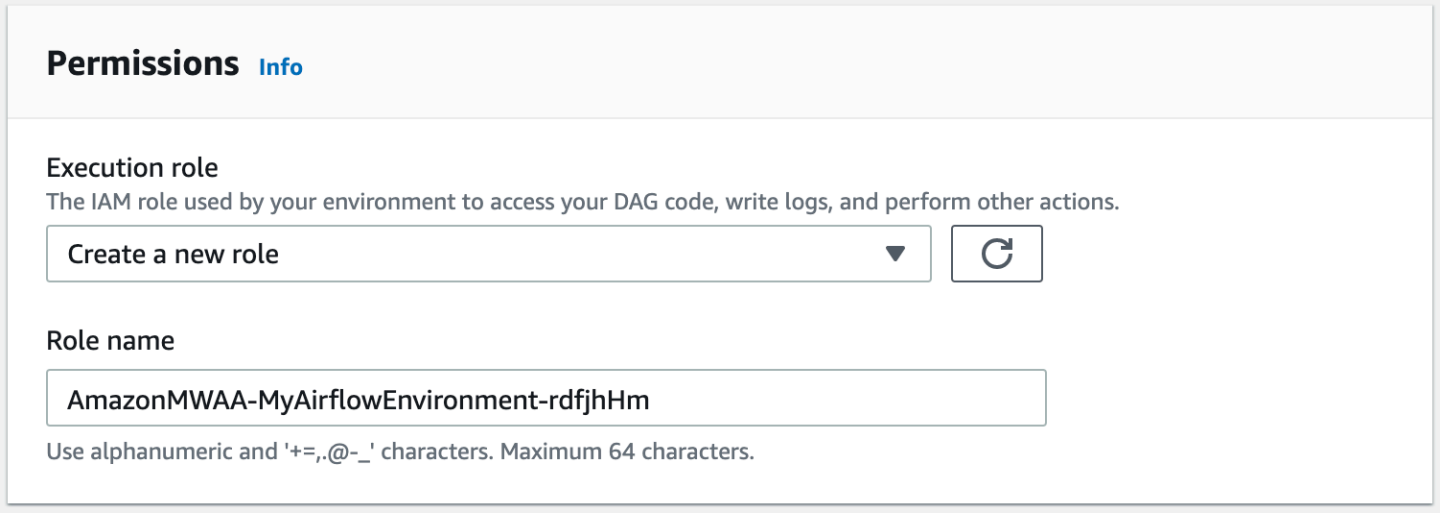
Assuming the execution role is already associated to your environment, Amazon MWAA can start using the added permission policies immediately. This also means if you remove any required permissions from an execution role, your DAGs may fail.

How to associate a new execution role

You can change the execution role for your environment at any time. If a new execution role is not already associated with your environment, use the steps on this page to create a new execution role policy, and associate the role to your environment.

Create a new role

By default, Amazon MWAA creates an AWS owned key for data encryption and an execution role on your behalf. You can choose the default options on the Amazon MWAA console when you create an environment. The following image shows the default option to create an execution role for an environment.



View and update an execution role policy

You can view the execution role for your environment on the Amazon MWAA console, and update the JSON policy for the role on the IAM console.

To update an execution role policy

1. Open the Environments page on the Amazon MWAA console.
2. Choose an environment.
3. Choose the execution role on the **Permissions** pane to open the permissions page in IAM.
4. Choose the execution role name to open the permissions policy.
5. Choose **Edit policy**.
6. Choose the **JSON** tab.
7. Update your JSON policy.
8. Choose **Review policy**.
9. Choose **Save changes**.

Attach a JSON policy to use other AWS services

You can create a JSON policy for an AWS service and attach it to your execution role. For example, you can attach the following JSON policy to grant read-only access to all resources in AWS Secrets Manager.

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

To attach a policy to your execution role

1. Open the Environments page on the Amazon MWAA console.
2. Choose an environment.
3. Choose your execution role on the **Permissions** pane.
4. Choose **Attach policies**.
5. Choose **Create policy**.
6. Choose **JSON**.
7. Paste the JSON policy.
8. Choose **Next: Tags**, **Next: Review**.
9. Enter a descriptive name (such as `SecretsManagerReadPolicy`) and a description for the policy.
10. Choose **Create policy**.

Grant access to Amazon S3 bucket with account-level public access block

You might want to block access to all buckets in your account by using the `PutPublicAccessBlock` Amazon S3 operation. When you block access to all buckets in your account, your environment execution role must include the `s3:GetAccountPublicAccessBlock` action in a permission policy.

The following example demonstrates the policy you must attach to your execution role when blocking access to all Amazon S3 buckets in your account.

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

For more information about restricting access to your Amazon S3 buckets, see Blocking public access to your Amazon S3 storage in the *Amazon Simple Storage Service User Guide*.

Use Apache Airflow connections

You can also create an Apache Airflow connection and specify your execution role and its ARN in your Apache Airflow connection object. To learn more, see Managing connections to Apache Airflow.

Sample JSON policies for an execution role

The sample permission policies in this section show two policies you can use to replace the permissions policy used for your existing execution role, or to create a new execution role and use for your environment. These policies contain Resource ARN placeholders for Apache Airflow log groups, an Amazon S3 bucket, and an Amazon MWAA environment.

We recommend copying the example policy, replacing the sample ARNs or placeholders, then using the JSON policy to create or update an execution role. For example, replacing `{your-region}` with `us-east-1`.

Sample policy for a customer managed key

The following example shows an execution role policy you can use for an Customer managed key.

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

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

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

Next, you need to allow Amazon MWAA to assume this role in order to perform actions on your behalf. This can be done by adding `"airflow.amazonaws.com"` and `"airflow-env.amazonaws.com"` service principals to the list of trusted entities for this execution role using the IAM console, or by placing these service principals in the assume role policy document for this execution role via the IAM create-role command using the AWS CLI. A sample assume role policy document can be found below:

```
{
```

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

Then attach the following JSON policy to your Customer managed key. This policy uses the `kms:EncryptionContext` condition key prefix to permit access to your Apache Airflow logs group in CloudWatch Logs.

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

Sample policy for an AWS owned key

The following example shows an execution role policy you can use for an AWS owned key.

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

```
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetAccountPublicAccessBlock"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "cloudwatch:PutMetricData",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "sqs:ChangeMessageVisibility",
                "sqs:DeleteMessage",
                "sqs:GetQueueAttributes",
                "sqs:GetQueueUrl",
                "sqs:ReceiveMessage",
                "sqs:SendMessage"
            ],
            "Resource": "arn:aws:sqs:{your-region}:*:airflow-celery-*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey",
                "kms:GenerateDataKey*",
                "kms:Encrypt"
```

```
            ],
            "NotResource": "arn:aws:kms:*:{your-account-id}:key/*",
            "Condition": {
                "StringLike": {
                    "kms:ViaService": [
                        "sqs.{your-region}.amazonaws.com"
                    ]
                }
            }
        }
    ]
}
```

What's next?

- Learn about the required permissions you and your Apache Airflow users need to access your environment in Accessing an Amazon MWAA environment.
- Learn about Using customer managed keys for encryption.
- Explore more Customer managed policy examples.

Cross-service confused deputy prevention

The confused deputy problem is a security issue where an entity that doesn't have permission to perform an action can coerce a more-privileged entity to perform the action. In AWS, cross-service impersonation can result in the confused deputy problem. Cross-service impersonation can occur when one service (the *calling service*) calls another service (the *called service*). The calling service can be manipulated to use its permissions to act on another customer's resources in a way it should not otherwise have permission to access. To prevent this, AWS provides tools that help you protect your data for all services with service principals that have been given access to resources in your account.

We recommend using the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in your environment' execution role to limit the permissions that Amazon MWAA gives another service to access the resource. Use `aws:SourceArn` if you want only one resource to be associated with the cross-service access. Use `aws:SourceAccount` if you want to allow any resource in that account to be associated with the cross-service use.

The most effective way to protect against the confused deputy problem is to use the `aws:SourceArn` global condition context key with the full ARN of the resource. If you don't know

the full ARN of the resource or if you are specifying multiple resources, use the `aws:SourceArn` global context condition key with wildcard characters (*) for the unknown portions of the ARN. For example, `arn:aws:airflow:*:123456789012:environment/*`.

The value of `aws:SourceArn` must be your Amazon MWAA environment ARN, for which you are creating an execution role.

The following example shows how you can use the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in your environment's execution role trust policy to prevent the confused deputy problem. You can use the following trust policy when you create a new execution role.

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

Apache Airflow access modes

The Amazon Managed Workflows for Apache Airflow console contains built-in options to configure private or public routing to the Apache Airflow *web server* on your environment. This guide describes the access modes available for the Apache Airflow *Web server* on your Amazon Managed Workflows for Apache Airflow environment, and the additional resources you'll need to configure in your Amazon VPC if you choose the private network option.

Contents

- Apache Airflow access modes
  - Public network
  - Private network
- Access modes overview
  - Public network access mode
  - Private network access mode
- Setup for private and public access modes
  - Setup for public network
  - Setup for private network
- Accessing the VPC endpoint for your Apache Airflow Web server (private network access)

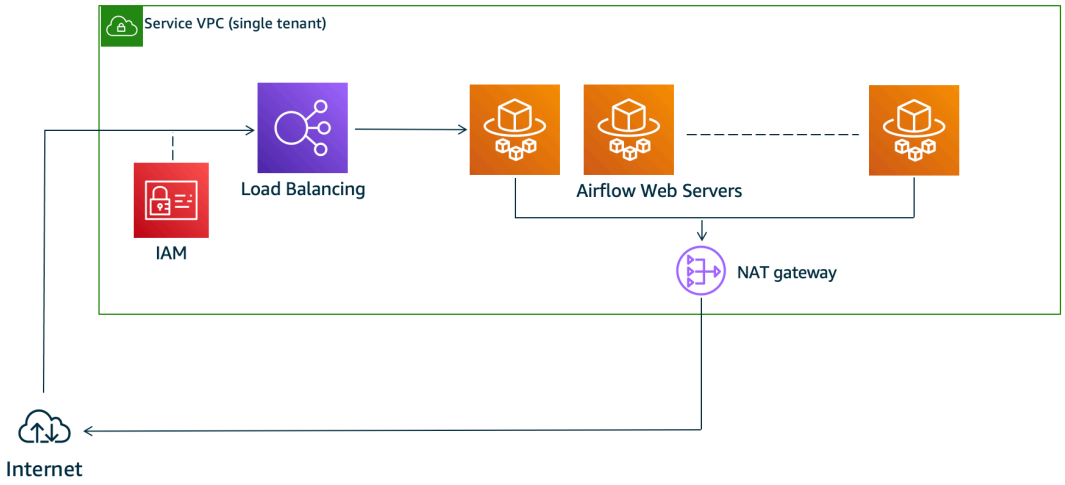
Apache Airflow access modes

You can choose private or public routing for your Apache Airflow *Web server*. To enable private routing, choose **Private network**. This limits user access to an Apache Airflow *Web server* to within an Amazon VPC. To enable public routing, choose **Public network**. This allows users to access the Apache Airflow *Web server* over the Internet.

Public network

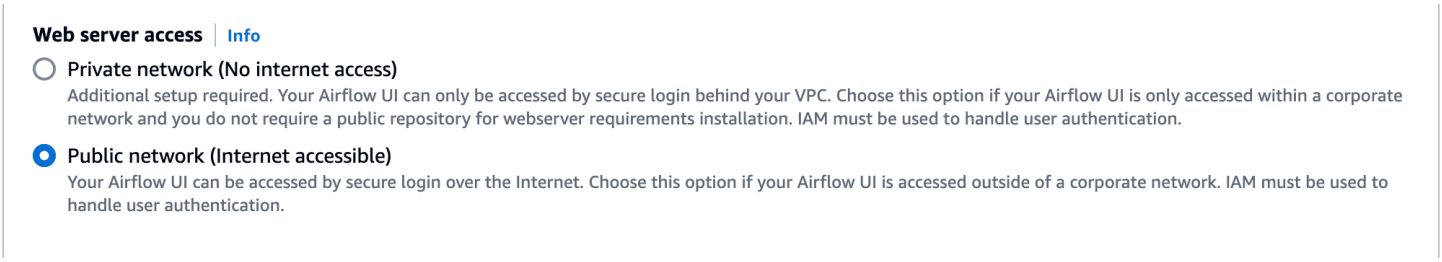
The following architectural diagram shows an Amazon MWAA environment with a public web server.

Public Web Server Option



The public network access mode allows the Apache Airflow UI to be accessed *over the internet* by users granted access to the IAM policy for your environment.

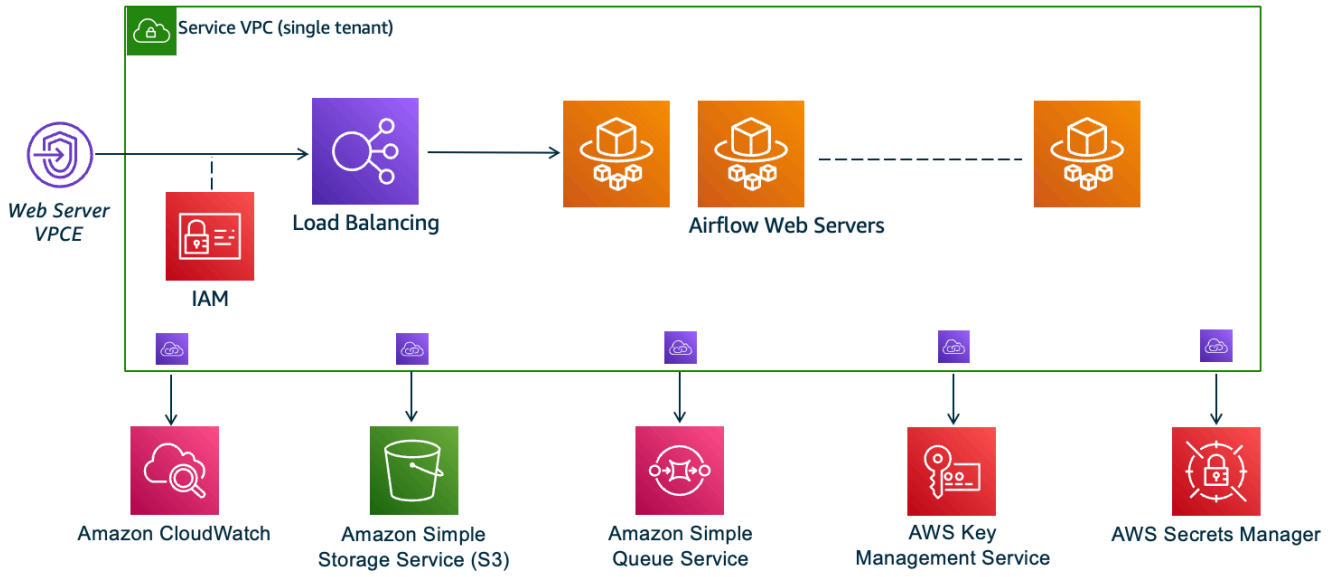
The following image shows where to find the **Public network** option on the Amazon MWAA console.



Private network

The following architectural diagram shows an Amazon MWAA environment with a private web server.

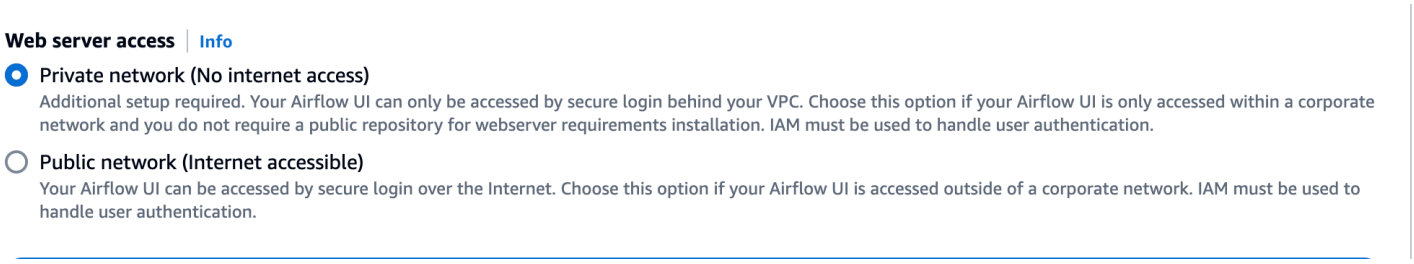
Private Web Server Option



The private network access mode limits access to the Apache Airflow UI to users *within your Amazon VPC* that have been granted access to the IAM policy for your environment.

When you create an environment with private web server access, you must package all of your dependencies in a Python wheel archive (`.whl`), then reference the `.whl` in your `requirements.txt`. For instructions on packaging and installing your dependencies using wheel, see Managing dependencies using Python wheel.

The following image shows where to find the **Private network** option on the Amazon MWAA console.



Access modes overview

This section describes the VPC endpoints (AWS PrivateLink) created in your Amazon VPC when you choose the **Public network** or **Private network** access mode.

Public network access mode

If you chose the **Public network** access mode for your Apache Airflow *Web server*, network traffic is publicly routed *over the Internet*.

- Amazon MWAA creates a VPC interface endpoint for your Amazon Aurora PostgreSQL metadata database. The endpoint is created in the Availability Zones mapped to your private subnets and is independent from other AWS accounts.
- Amazon MWAA then binds an IP address from your private subnets to the interface endpoints. This is designed to support the best practice of binding a single IP from each Availability Zone of the Amazon VPC.

Private network access mode

If you chose the **Private network** access mode for your Apache Airflow *Web server*, network traffic is privately routed *within your Amazon VPC*.

- Amazon MWAA creates a VPC interface endpoint for your Apache Airflow *Web server*, and an interface endpoint for your Amazon Aurora PostgreSQL metadata database. The endpoints are created in the Availability Zones mapped to your private subnets and is independent from other AWS accounts.
- Amazon MWAA then binds an IP address from your private subnets to the interface endpoints. This is designed to support the best practice of binding a single IP from each Availability Zone of the Amazon VPC.

To learn more, see the section called “Example use cases for an Amazon VPC and Apache Airflow access mode”.

Setup for private and public access modes

The following section describes the additional setup and configurations you'll need based on the Apache Airflow access mode you've chosen for your environment.

Setup for public network

If you choose the **Public network** option for your Apache Airflow *Web server*, you can begin using the Apache Airflow UI after you create your environment.

You'll need to take the following steps to configure access for your users, and permission for your environment to use other AWS services.

1. **Add permissions**. Amazon MWAA needs permission to use other AWS services. When you create an environment, Amazon MWAA creates a service-linked role that allows it to use certain IAM actions for Amazon Elastic Container Registry (Amazon ECR), CloudWatch Logs, and Amazon EC2.

   You can add permission to use additional actions for these services, or to use other AWS services by adding permissions to your execution role. To learn more, see Amazon MWAA execution role.
2. **Create user policies**. You may need to create multiple IAM policies for your users to configure access to your environment and Apache Airflow UI. To learn more, see Accessing an Amazon MWAA environment.

Setup for private network

If you choose the **Private network** option for your Apache Airflow *Web server*, you'll need to configure access for your users, permission for your environment to use other AWS services, and create a mechanism to access the resources in your Amazon VPC from your computer.

1. **Add permissions**. Amazon MWAA needs permission to use other AWS services. When you create an environment, Amazon MWAA creates a service-linked role that allows it to use certain IAM actions for Amazon Elastic Container Registry (Amazon ECR), CloudWatch Logs, and Amazon EC2.

   You can add permission to use additional actions for these services, or to use other AWS services by adding permissions to your execution role. To learn more, see Amazon MWAA execution role.
2. **Create user policies**. You may need to create multiple IAM policies for your users to configure access to your environment and Apache Airflow UI. To learn more, see Accessing an Amazon MWAA environment.
3. **Enable network access**. You'll need to create a mechanism in your Amazon VPC to connect to the VPC endpoint (AWS PrivateLink) for your Apache Airflow *Web server*. For example, by creating a VPN tunnel from your computer using an AWS Client VPN.

Accessing the VPC endpoint for your Apache Airflow Web server (private network access)

If you've chosen the **Private network** option, you'll need to create a mechanism in your Amazon VPC to access the VPC endpoint (AWS PrivateLink) for your Apache Airflow *Web server*. We recommend using the same Amazon VPC, VPC security group, and private subnets as your Amazon MWAA environment for these resources.

To learn more, see Managing access for VPC endpoints.

Accessing Apache Airflow

Amazon MWAA let's you access your Apache Airflow environment using multiple methods: the Apache Airflow user interface (UI) console, the Apache Airflow CLI, and the Apache Airflow REST API. You can use the Amazon MWAA console to view and invoke a DAG in your Apache Airflow UI, or use Amazon MWAA APIs to get a token and invoke a DAG. This section describes the permissions needed to access the Apache Airflow UI, how to generate a token to make Amazon MWAA API calls directly in your command shell, and the supported commands in the Apache Airflow CLI.

Topics

- Prerequisites
- Open the Apache Airflow UI
- Logging into Apache Airflow
- Create a Apache Airflow web server access token
- Setting up a custom domain for the Apache Airflow web server
- Creating an Apache Airflow CLI token
- Using the Apache Airflow REST API
- Apache Airflow CLI command reference

Prerequisites

The following section describes the preliminary steps required to use the commands and scripts in this section.

Access

- AWS account access in AWS Identity and Access Management (IAM) to the Amazon MWAA permissions policy in Apache Airflow UI access policy: AmazonMWAAWebServerAccess.
- AWS account access in AWS Identity and Access Management (IAM) to the Amazon MWAA permissions policy Full API and console access policy: AmazonMWAAFullApiAccess.

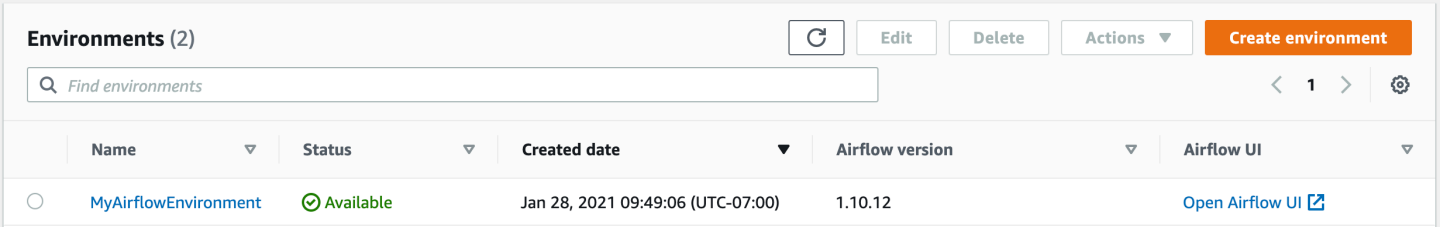
AWS CLI

The AWS Command Line Interface (AWS CLI) is an open source tool that enables you to interact with AWS services using commands in your command-line shell. To complete the steps on this page, you need the following:

- AWS CLI – Install version 2.
- AWS CLI – Quick configuration with `aws configure`.

Open the Apache Airflow UI

The following image shows the link to your Apache Airflow UI on the Amazon MWAA console.



Logging into Apache Airflow

You need Apache Airflow UI access policy: AmazonMWAAWebServerAccess permissions for your AWS account in AWS Identity and Access Management (IAM) to view your Apache Airflow UI.

To access your Apache Airflow UI

1. Open the Environments page on the Amazon MWAA console.
2. Choose an environment.
3. Choose **Open Airflow UI**.

Create a Apache Airflow web server access token

You can use the commands on this page to create a web server access token. An access token allows you access to your Amazon MWAA environment. For example, you can get a token, then deploy DAGs programmatically using Amazon MWAA APIs. The following section includes the steps to create an Apache Airflow web login token using the AWS CLI, a bash script, a POST API request, or a Python script. The token returned in the response is valid for 60 seconds.

Contents

- Prerequisites
  - Access
  - AWS CLI
- Using the AWS CLI
- Using a bash script
- Using a Python script
- What's next?

Prerequisites

The following section describes the preliminary steps required to use the commands and scripts on this page.

Access

- AWS account access in AWS Identity and Access Management (IAM) to the Amazon MWAA permissions policy in Apache Airflow UI access policy: AmazonMWAAWebServerAccess.
- AWS account access in AWS Identity and Access Management (IAM) to the Amazon MWAA permissions policy Full API and console access policy: AmazonMWAAFullApiAccess.

AWS CLI

The AWS Command Line Interface (AWS CLI) is an open source tool that enables you to interact with AWS services using commands in your command-line shell. To complete the steps on this page, you need the following:

- AWS CLI – Install version 2.
- AWS CLI – Quick configuration with `aws configure`.

Using the AWS CLI

The following example uses the create-web-login-token command in the AWS CLI to create an Apache Airflow web login token.

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

Using a bash script

The following example uses a bash script to call the create-web-login-token command in the AWS CLI to create an Apache Airflow web login token.

1. Copy the contents of the following code sample and save locally as `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. Substitute the placeholders in *red* for YOUR_HOST_NAME and YOUR_ENVIRONMENT_NAME. For example, a host name for a public network may look like this (without the *https://*):

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

3. (optional) macOS and Linux users may need to run the following command to ensure the script is executable.

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

4. Run the following script to get a web login token.

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

5. You should see the following in your command prompt:

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

Using a Python script

The following example uses the boto3 create_web_login_token method in a Python script to create an Apache Airflow web login token. You can run this script outside of Amazon MWAA. The only

thing you need to do is install the boto3 library. You may want to create a virtual environment to install the library. It assumes you have configured AWS authentication credentials for your account.

1. Copy the contents of the following code sample and save locally as `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. Substitute the placeholder in *red* for YOUR_ENVIRONMENT_NAME.
3. Run the following script to get a web login token.

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

What's next?

- Explore the Amazon MWAA API operation used to create a web login token at CreateWebLoginToken.

Setting up a custom domain for the Apache Airflow web server

Amazon Managed Workflows for Apache Airflow (Amazon MWAA) lets you to set up a custom domain for the managed Apache Airflow web server. Using a custom domain, you can access your environment's Amazon MWAA managed Apache Airflow web server using the Apache Airflow UI, the Apache Airflow CLI, or the Apache Airflow web server.

> **Note**
>
> You can only use custom domain with a private web server without internet access.

Use cases for a custom domain on Amazon MWAA

1. Share the web server domain across your cloud application on AWS — Using a custom domain lets you define a user-friendly URL for accessing the web server, instead of the generated service domain name. You can store this custom domain and share it as an environment variable in your applications.
2. Access a private web server — If you want to configure access for a web server in a VPC with no internet access, using a custom domain simplifies the URL redirection work flow.

Topics

- Configure the custom domain
- Set up the networking infrastructure

Configure the custom domain

To configure the custom domain feature, you need to provide the custom domain value via the `webserver.base_url` Apache Airflow configuration when creating or updating your Amazon MWAA environment. The following constraints apply to your custom domain name:

- The value should be a fully qualified domain name (FQDN) without any protocol or path. For example, `your-custom-domain.com`.
- Amazon MWAA does not allow a path in the URL. For example, `your-custom-domain.com/dags/` is not a valid custom domain name.
- The URL length is limited to 255 ASCII characters.
- If you provide an empty string, by default, the environment will be created with a web server URL generated by Amazon MWAA.

The following example shows using the AWS CLI to create an environment with a custom web server domain name.

```
$ aws mwaa create-environment \
    --name my-mwaa-env \
    --source-bucket-arn arn:aws:s3:::my-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
```

After the environment is created or updated, you need to set up the networking infrastructure in your AWS account to access the private web server via the custom domain.

To revert back to the default service-generated URL, update your private environment and remove the `webserver.base_url` configuration option.

Set up the networking infrastructure

Use the following steps to set up the required networking infrastructure to use with your custom domain in your AWS account.

1. Get the IP addresses for the Amazon VPC Endpoint Network Interfaces (ENI). To do this, first, use `get-environment` to find the `WebserverVpcEndpointService` for your environment.

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

   If successful, you'll see output similar to the following.

   ```
   {
       "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"
       }
   }
   ```

   Note the `WebserverVpcEndpointService` value and use it for `web-server-vpc-endpoint-service` in the following Amazon EC2 `describe-vpc-endpoints` command. `--filters Name=service-name,Values=web-server-vpc-endpoint-service-id` in the following command.

2. Retrieve the Amazon VPC endpoint details. This command fetches details about Amazon VPC endpoints that match a specific service name, returning the endpoint ID and associated network interface IDs in a text format.

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

3. Get the network interface details. This command retrieves private IP addresses for each network interface associated with the Amazon VPC endpoints identified in the previous step.

```
$ 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. Use `create-target-group` to create a new target group. You will use this target group to register the IP addresses for your web server Amazon VPC endpoints.

```
$ 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"'
```

   Register the IP addresses using the `register-targets` command.

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

5. Request an ACM certificate. Skip this step if you are using an existing certificate.

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

6. Configure an Application Load Balancer. First, create the load balancer, then create a listener for the load balancer. Specify the ACM certificate you created in the previous step.

```
$ 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
```

   If you use a Network Load Balancer in a private subnet, set up a bastion host or AWS VPN tunnel to access the web server.

7. Create a hosted zone using Route 53 for the domain.

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

   Create an A record for the domain. To do this using the AWS CLI, get the hosted zone ID using `list-hosted-zones-by-name` then apply the record with `change-resource-record-sets`.

```
$ 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. Update the security group rules for the web server Amazon VPC endpoint to follow the principle of least privilege by allowing HTTPS traffic only from the public subnets where the Application Load Balancer is located. Save the following JSON locally. For example, as `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"
```

```
    }
  ]
}
]
```

Run the following Amazon EC2 command to update your ingress security group rules. Specify the JSON file for `--ip-permissions`.

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

Run the following Amazon EC2 command to update your egress rules.

```
$ 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
```

Open the Amazon MWAA console and navigate to the Apache Airflow UI. If you are setting up an Network Load Balancer in a private subnet instead of the Application Load Balancer used here, you must access the web server with one of the following options.

- the section called “Tutorial: Linux Bastion Host”
- the section called “Tutorial: AWS Client VPN”

Creating an Apache Airflow CLI token

You can use the commands on this page to generate a CLI token, and then make Amazon Managed Workflows for Apache Airflow API calls directly in your command shell. For example, you can get a token, then deploy DAGs programmatically using Amazon MWAA APIs. The following section includes the steps to create an Apache Airflow CLI token using the AWS CLI, a curl script, a Python script, or a bash script. The token returned in the response is valid for 60 seconds.

Note

The AWS CLI token is intended as a replacement for synchronous shell actions, not asynchronous API commands. As such, available concurrency is limited. To ensure that the web server remains responsive for users, it is recommended not to open a new AWS CLI request until the previous one completes successfully.

Contents

- Prerequisites
  - Access
  - AWS CLI
- Using the AWS CLI
- Using a curl script
- Using a bash script
- Using a Python script
- What's next?

Prerequisites

The following section describes the preliminary steps required to use the commands and scripts on this page.

Access

- AWS account access in AWS Identity and Access Management (IAM) to the Amazon MWAA permissions policy in Apache Airflow UI access policy: AmazonMWAAWebServerAccess.
- AWS account access in AWS Identity and Access Management (IAM) to the Amazon MWAA permissions policy Full API and console access policy: AmazonMWAAFullApiAccess.

AWS CLI

The AWS Command Line Interface (AWS CLI) is an open source tool that enables you to interact with AWS services using commands in your command-line shell. To complete the steps on this page, you need the following:

- AWS CLI – Install version 2.
- AWS CLI – Quick configuration with `aws configure`.

Using the AWS CLI

The following example uses the create-cli-token command in the AWS CLI to create an Apache Airflow CLI token.

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

Using a curl script

The following example uses a curl script to call the create-web-login-token command in the AWS CLI to invoke the Apache Airflow CLI via an endpoint on the Apache Airflow web server.

Apache Airflow v2

1. Copy the curl statement from your text file and paste it in your command shell.

   > **Note**
   >
   > After copying it to your clipboard, you may need to use **Edit > Paste** from your shell menu.

   ```
   CLI_JSON=$(aws mwaa --region YOUR_REGION create-cli-token --
   name YOUR_ENVIRONMENT_NAME) \
     && CLI_TOKEN=$(echo $CLI_JSON | jq -r '.CliToken') \
     && WEB_SERVER_HOSTNAME=$(echo $CLI_JSON | jq -r '.WebServerHostname') \
     && CLI_RESULTS=$(curl --request POST "https://$WEB_SERVER_HOSTNAME/aws_mwaa/
   cli" \
     --header "Authorization: Bearer $CLI_TOKEN" \
     --header "Content-Type: text/plain" \
     --data-raw "dags trigger YOUR_DAG_NAME") \
     && echo "Output:" \
     && echo $CLI_RESULTS | jq -r '.stdout' | base64 --decode \
     && echo "Errors:" \
     && echo $CLI_RESULTS | jq -r '.stderr' | base64 --decode
   ```

2. Substitute the placeholders for YOUR_REGION with the AWS region for your environment, YOUR_DAG_NAME, and YOUR_ENVIRONMENT_NAME. For example, a host name for a public network may look like this (without the *https://*):

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

3. You should see the following in your command prompt:

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

Apache Airflow v1

1. Copy the cURL statement from your text file and paste it in your command shell.

> **Note**
>
> After copying it to your clipboard, you may need to use **Edit > Paste** from your shell menu.

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

2. Substitute the placeholders for YOUR_REGION with the AWS region for your environment, YOUR_DAG_NAME, and YOUR_HOST_NAME. For example, a host name for a public network may look like this (without the *https://*):

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

3. You should see the following in your command prompt:

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

4. Substitute the placeholders for YOUR_ENVIRONMENT_NAME and YOUR_DAG_NAME.

Using a bash script

The following example uses a bash script to call the create-cli-token command in the AWS CLI to create an Apache Airflow CLI token.

Apache Airflow v2

1. Copy the contents of the following code sample and save locally as `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. Substitute the placeholders in *red* for YOUR_ENVIRONMENT_NAME, YOUR_HOST_NAME, and YOUR_DAG_NAME. For example, a host name for a public network may look like this (without the *https://*):

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

3. (optional) macOS and Linux users may need to run the following command to ensure the script is executable.

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

4. Run the following script to create an Apache Airflow CLI token.

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

Apache Airflow v1

1. Copy the contents of the following code sample and save locally as `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 "trigger_dag YOUR_DAG_NAME"
```

2. Substitute the placeholders in *red* for YOUR_ENVIRONMENT_NAME, YOUR_HOST_NAME, and YOUR_DAG_NAME. For example, a host name for a public network may look like this (without the *https://*):

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

3. (optional) macOS and Linux users may need to run the following command to ensure the script is executable.

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

4. Run the following script to create an Apache Airflow CLI token.

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

Using a Python script

The following example uses the boto3 create_cli_token method in a Python script to create an Apache Airflow CLI token and trigger a DAG. You can run this script outside of Amazon MWAA. The only thing you need to do is install the boto3 library. You may want to create a virtual environment

to install the library. It assumes you have configured AWS authentication credentials for your account.

Apache Airflow v2

1. Copy the contents of the following code sample and save locally as `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. Substitute the placeholders for YOUR_ENVIRONMENT_NAME and YOUR_DAG_NAME.
3. Run the following script to create an Apache Airflow CLI token.

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

Apache Airflow v1

1. Copy the contents of the following code sample and save locally as `create-cli-token.py`.

```
import boto3
import json
import requests
import base64

mwaa_env_name = 'YOUR_ENVIRONMENT_NAME'
dag_name = 'YOUR_DAG_NAME'
mwaa_cli_command = 'trigger_dag'

client = boto3.client('mwaa')

mwaa_cli_token = client.create_cli_token(
```

```
    Name=mwaa_env_name
)

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

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

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

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

2. Substitute the placeholders for YOUR_ENVIRONMENT_NAME and YOUR_DAG_NAME.
3. Run the following script to create an Apache Airflow CLI token.

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

What's next?

- Explore the Amazon MWAA API operation used to create a CLI token at CreateCliToken.

Using the Apache Airflow REST API

Amazon Managed Workflows for Apache Airflow (Amazon MWAA) supports interacting with your Apache Airflow environments directly using the Apache Airflow REST API for environments running Apache Airflow v2.4.3 and above. This lets you access and manage your Amazon MWAA

environments programmatically, providing a standardized way to invoke data orchestration workflows, manage your DAGs, and monitor the status of various Apache Airflow components such as the metadata database, triggerer, and scheduler.

In order to support scalability while using the Apache Airflow REST API, Amazon MWAA provides you with the option to horizontally scale web server capacity to handle increased demand, whether from REST API requests, command line interface (CLI) usage, or more concurrent Apache Airflow user interface (UI) users. For more information on how Amazon MWAA scales web servers, see the section called "Configuring web server auto scaling".

You can use the Apache Airflow REST API to implement the following use-cases for your environments:

- **Programmatic access** – You can now start Apache Airflow DAG runs, manage datasets, and retrieve the status of various components such as the metadata database, triggerers, and schedulers without relying on the Apache Airflow UI or CLI.
- **Integrate with external applications and microservices** – REST API support allows you to build custom solutions that integrate your Amazon MWAA environments with other systems. For example, you can start workflows in response to events from external systems, such as completed database jobs or new user sign-ups.
- **Centralized monitoring** – You can build monitoring dashboards that aggregate the status of your DAGs across multiple Amazon MWAA environments, enabling centralized monitoring and management.

For more information about the Apache Airflow REST API, see The Apache Airflow REST API Reference.

By using `InvokeRestApi`, you can access the Apache Airflow REST API using AWS credentials. Alternatively, you can also access it by obtaining a web server access token and then using the token to call it.

Note

- If you encounter an error with the message "Update your environment to use `InvokeRestApi`" while using the `InvokeRestApi` operation, it indicates that you need to update your Amazon MWAA environment. This error occurs when your Amazon MWAA environment is not compatible with the latest changes related to the `InvokeRestApi`

feature. To resolve this issue, update your Amazon MWAA environment to incorporate the necessary changes for the `InvokeRestApi` feature.

- The `InvokeRestApi` operation has a default timeout duration of 10 seconds. If the operation does not complete within this 10-second timeframe, it will be automatically terminated, and an error will be raised. Ensure that your REST API calls are designed to complete within this timeout period to avoid encountering errors.

The following examples show how you to make API calls to the Apache Airflow REST API and start a new DAG run:

Topics

- Granting access to the Apache Airflow REST API: airflow:InvokeRestApi
- Calling the Apache Airflow REST API
- Creating a web server session token and calling the Apache Airflow REST API

Granting access to the Apache Airflow REST API: `airflow:InvokeRestApi`

To access the Apache Airflow REST API using AWS credential, you must grant the `airflow:InvokeRestApi` permission in your IAM policy. In the following policy sample, specify the `Admin`, `Op`, `User`, `Viewer` or the `Public` role in `{airflow-role}` to customize the level of user access. For more information, see Default Roles in the *Apache Airflow reference guide*.

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

```
}
```

> **Note**
>
> While configuring a private web server, the `InvokeRestApi` action cannot be invoked from outside of a Virtual Private Cloud (VPC). You can use the `aws:SourceVpc` key to apply more granular access control for this operation. For more information, see aws:SourceVpc.

Calling the Apache Airflow REST API

This following sample script covers how to use the Apache Airflow REST API to list the available DAGs in your environment and how to create an Apache Airflow variable:

```
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)
```

Creating a web server session token and calling the Apache Airflow REST API

To create a web server access token, use the following Python function. This function first calls the Amazon MWAA API to obtain a web login token. The web login token, which expires after 60 seconds, is then exchanged for a web *session* token, which lets you access the web server and use the Apache Airflow REST API. If you require more than 10 transactions per second (TPS) of throttling capacity, you can use this method to access the Apache Airflow REST API.

Note

The session token expires after 12 hours.

```
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
```

Once authentication is complete, you have the credentials to start sending requests to the API endpoints. In the example below, use the endpoint `dags/{dag_id}/dagRuns`.

```
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 command reference

This topic describes the supported and unsupported Apache Airflow CLI commands on Amazon Managed Workflows for Apache Airflow.

Contents

- Prerequisites
  - Access
  - AWS CLI
- What's changed in v2
- Supported CLI commands
  - Supported commands
  - Using commands that parse DAGs
- Sample code
  - Set, get or delete an Apache Airflow v2 variable
  - Add a configuration when triggering a DAG
  - Run CLI commands on an SSH tunnel to a bastion host
  - Samples in GitHub and AWS tutorials

Prerequisites

The following section describes the preliminary steps required to use the commands and scripts on this page.

Access

- AWS account access in AWS Identity and Access Management (IAM) to the Amazon MWAA permissions policy in Apache Airflow UI access policy: AmazonMWAAWebServerAccess.
- AWS account access in AWS Identity and Access Management (IAM) to the Amazon MWAA permissions policy Full API and console access policy: AmazonMWAAFullApiAccess.

AWS CLI

The AWS Command Line Interface (AWS CLI) is an open source tool that enables you to interact with AWS services using commands in your command-line shell. To complete the steps on this page, you need the following:

- AWS CLI – Install version 2.
- AWS CLI – Quick configuration with `aws configure`.

What's changed in v2

- **New: Airflow CLI command structure**. The Apache Airflow v2 CLI is organized so that related commands are grouped together as subcommands, which means you need to update Apache Airflow v1 scripts if you want to upgrade to Apache Airflow v2. For example, `unpause` in Apache Airflow v1 is now `dags unpause` in Apache Airflow v2. To learn more, see Airflow CLI changes in 2 in the *Apache Airflow reference guide*.

Supported CLI commands

The following section lists the Apache Airflow CLI commands available on Amazon MWAA.

Supported commands

Apache Airflow v2

Minor versions	Command	
v2.0+	cheat-sheet	
v2.0+	connections add	

Minor versions	Command
v2.0+	connections delete
v2.2+ (note)	dags backfill
v2.0+	dags delete
v2.2+ (note)	dags list
v2.0+	dags list-jobs
v2.6+	dags list-import-errors
v2.2+ (note)	dags list-runs
v2.2+ (note)	dags next-execution
v2.0+	dags pause
v2.0+	dags report
v2.4+	dags reserialize
v2.0+	dags show
v2.0+	dags state
v2.0+	dags test
v2.0+	dags trigger
v2.0+	dags unpause
v2.4+	db clean
v2.0+	providers behaviours
v2.0+	providers get
v2.0+	providers hooks

Minor versions	Command
v2.0+	providers links
v2.0+	providers list
v2.8+	providers notifications
v2.6+	providers secrets
v2.7+	providers triggerer
v2.0+	providers widgets
v2.6+	roles add-perms
v2.6+	roles del-perms
v2.6+	roles create
v2.0+	roles list
v2.0+	tasks clear
v2.0+	tasks failed-deps
v2.0+	tasks list
v2.0+	tasks render
v2.0+	tasks state
v2.0+	tasks states-for-dag-run
v2.0+	tasks test
v2.0+	variables delete
v2.0+	variables get
v2.0+	variables set

Minor versions	Command
v2.0+	variables list
v2.0+	version

Using commands that parse DAGs

If your environment is running Apache Airflow v1.10.12 or v2.0.2, CLI commands that parse DAGs will fail if the DAG uses plugins that depend on packages installed through a `requirements.txt`:

Apache Airflow v2.0.2

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

You can use these CLI commands if your DAGs do not use plugins that depend on packages installed through a `requirements.txt`.

Sample code

The following section contains examples of different ways to use the Apache Airflow CLI.

Set, get or delete an Apache Airflow v2 variable

You can use the following sample code to set, get or delete a variable in the format of `<script> <mwaa env name> get | set | delete <variable> <variable value> </variable> </variable>`.

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

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

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

```
else
    echo "Not a valid command"
    exit 1
fi

CLI_JSON=$(aws mwaa --region $AWS_REGION create-cli-token --name $1) \
    && CLI_TOKEN=$(echo $CLI_JSON | jq -r '.CliToken') \
    && WEB_SERVER_HOSTNAME=$(echo $CLI_JSON | jq -r '.WebServerHostname') \
    && CLI_RESULTS=$(curl --request POST "https://$WEB_SERVER_HOSTNAME/aws_mwaa/cli" \
    --header "Authorization: Bearer $CLI_TOKEN" \
    --header "Content-Type: text/plain" \
    --data-raw "$dag" ) \
    && echo "Output:" \
    && echo $CLI_RESULTS | jq -r '.stdout' | base64 --decode \
    && echo "Errors:" \
    && echo $CLI_RESULTS | jq -r '.stderr' | base64 --decode
```

Add a configuration when triggering a DAG

You can use the following sample code with Apache Airflow v1 and Apache Airflow v2 to add a configuration when triggering a DAG, such as `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)
```

Run CLI commands on an SSH tunnel to a bastion host

The following example shows how to run Airflow CLI commands using an SSH tunnel proxy to a Linux Bastion Host.

Using 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
   ```

Samples in GitHub and AWS tutorials

- Working with Apache Airflow v2.0.2 parameters and variables in Amazon Managed Workflows for Apache Airflow

Managing connections to Apache Airflow

This chapter describes how to configure an Apache Airflow connection for an Amazon Managed Workflows for Apache Airflow environment.

Topics

- Overview of Apache Airflow variables and connections
- Apache Airflow provider packages installed on Amazon MWAA environments
- Overview of connection types
- Configuring an Apache Airflow connection using a AWS Secrets Manager secret

Overview of Apache Airflow variables and connections

In some cases, you may want to specify additional connections or variables for an environment, such as an AWS profile, or to add your execution role in a connection object in the Apache Airflow metastore, then refer to the connection from within a DAG.

- **Self-managed Apache Airflow**. On a self-managed Apache Airflow installation, you set Apache Airflow configuration options in `airflow.cfg`.

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

- **Apache Airflow on Amazon MWAA**. On Amazon MWAA, you need to add these configuration settings as Apache Airflow configuration options on the Amazon MWAA console. Apache Airflow configuration options are written as environment variables to your environment and override all other existing configurations for the same setting.

Apache Airflow provider packages installed on Amazon MWAA environments

Amazon MWAA installs provider extras for Apache Airflow v2 and above connection types when you create a new environment. Installing provider packages allows you to view a connection type

in the Apache Airflow UI. It also means you don't need to specify these packages as a Python dependency in your `requirements.txt` file. This page lists the Apache Airflow provider packages installed by Amazon MWAA for all Apache Airflow v2 environments.

> **Note**
>
> For Apache Airflow v2 and above, Amazon MWAA installs Watchtower version 2.0.1 after perfming `pip3 install -r requirements.txt`, to ensure compatibility with CloudWatch logging is not overridden by other Python library installations.

Contents

- Provider packages for Apache Airflow v2.10.3 connections
- Provider packages for Apache Airflow v2.10.1 connections
- Provider packages for Apache Airflow v2.9.2 connections
- Provider packages for Apache Airflow v2.8.1 connections
- Provider packages for Apache Airflow v2.7.2 connections
- Provider packages for Apache Airflow v2.6.3 connections
- Provider packages for Apache Airflow v2.5.1 connections
- Provider packages for Apache Airflow v2.4.3 connections
- Provider packages for Apache Airflow v2.2.2 connections
- Provider packages for Apache Airflow v2.0.2 connections
- Specifying newer provider packages

Provider packages for Apache Airflow v2.10.3 connections

When you create an Amazon MWAA environment in Apache Airflow v2.10.3, Amazon MWAA installs the following provider packages used for Apache Airflow connections.

> **Note**
>
> You can specify the latest supported version of `apache-airflow-providers-amazon` to upgrade this provider. For more information on specifying newer versions, see the section called “Specifying newer provider packages”.

Connection type	Package
AWS Connection	apache-airflow-providers-amazon[aiobotocore]==9.0.0
Postgres Connection	apache-airflow-providers-postgres==5.13.1
FTP Connection	apache-airflow-providers-ftp==3.11.1
Fab Connection	apache-airflow-providers-fab==1.5.0
Celery Connection	apache-airflow-providers-celery==3.8.3
HTTP Connection	apache-airflow-providers-http==4.13.2
IMAP Connection	apache-airflow-providers-imap==3.7.0
Common SQL	apache-airflow-providers-common-sql==1.19.0
SQLite Connection	apache-airflow-providers-sqlite==3.9.0
SMTP Connection	apache-airflow-providers-smtp==1.8.0

Provider packages for Apache Airflow v2.10.1 connections

When you create an Amazon MWAA environment in Apache Airflow v2.10.1, Amazon MWAA installs the following provider packages used for Apache Airflow connections.

> **Note**
>
> You can specify the latest supported version of `apache-airflow-providers-amazon` to upgrade this provider. For more information on specifying newer versions, see the section called “Specifying newer provider packages”.

Connection type	Package
AWS Connection	apache-airflow-providers-amazon[aiobotocore]==8.28.0
Postgres Connection	apache-airflow-providers-postgres==5.12.0
FTP Connection	apache-airflow-providers-ftp==3.11.0
Fab Connection	apache-airflow-providers-fab==1.3.0
Celery Connection	apache-airflow-providers-celery==3.8.1
HTTP Connection	apache-airflow-providers-http==4.13.0
IMAP Connection	apache-airflow-providers-imap==3.7.0
Common SQL	apache-airflow-providers-common-sql==1.16.0
SQLite Connection	apache-airflow-providers-sqlite==3.9.0
SMTP Connection	apache-airflow-providers-smtp==1.8.0

Provider packages for Apache Airflow v2.9.2 connections

When you create an Amazon MWAA environment in Apache Airflow v2.9.2, Amazon MWAA installs the following provider packages used for Apache Airflow connections.

> **Note**
>
> You can specify the latest supported version of `apache-airflow-providers-amazon` to upgrade this provider. For more information on specifying newer versions, see the section called “Specifying newer provider packages”.

Connection type	Package
AWS Connection	apache-airflow-providers-amazon[aiobotocore]==8.24.0
Postgres Connection	apache-airflow-providers-postgres==5.11.1
FTP Connection	apache-airflow-providers-ftp==3.9.1
Fab Connection	apache-airflow-providers-fab==1.1.1
Celery Connection	apache-airflow-providers-celery==3.7.2
HTTP Connection	apache-airflow-providers-http==4.11.1
IMAP Connection	apache-airflow-providers-imap==3.6.1
Common SQL	apache-airflow-providers-common-sql==1.14.0
SQLite Connection	apache-airflow-providers-sqlite==3.8.1
SMTP Connection	apache-airflow-providers-smtp==1.7.1

Provider packages for Apache Airflow v2.8.1 connections

When you create an Amazon MWAA environment in Apache Airflow v2.8.1, Amazon MWAA installs the following provider packages used for Apache Airflow connections.

> **Note**
>
> You can specify the latest supported version of `apache-airflow-providers-amazon` to upgrade this provider. For more information on specifying newer versions, see the section called “Specifying newer provider packages”.

Connection type	Package
AWS Connection	apache-airflow-providers-amazon[aiobotocore]==8.16.0
Postgres Connection	apache-airflow-providers-postgres==5.10.0
FTP Connection	apache-airflow-providers-ftp==3.7.0
Celery Connection	apache-airflow-providers-celery==3.5.1
HTTP Connection	apache-airflow-providers-http==4.8.0
IMAP Connection	apache-airflow-providers-imap==3.5.0
Common SQL	apache-airflow-providers-common-sql==1.10.0
SQLite Connection	apache-airflow-providers-sqlite==3.7.0

Provider packages for Apache Airflow v2.7.2 connections

When you create an Amazon MWAA environment in Apache Airflow v2.7.2, Amazon MWAA installs the following provider packages used for Apache Airflow connections.

> **Note**
>
> You can specify the latest supported version of `apache-airflow-providers-amazon` to upgrade this provider. For more information on specifying newer versions, see the section called “Specifying newer provider packages”.

Connection type	Package
AWS Connection	apache-airflow-providers-amazon[aiobotocore]==8.7.1
Postgres Connection	apache-airflow-providers-postgres==5.6.1

Connection type	Package
FTP Connection	apache-airflow-providers-ftp==3.5.2
Celery Connection	apache-airflow-providers-celery==3.3.4
HTTP Connection	apache-airflow-providers-http==4.5.2
IMAP Connection	apache-airflow-providers-imap==3.3.2
Common SQL	apache-airflow-providers-common-sql==1.7.2
SQLite Connection	apache-airflow-providers-sqlite==3.4.3

Provider packages for Apache Airflow v2.6.3 connections

When you create an Amazon MWAA environment in Apache Airflow v2.6.3, Amazon MWAA installs the following provider packages used for Apache Airflow connections.

> **Note**
>
> You can specify the latest supported version of `apache-airflow-providers-amazon` to upgrade this provider. For more information on specifying newer versions, see the section called "Specifying newer provider packages".

Connection type	Package
AWS Connection	apache-airflow-providers-amazon[aiobotocore]==8.2.0
Postgres Connection	apache-airflow-providers-postgres==5.5.1
FTP Connection	apache-airflow-providers-ftp==3.4.2
Celery Connection	apache-airflow-providers-celery==3.2.1
HTTP Connection	apache-airflow-providers-http==4.4.2

Connection type	Package
IMAP Connection	apache-airflow-providers-imap==3.2.2
Common SQL	apache-airflow-providers-common-sql==1.5.2
SQLite Connection	apache-airflow-providers-sqlite==3.4.2

Provider packages for Apache Airflow v2.5.1 connections

When you create an Amazon MWAA environment in Apache Airflow v2.5.1, Amazon MWAA installs the following provider packages used for Apache Airflow connections.

> **Note**
>
> You can specify the latest supported version of `apache-airflow-providers-amazon` to upgrade this provider. For more information on specifying newer versions, see the section called "Specifying newer provider packages".

Connection type	Package
AWS Connection	apache-airflow-providers-amazon==7.1.0
Postgres Connection	apache-airflow-providers-postgres==5.4.0
FTP Connection	apache-airflow-providers-ftp==3.3.0
Celery Connection	apache-airflow-providers-celery==3.1.0
HTTP Connection	apache-airflow-providers-http==4.1.1
IMAP Connection	apache-airflow-providers-imap==3.1.1
Common SQL	apache-airflow-providers-common-sql==1.3.3
SQLite Connection	apache-airflow-providers-sqlite==3.3.1

Provider packages for Apache Airflow v2.4.3 connections

When you create an Amazon MWAA environment in Apache Airflow v2.4.3, Amazon MWAA installs the following provider packages used for Apache Airflow connections.

Connection type	Package
AWS Connection	apache-airflow-providers-amazon==6.0.0
Postgres Connection	apache-airflow-providers-postgres==5.2.2
FTP Connection	apache-airflow-providers-ftp==3.1.0
Celery Connection	apache-airflow-providers-celery==3.0.0
HTTP Connection	apache-airflow-providers-http==4.0.0
IMAP Connection	apache-airflow-providers-imap==3.0.0
Common SQL	apache-airflow-providers-common-sql==1.2.0
SQLite Connection	apache-airflow-providers-sqlite==3.2.1

Provider packages for Apache Airflow v2.2.2 connections

When you create an Amazon MWAA environment in Apache Airflow v2.2.2, Amazon MWAA installs the following provider packages used for Apache Airflow connections.

Connection type	Package
AWS Connection	apache-airflow-providers-amazon==2.4.0
Postgres Connection	apache-airflow-providers-postgres==2.3.0
FTP Connection	apache-airflow-providers-ftp==2.0.1
Celery Connection	apache-airflow-providers-celery==2.1.0
HTTP Connection	apache-airflow-providers-http==2.0.1

Connection type	Package
IMAP Connection	apache-airflow-providers-imap==2.0.1
SQLite Connection	apache-airflow-providers-sqlite==2.0.1

Provider packages for Apache Airflow v2.0.2 connections

When you create an Amazon MWAA environment in Apache Airflow v2.0.2, Amazon MWAA installs the following provider packages used for Apache Airflow connections.

Connection type	Package
Tableau Connection	apache-airflow-providers-tableau==1.0.0
Databricks Connection	apache-airflow-providers-databricks==1.0.1
SSH Connection	apache-airflow-providers-ssh==1.3.0
Postgres Connection	apache-airflow-providers-postgres==1.0.2
Docker Connection	apache-airflow-providers-docker==1.2.0
Oracle Connection	apache-airflow-providers-oracle==1.1.0
Presto Connection	apache-airflow-providers-presto==1.0.2
SFTP Connection	apache-airflow-providers-sftp==1.2.0

Specifying newer provider packages

Beginning with Apache Airflow v2.7.2, your requirements file must include a `--constraint` statement. If you do not provide a constraint, Amazon MWAA will specify one for you to ensure the packages listed in your requirements are compatible with the version of Apache Airflow you are using.

Apache Airflow constraints files specify the provider versions available at the time of a Apache Airflow release. In many cases, however, newer providers are compatible with that version of

Apache Airflow. Because you must use constraints, to specify a newer version of a provider package, you can modify the constraints file for a specific provider version:

1. Download the version-specific constraints file from https://raw.githubusercontent.com/apache/airflow/constraints-2.7.2/constraints-3.11.txt"
2. Modify the `apache-airflow-providers-amazon` version in the constraints file to the version you want to use.
3. Save the modified constraints file to the Amazon S3 dags folder of your Amazon MWAA environment, for example, as `constraints-3.11-updated.txt`
4. Specify your requirements as shown in the following.

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

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

> **Note**
>
> If you are using a private web server, we recommend you package the required libraries as WHL files by using the Amazon MWAA local-runner.

Overview of connection types

Apache Airflow stores connections as a connection URI string. It provides a connections template in the Apache Airflow UI to generate the connection URI string, regardless of the connection type. If a connection template is not available in the Apache Airflow UI, an alternate connection template can be used to generate this connection URI string, such as using the HTTP connection template. The primary difference is the URI prefix, such as `my-conn-type://`, which Apache Airflow providers typically ignore for a connection. This page describes how to use connection templates in the Apache Airflow UI interchangeably for different connection types.

> **Warning**
>
> Do not overwrite the `aws_default` connection in Amazon MWAA. Amazon MWAA uses this connection to perform a variety of critical tasks, such as collecting task logs.

Overwriting this connection might result in data loss and disruptions to your environment availability.

Topics

- Example connection URI string
- Example connection template
- Example using an HTTP connection template for a Jdbc connection

Example connection URI string

The following example shows a connection URI string for the MySQL connection type.

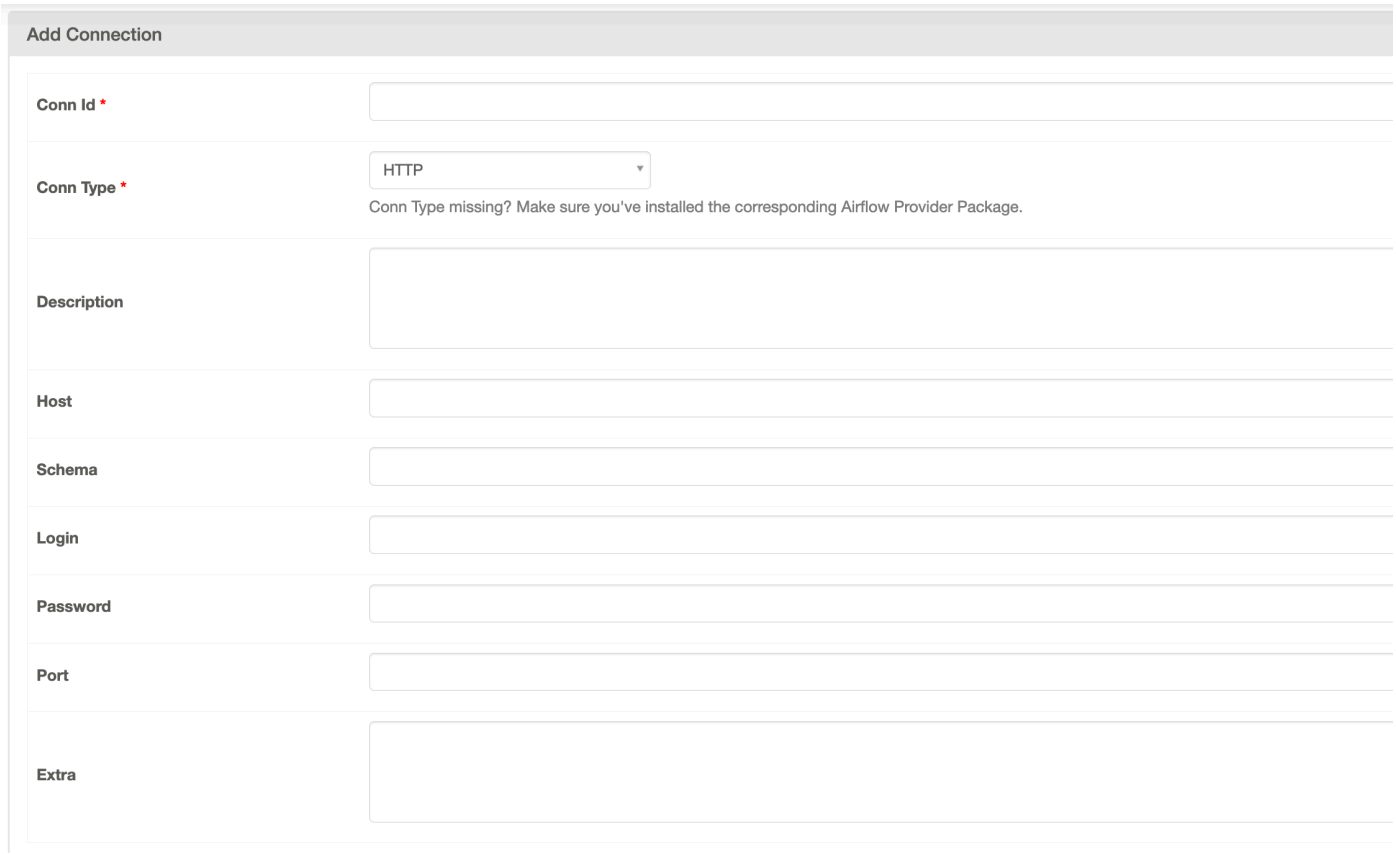
```
'mysql://288888a0-50a0-888-9a88-1a111aaa0000.a1.us-east-1.airflow.amazonaws.com
%2Fhome?role_arn=arn%3Aaws%3Aiam%3A%3A001122332255%3Arole%2Fservice-role%2FAmazonMWAA-
MyAirflowEnvironment-iAaaaA&region_name=us-east-1'
```

Example connection template

The following example shows the HTTP connection template in the Apache Airflow UI.

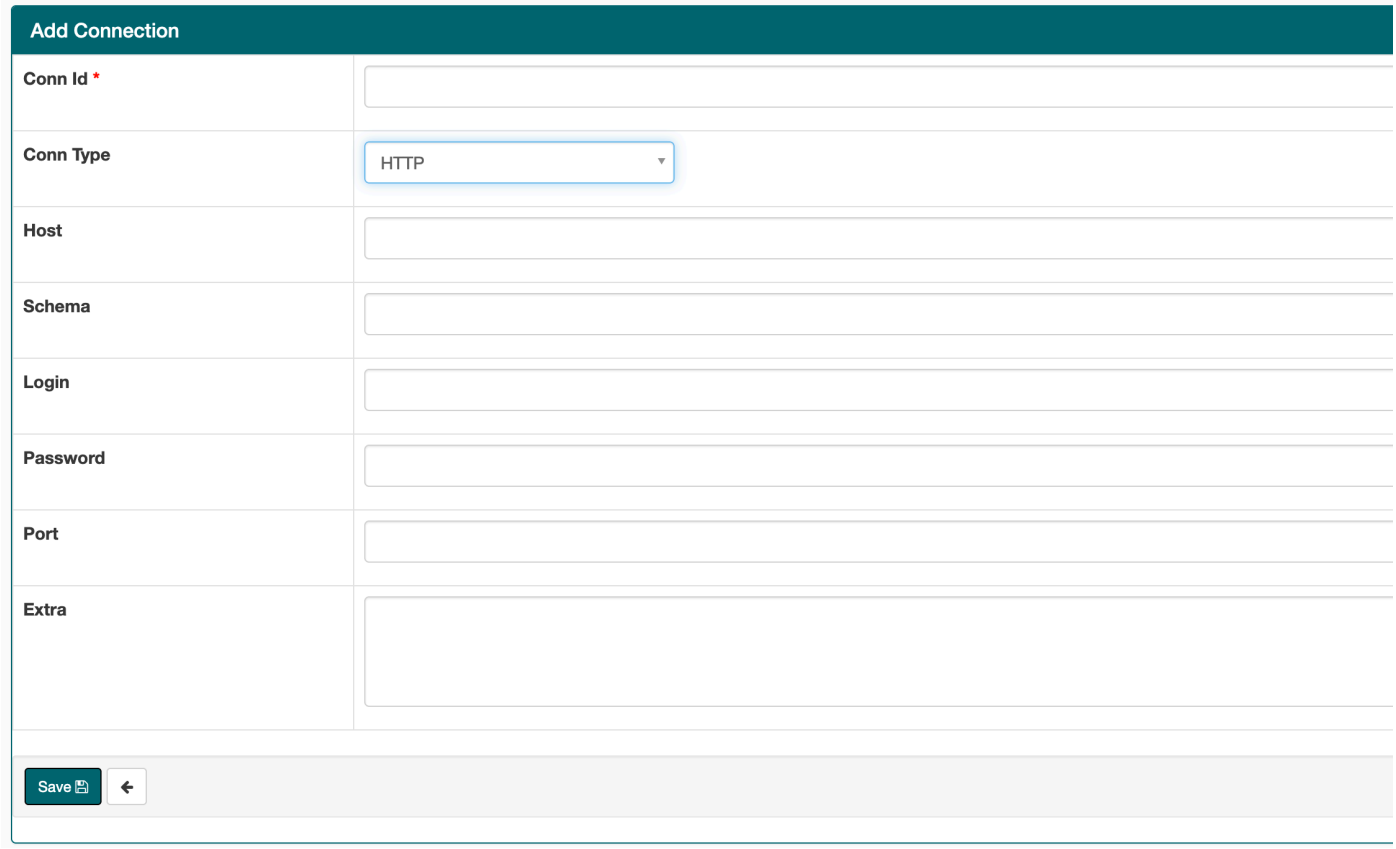
Apache Airflow v2

The following example shows the HTTP connection template for Apache Airflow v2 in the Apache Airflow UI.



Apache Airflow v1

The following example shows the HTTP connection template for Apache Airflow v1 in the Apache Airflow UI.



Example using an HTTP connection template for a Jdbc connection

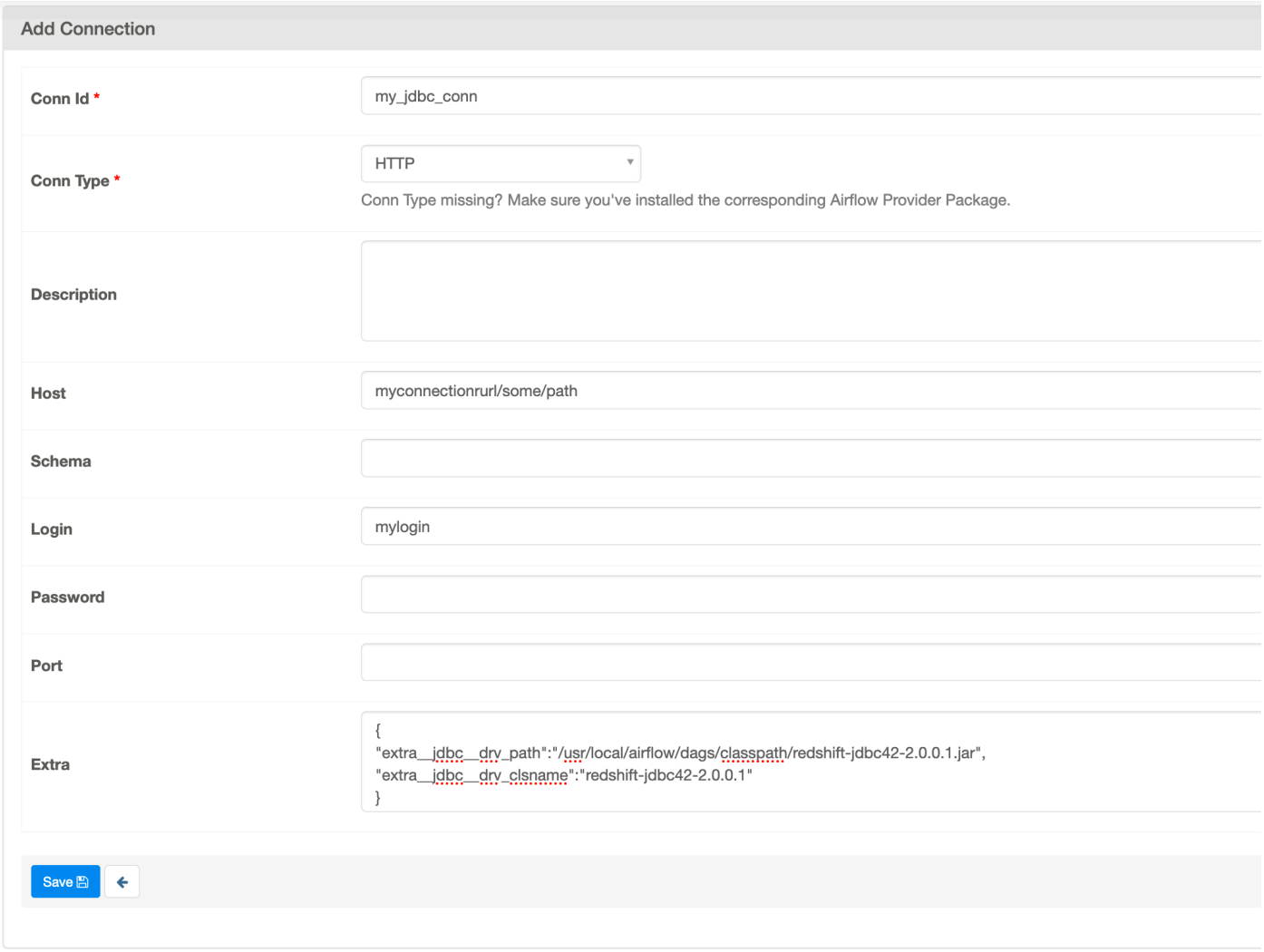
The following example shows how to use the **HTTP** connection template for a *Jdbc* connection type in Apache Airflow v2.0.2.

Apache Airflow v2

The following example shows the connection URI string generated by Apache Airflow for the example in this section.

```
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
```

The following example shows how to use the HTTP connection template for a *Jdbc* connection for Apache Airflow v2 in the Apache Airflow UI.



Configuring an Apache Airflow connection using a AWS Secrets Manager secret

AWS Secrets Manager is a supported alternative Apache Airflow backend on an Amazon Managed Workflows for Apache Airflow environment. This topic shows how to use AWS Secrets Manager to securely store secrets for Apache Airflow variables and an Apache Airflow connection on Amazon Managed Workflows for Apache Airflow.

Note

- You will be charged for the secrets you create. For more information on Secrets Manager pricing, see AWS Pricing.

- AWS Systems Manager Parameter Store is also supported as a secrets backend in Amazon MWAA. For more information, see Amazon Provider Package documentation.

Contents

- Step one: Provide Amazon MWAA with permission to access Secrets Manager secret keys
- Step two: Create the Secrets Manager backend as an Apache Airflow configuration option
- Step three: Generate an Apache Airflow AWS connection URI string
- Step four: Add the variables in Secrets Manager
- Step five: Add the connection in Secrets Manager
- Sample code
- Resources
- What's next?

Step one: Provide Amazon MWAA with permission to access Secrets Manager secret keys

The execution role for your Amazon MWAA environment needs read access to the secret key in AWS Secrets Manager. The following IAM policy allows read-write access using the AWS managed SecretsManagerReadWrite policy.

To attach the policy to your execution role

1. Open the Environments page on the Amazon MWAA console.
2. Choose an environment.
3. Choose your execution role on the **Permissions** pane.
4. Choose **Attach policies**.
5. Type `SecretsManagerReadWrite` in the **Filter policies** text field.
6. Choose **Attach policy**.

If you do not want to use an AWS managed permission policy, you can directly update your environment's execution role to allow any level of access to your Secrets Manager resources. For

example, the following policy statement grants read access to all secrets you create in a specific AWS Region in Secrets Manager.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetResourcePolicy",
                "secretsmanager:GetSecretValue",
                "secretsmanager:DescribeSecret",
                "secretsmanager:ListSecretVersionIds"
            ],
            "Resource": "arn:aws:secretsmanager:us-west-2:012345678910:secret:*"
        },
        {
            "Effect": "Allow",
            "Action": "secretsmanager:ListSecrets",
            "Resource": "*"
        }
    ]
}
```

Step two: Create the Secrets Manager backend as an Apache Airflow configuration option

The following section describes how to create an Apache Airflow configuration option on the Amazon MWAA console for the AWS Secrets Manager backend. If you're using a configuration setting of the same name in `airflow.cfg`, the configuration you create in the following steps will take precedence and override the configuration settings.

1. Open the Environments page on the Amazon MWAA console.
2. Choose an environment.
3. Choose **Edit**.
4. Choose **Next**.
5. Choose **Add custom configuration** in the **Airflow configuration options** pane. Add the following key-value pairs:

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

b. **`secrets.backend_kwargs`**: **`{"connections_prefix" : "airflow/connections", "variables_prefix" : "airflow/variables"}`** This configures Apache Airflow to look for connection strings and variables at `airflow/connections/*` and `airflow/variables/*` paths.

   You can use a lookup pattern to reduces the number of API calls Amazon MWAA makes to Secrets Manager on your behalf. If you do not specify a lookup pattern, Apache Airflow searches for all connections and variables in the configured backend. By specifying a pattern, you narrow the possible paths that Apache Airflow looks. This lowers your costs when using Secrets Manager with Amazon MWAA.

   To specify a lookup pattern, specify the `connections_lookup_pattern` and `variables_lookup_pattern` parameters. These parameters accept a RegEx string as input. For example, to look for secrets that start with `test`, enter the following for `secrets.backend_kwargs`:

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

   Note

   To use `connections_lookup_pattern` and `variables_lookup_pattern`, you must install `apache-airflow-providers-amazon` version 7.3.0 or higher. For more information on updating provder pacakges for to newer versions, see the section called “Specifying newer provider packages”.

6. Choose **Save**.

Step three: Generate an Apache Airflow AWS connection URI string

To create a connection string, use the "tab" key on your keyboard to indent the key-value pairs in the Connection object. We also recommend creating a variable for the `extra` object in your shell session. The following section walks you through the steps to generate an Apache Airflow connection URI string for an Amazon MWAA environment using Apache Airflow or a Python script.

Apache Airflow CLI

The following shell session uses your local Airflow CLI to generate a connection string. If you don't have the CLI installed, we recommend using the Python script.

1. Open a Python shell session:

```
python3
```

2. Enter the following command:

```
>>> import json
```

3. Enter the following command:

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

4. Create a variable in your shell session for the `extra` object. Substitute the sample values in *YOUR_EXECUTION_ROLE_ARN* with the execution role ARN, and the region in *YOUR_REGION* (such as `us-east-1`).

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

5. Create the connection object. Substitute the sample value in `myconn` with the name of the Apache Airflow connection.

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

6. Use the "tab" key on your keyboard to indent each of the following key-value pairs in your connection object. Substitute the sample values in *red*.

   a. Specify the AWS connection type:

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

b. Specify the Apache Airflow database option:

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

c. Specify the Apache Airflow UI URL on Amazon MWAA:

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

d. Specify the AWS access key ID (username) to login to Amazon MWAA:

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

e. Specify the AWS secret access key (password) to login to Amazon MWAA:

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

f. Specify the `extra` shell session variable:

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

g. Close the connection object.

```
... )
```

7. Print the connection URI string:

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

You should see the connection URI string in the response:

```
'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

The following Python script does not require the Apache Airflow CLI.

1. Copy the contents of the following code sample and save locally as `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 = 'YOUR_REGION'

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

2. Substitute the placeholders in *red*.
3. Run the following script to generate a connection string.

```
python3 mwaa_connection.py
```

Step four: Add the variables in Secrets Manager

The following section describes how to create the secret for a variable in Secrets Manager.

To create the secret

1. Open the AWS Secrets Manager console.
2. Choose **Store a new secret**.
3. Choose **Other type of secret**.
4. On the **Specify the key/value pairs to be stored in this secret** pane, choose **Plaintext**.
5. Add the variable value as **Plaintext** in the following format.

```
"YOUR_VARIABLE_VALUE"
```

For example, to specify an integer:

```
14
```

For example, to specify a string:

```
"mystring"
```

6. For **Encryption key**, choose an AWS KMS key option from the dropdown list.
7. Enter a name in the text field for **Secret name** in the following format.

   ```
   airflow/variables/YOUR_VARIABLE_NAME
   ```

   For example:

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

8. Choose **Next**.
9. On the **Configure secret** page, on the **Secret name and description** pane, do the following.

   a. For **Secret name**, provide a name for your secret.

   b. (Optional) For **Description**, provide a description for your secret.

   Choose **Next**.
10. On the **Configure rotation - optional** leave the default options and choose **Next**.
11. Repeat these steps in Secrets Manager for any additional variables you want to add.
12. On the **Review** page, review your secret, then choose **Store**.

Step five: Add the connection in Secrets Manager

The following section describes how to create the secret for your connection string URI in Secrets Manager.

To create the secret

1. Open the AWS Secrets Manager console.
2. Choose **Store a new secret**.
3. Choose **Other type of secret**.
4. On the **Specify the key/value pairs to be stored in this secret** pane, choose **Plaintext**.
5. Add the connection URI string as **Plaintext** in the following format.

   ```
   YOUR_CONNECTION_URI_STRING
   ```

   For example:

   ```
   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 parses each of the values in the connection string. You must **not** use single nor double quotes, or it will parse the connection as a single string.

6. For **Encryption key**, choose an AWS KMS key option from the dropdown list.
7. Enter a name in the text field for **Secret name** in the following format.

   ```
   airflow/connections/YOUR_CONNECTION_NAME
   ```

   For example:

   ```
   airflow/connections/myconn
   ```

8. Choose **Next**.
9. On the **Configure secret** page, on the **Secret name and description** pane, do the following.

   a. For **Secret name**, provide a name for your secret.

   b. (Optional) For **Description**, provide a description for your secret.

Choose **Next**.

10. On the **Configure rotation - optional** leave the default options and choose **Next**.
11. Repeat these steps in Secrets Manager for any additional variables you want to add.
12. On the **Review** page, review your secret, then choose **Store**.

Sample code

- Learn how to use the secret key for the Apache Airflow connection (`myconn`) on this page using the sample code at Using a secret key in AWS Secrets Manager for an Apache Airflow connection.
- Learn how to use the secret key for the Apache Airflow variable (`test-variable`) on this page using the sample code at Using a secret key in AWS Secrets Manager for an Apache Airflow variable.

Resources

- For more information about configuring Secrets Manager secrets using the console and the AWS CLI, see Create a secret in the *AWS Secrets Manager User Guide*.
- Use a Python script to migrate a large volume of Apache Airflow variables and connections to Secrets Manager in Move your Apache Airflow connections and variables to AWS Secrets Manager.

What's next?

- Learn how to generate a token to access the Apache Airflow UI in Accessing Apache Airflow.

Managing Amazon MWAA environments

The Amazon Managed Workflows for Apache Airflow console contains built-in options to configure private or public access to the Apache Airflow UI. It also contains built-in options to configure the environment size, when to scale workers, and Apache Airflow configuration options that allow you to override Apache Airflow configurations that are normally only accessible in `airflow.cfg`. This chapter describes how to use these configurations on the Amazon MWAA console.

Topics

- Configuring the Amazon MWAA environment class
- Configuring Amazon MWAA worker automatic scaling
- Configuring Amazon MWAA web server automatic scaling
- Using Apache Airflow configuration options on Amazon MWAA
- Upgrading the Apache Airflow version
- Using a startup script with Amazon MWAA

Configuring the Amazon MWAA environment class

The environment class you choose for your Amazon MWAA environment determines the size of the AWS-managed AWS Fargate containers where the Celery Executor runs, and the AWS-managed Amazon Aurora PostgreSQL metadata database where the Apache Airflow schedulers creates task instances. This topic describes each Amazon MWAA environment class, and how to update the environment class on the Amazon MWAA console.

Sections

- Environment capabilities
- Apache Airflow Schedulers

Environment capabilities

The following section contains the default concurrent Apache Airflow tasks, Random Access Memory (RAM), and the virtual centralized processing units (vCPUs) for each environment class. The concurrent tasks listed assume that task concurrency does not exceed the Apache Airflow *Worker* capacity in the environment.

In the following table, DAG capacity refers to DAG definitions, not executions, and assumes that your DAGs are dynamic in a single Python file and written with Apache Airflow best practices.

Task executions depend by how many are scheduled simultaneously, and assumes that the number of DAG runs set to start at the same time does not exceed the default `max_dagruns_per_loop_to_schedule`, as well as the size and number of workers as detailed in this topic.

mw1.micro

- Up to 25 DAG capacity
- 3 concurrent tasks (by default)
- Components:
  - Web server: 1 vCPU, 3GB RAM
  - Worker and scheduler: 1 vCPU, 3GB RAM
  - Database: 2 vCPU, 4GB RAM

> **Note**
>
> mw1.micro does not support auto-scaling.

mw1.small

- Up to 50 DAG capacity
- 5 concurrent tasks (by default)
- Components:
  - Web servers: 1 vCPU, 2GB RAM each
  - Workers: 1 vCPU, 2GB RAM each
  - Schedulers: 1 vCPU, 2GB RAM each
  - Database: 2 vCPU, 4GB RAM

mw1.medium

- Up to 250 DAG capacity
- 10 concurrent tasks (by default)

- Components:
  - Web servers: 1 vCPU 2GB RAM each
  - Workers: 2 vCPU 4GB RAM each
  - Schedulers: 2 vCPU 4GB RAM each
  - Database: 2 vCPU 8GB RAM

mw1.large

- Up to 1000 DAG capacity
- 20 concurrent tasks (by default)
- Components:
  - Web servers: 2 vCPU 4GB RAM each
  - Workers: 4 vCPU 8GB RAM each
  - Schedulers: 4 vCPU 8GB RAM each
  - Database: 2 vCPU 8GB RAM

mw1.xlarge

- Up to 2000 DAG capacity
- 40 concurrent tasks (by default)
- Components:
  - Web servers: 4 vCPU 12GB RAM each
  - Workers: 8 vCPU 24GB RAM each
  - Schedulers: 8 vCPU 24GB RAM each
  - Database: 4 vCPU 32GB RAM

mw1.2xlarge

- Up to 4000 DAG capacity
- 80 concurrent tasks (by default)
- Componenets:
  - Web servers: 8 vCPU 24GB RAM each

- Workers: 16 vCPU 48GB RAM each
- Schedulers: 16 vCPU 48GB RAM each
- Database: 8 vCPU 64GB RAM

You can use `celery.worker_autoscale` to increase tasks per worker. For more information, see the the section called “Example high performance use case”.

Apache Airflow Schedulers

The following section contains the Apache Airflow *scheduler* options available on the Amazon MWAA, and how the number of schedulers affects the number of *triggerers*.

In Apache Airflow, a triggerer manages tasks which it defers until certain conditions specified using a *trigger* have been met. In Amazon MWAA the triggerer runs alongside the scheduler on the same Fargate task. Increasing the scheduler count correspondingly increases the number of available triggerers, optimizing how the environment manages deferred tasks. This ensures efficient handling of tasks, promptly scheduling them to run when conditions are satisfied.

Apache Airflow v2

- **v2** - For environments larger than mw1.micro, accepts values from 2 to 5. Defaults to 2 for all environment sizes except mw1.micro, which defaults to 1.

Configuring Amazon MWAA worker automatic scaling

The auto scaling mechanism automatically increases the number of Apache Airflow workers in response to running and queued tasks on your Amazon Managed Workflows for Apache Airflow environment and disposes of extra workers when there are no more tasks queued or executing. This topic describes how you can configure auto scaling by specifying the maximum number of Apache Airflow workers that run on your environment using the Amazon MWAA console.

> **Note**
>
> Amazon MWAA uses Apache Airflow metrics to determine when additional Celery Executor workers are needed, and as required increases the number of Fargate workers up to the value specified by `max-workers`. As the additional workers complete work and work load

decreases, Amazon MWAA removes them, thus downscaling back to the value set by `min-workers`.
If workers pick up new tasks while downscaling, Amazon MWAA keeps the Fargate resource and does not remove the worker. For more information, see How Amazon MWAA auto scaling works.

Sections

- How worker scaling works
- Using the Amazon MWAA console
- Example high performance use case
- Troubleshooting tasks stuck in the running state
- What's next?

How worker scaling works

Amazon MWAA uses `RunningTasks` and `QueuedTasks` metrics, where *(tasks running + tasks queued) / (tasks per worker) = (required workers)*. If the required number of workers is greater than the current number of workers, Amazon MWAA will add Fargate worker containers to that value, up to the maximum value specified by `max-workers`.

As the workload decreases and the `RunningTasks` and `QueuedTasks` metric sum reduces, Amazon MWAA requests Fargate to scale down the workers for the environment. Any workers which still completing work remain protected during downscaling until they complete their work. Depending on the workload, tasks may be queued while workers downscale.

Using the Amazon MWAA console

You can choose the maximum number of workers that can run on your environment concurrently on the Amazon MWAA console. By default, you can specify a maximum value up to 25.

To configure the number of workers

1. Open the Environments page on the Amazon MWAA console.
2. Choose an environment.
3. Choose **Edit**.

4. Choose **Next**.
5. On the **Environment class** pane, enter a value in **Maximum worker count**.
6. Choose **Save**.

> **Note**
>
> It can take a few minutes before changes take effect on your environment.

Example high performance use case

The following section describes the type of configurations you can use to enable high performance and parallelism on an environment.

On-premise Apache Airflow

Typically, in an on-premise Apache Airflow platform, you would configure task parallelism, auto scaling, and concurrency settings in your `airflow.cfg` file:

- `core.parallelism` – The maximum number of task instances that can run simultaneously per scheduler.
- `core.dag_concurrency` – The maximum concurrency for DAGs (not workers).
- `celery.worker_autoscale` – The maximum and minimum number of tasks that can run concurrently on any worker.

For example, if `core.parallelism` was set to `100` and `core.dag_concurrency` was set to 7, you would still only be able to run a total of 14 tasks concurrently if you had 2 DAGs. Given, each DAG is set to run only seven tasks concurrently (in `core.dag_concurrency`), even though overall parallelism is set to `100` (in `core.parallelism`).

On an Amazon MWAA environment

On an Amazon MWAA environment, you can configure these settings directly on the Amazon MWAA console using Using Apache Airflow configuration options on Amazon MWAA, Configuring the Amazon MWAA environment class, and the **Maximum worker count** auto scaling mechanism. While `core.dag_concurrency` is not available in the drop down list as an **Apache Airflow**

configuration option on the Amazon MWAA console, you can add it as a custom Apache Airflow configuration option.

Let's say, when you created your environment, you chose the following settings:

1. The **mw1.small** environment class which controls the maximum number of concurrent tasks each worker can run by default and the vCPU of containers.
2. The default setting of `10` Workers in **Maximum worker count**.
3. An Apache Airflow configuration option for `celery.worker_autoscale` of `5,5` tasks per worker.

This means you can run 50 concurrent tasks in your environment. Any tasks beyond 50 will be queued, and wait for the running tasks to complete.

Run more concurrent tasks. You can modify your environment to run more tasks concurrently using the following configurations:

1. Increase the maximum number of concurrent tasks each worker can run by default and the vCPU of containers by choosing the `mw1.medium` (10 concurrent tasks by default) environment class.
2. Add `celery.worker_autoscale` as an Apache Airflow configuration option.
3. Increase the **Maximum worker count**. In this example, increasing maximum workers from `10` to `20` would double the number of concurrent tasks the environment can run.

Specify Minimum workers. You can also specify the minimum and maximum number of Apache Airflow *Workers* that run in your environment using the AWS Command Line Interface (AWS CLI). For example:

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

To learn more, see the update-environment command in the AWS CLI.

Troubleshooting tasks stuck in the running state

In rare cases, Apache Airflow may think there are tasks still running. To resolve this issue, you need to clear the stranded task in your Apache Airflow UI. For more information, see the Troubleshooting Amazon Managed Workflows for Apache Airflow troubleshooting topic.

What's next?

- Learn more about the best practices we recommend to tune the performance of your environment in Performance tuning for Apache Airflow on Amazon MWAA.

Configuring Amazon MWAA web server automatic scaling

For environments running Apache Airflow v2.2.2 and above, Amazon MWAA dynamically scales your web servers to handle fluctuating workloads, which in turn prevents performance issues during peak loads. By automatically scaling the number of web servers based on CPU utilization and active connection count, Amazon MWAA ensures that your Apache Airflow environment can seamlessly accommodate increased demand, whether from REST API requests, CLI usage, or more concurrent Apache Airflow user interface users.

Sections

- How web server scaling works
- Using the Amazon MWAA console

How web server scaling works

Amazon MWAA uses the container metric, `CPUUtilization`, and the load balancer metric, `ActiveConnectionCount`, to determine if scaling the web servers is required based on the amount of traffic. If `CPUUtilization` is higher than 70 or `ActiveConnectionCount` is higher than 15, Amazon MWAA will add additional Fargate web server containers up to the maximum value specified by `MaxWebservers`.

As traffic decreases and the `CPUUtilization` and `ActiveConnectionCount` values reduce, Amazon MWAA requests Fargate to scale down the web server containers for the environment to the minimum value set by `MinimumWebservers`.

Using the Amazon MWAA console

You can choose the number of web servers that can run on your environment concurrently on the Amazon MWAA console. By default, the minimum number of web servers is two, and the maximum number of web servers is five.

To configure the number of web servers

1. Open the Environments page on the Amazon MWAA console.
2. Choose an environment.
3. Choose **Edit**.
4. Choose **Next**.
5. On the **Environment class** pane, enter a value in **Maximum web server count**.
6. Next, enter a value in **Minimum web server count**.
7. Choose **Save**.

> **Note**
>
> It can take a few minutes before changes take effect on your environment.

Using Apache Airflow configuration options on Amazon MWAA

Apache Airflow configuration options can be attached to your Amazon Managed Workflows for Apache Airflow environment as environment variables. You can choose from the suggested dropdown list, or specify custom configuration options for your Apache Airflow version on the Amazon MWAA console. This topic describes the Apache Airflow configuration options available, and how to use these options to override Apache Airflow configuration settings on your environment.

Contents

- Prerequisites
- How it works
- Using configuration options to load plugins in Apache Airflow v2
- Configuration options overview
  - Apache Airflow configuration options
  - Apache Airflow reference
  - Using the Amazon MWAA console
- Configuration reference
  - Email configurations

- Task configurations
- Scheduler configurations
- Worker configurations
- Web server configurations
- Triggerer configurations
- Examples and sample code
  - Example DAG
  - Example email notification settings
- What's next?

Prerequisites

You'll need the following before you can complete the steps on this page.

- **Permissions** — Your AWS account must have been granted access by your administrator to the AmazonMWAAFullConsoleAccess access control policy for your environment. In addition, your Amazon MWAA environment must be permitted by your execution role to access the AWS resources used by your environment.
- **Access** — If you require access to public repositories to install dependencies directly on the web server, your environment must be configured with **public network** web server access. For more information, see the section called "Apache Airflow access modes".
- **Amazon S3 configuration** — The Amazon S3 bucket used to store your DAGs, custom plugins in `plugins.zip`, and Python dependencies in `requirements.txt` must be configured with *Public Access Blocked* and *Versioning Enabled*.

How it works

When you create an environment, Amazon MWAA attaches the configuration settings you specify on the Amazon MWAA console in **Airflow configuration options** as environment variables to the AWS Fargate container for your environment. If you're using a setting of the same name in `airflow.cfg`, the options you specify on the Amazon MWAA console override the values in `airflow.cfg`.

While we don't expose the `airflow.cfg` in the Apache Airflow UI of an Amazon MWAA environment by default, you can change the Apache Airflow configuration options directly on

the Amazon MWAA console, including setting `webserver.expose_config` to expose the configurations.

Using configuration options to load plugins in Apache Airflow v2

By default in Apache Airflow v2, plugins are configured to be "lazily" loaded using the `core.lazy_load_plugins : True` setting. If you're using custom plugins in Apache Airflow v2, you must add `core.lazy_load_plugins : False` as an Apache Airflow configuration option to load plugins at the start of each Airflow process to override the default setting.

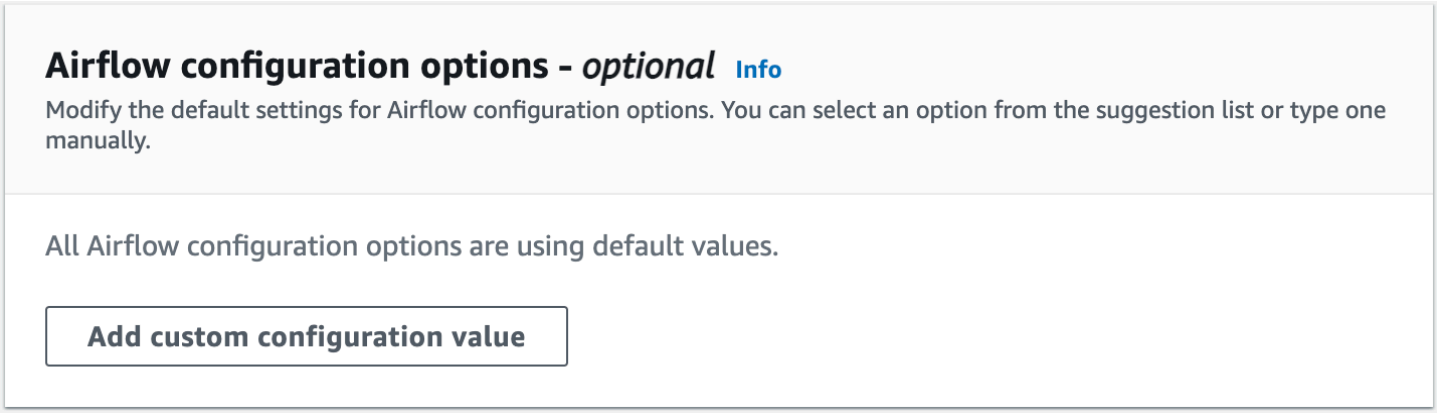
Configuration options overview

When you add a configuration on the Amazon MWAA console, Amazon MWAA writes the configuration as an environment variable.

- **Listed options**. You can choose from one of the configuration settings available for your Apache Airflow version in the dropdown list. For example, `dag_concurrency : 16`. The configuration setting is translated to your environment's Fargate container as `AIRFLOW__CORE__DAG_CONCURRENCY : 16`
- **Custom options**. You can also specify Airflow configuration options that are not listed for your Apache Airflow version in the dropdown list. For example, `foo.user : YOUR_USER_NAME`. The configuration setting is translated to your environment's Fargate container as `AIRFLOW__FOO__USER : YOUR_USER_NAME`

Apache Airflow configuration options

The following image shows where you can customize the **Apache Airflow configuration options** on the Amazon MWAA console.



Apache Airflow reference

For a list of configuration options supported by Apache Airflow, see Configuration Reference in the *Apache Airflow reference guide*. To view the options for the version of Apache Airflow you are running on Amazon MWAA, select the version from the drop down list.

Using the Amazon MWAA console

The following procedure walks you through the steps of adding an Airflow configuration option to your environment.

1. Open the Environments page on the Amazon MWAA console.
2. Choose an environment.
3. Choose **Edit**.
4. Choose **Next**.
5. Choose **Add custom configuration** in the **Airflow configuration options** pane.
6. Choose a configuration from the dropdown list and enter a value, or type a custom configuration and enter a value.
7. Choose **Add custom configuration** for each configuration you want to add.
8. Choose **Save**.

Configuration reference

The following section contains the list of available Apache Airflow configurations in the dropdown list on the Amazon MWAA console.

Email configurations

The following list shows the Airflow email notification configuration options available on Amazon MWAA.

We recommend using port 587 for SMTP traffic. By default, AWS blocks outbound SMTP traffic on port 25 of all Amazon EC2 instances. If you want to send outbound traffic on port 25, you can request for this restriction to be removed.

Apache Airflow v2

Airflow version	Airflow configuration option	Description	Example value
v2	email.email_backend	The Apache Airflow utility used for email notifications in email_backend.	airflow.utils.email.send_email_smtp
v2	smtp.smtp_host	The name of the outbound server used for the email address in smtp_host.	localhost
v2	smtp.smtp_starttls	Transport Layer Security (TLS) is used to encrypt the email over the Internet in smtp_starttls.	False
v2	smtp.smtp_ssl	Secure Sockets Layer (SSL) is used to connect the server and email client in smtp_ssl.	True
v2	smtp.smtp_port	The Transmission Control Protocol (TCP) port designated to the server in smtp_port.	587
v2	smtp.smtp_mail_from	The outbound email address in smtp_mail_from.	myemail@domain.com

Task configurations

The following list shows the configurations available in the dropdown list for Airflow tasks on Amazon MWAA.

Apache Airflow v2

Airflow version	Airflow configuration option	Description	Example value
v2	core.default_task_retries	The number of times to retry an Apache Airflow task in default_task_retries.	3
v2	core.parallelism	The maximum number of task instances that can run simultaneously across the entire environment in parallel (parallelism).	40

Scheduler configurations

The following list shows the Apache Airflow scheduler configurations available in the dropdown list on Amazon MWAA.

Apache Airflow v2

Airflow version	Airflow configuration option	Description	Example value
v2	scheduler.catchup_by_default	Tells the scheduler to create a DAG run to "catch up"	False

Airflow version	Airflow configuration option	Description	Example value
		to the specific time interval in catchup_by_default.	
v2	scheduler.scheduler_zombie_task_threshold	Tells the scheduler whether to mark the task instance as failed and reschedule the task in scheduler_zombie_task_threshold.	300

Worker configurations

The following list shows the Airflow worker configurations available in the dropdown list on Amazon MWAA.

Apache Airflow v2

Airflow version	Airflow configuration option	Description	Example value
v2	celery.worker_autoscale	The maximum and minimum number of tasks that can run concurrently on any worker using the Celery Executor in worker_autoscale. Value must be comma-separated in the following order: max_concu	16,12

Airflow version	Airflow configuration option	Description	Example value
		`rrency,mi n_concurrency` .	

Web server configurations

The following list shows the Airflow web server configurations available in the dropdown list on Amazon MWAA.

Apache Airflow v2

Airflow version	Airflow configuration option	Description	Example value
v2	webserver.default_ui_timezone	The default Apache Airflow UI datetime setting in default_ui_timezone. **Note** Setting the `default_ui_timezone` option does not change the time zone in which your DAGs are scheduled to run. To change the time zone for your DAGs,	America/New_York

Airflow version	Airflow configuration option	Description	Example value
		you can use a custom plugin. For more information, see the section called "Changing a DAG's timezone".	

Triggerer configurations

The following list shows the Apache Airflow triggerer configurations available on Amazon MWAA.

Apache Airflow v2

Airflow version	Airflow configuration option	Description	Example value
v2.7	mwaa.triggerer_enabled	Used for activating and deactivating the triggerer on Amazon MWAA. By default, this value is set to `True`. If set to `False`, Amazon MWAA will not start any triggerer processes on schedulers.	True

Airflow version	Airflow configuration option	Description	Example value
v2.7	triggerer.default_capacity	Defines the number triggers each triggerer can run in parallel. On Amazon MWAA, this capacity is set per each triggerer and per each scheduler as both components run alongside each other. The default per scheduler is set to `60`, `125`, `250`, `500`, and `1000` for small, medium and large, xlarge, and 2xlarge instances, respectively.	125

Examples and sample code

Example DAG

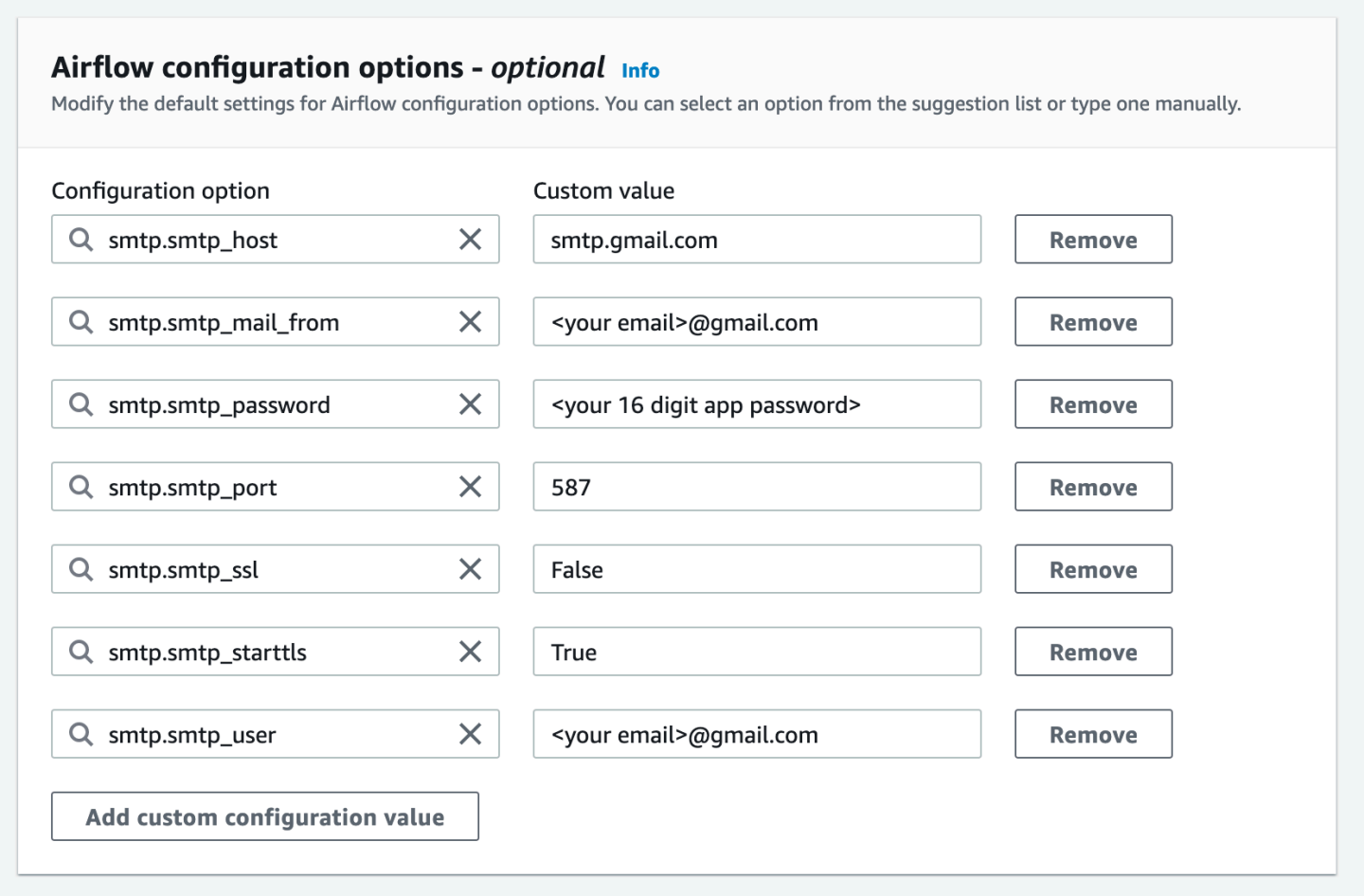
You can use the following DAG to print your `email_backend` Apache Airflow configuration options. To run in response to Amazon MWAA events, copy the code to your environment's DAGs folder on your Amazon S3 storage bucket.

```
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()
```

Example email notification settings

The following Apache Airflow configuration options can be used for a Gmail.com email account using an app password. For more information, see Sign in using app passwords in the *Gmail Help reference guide*.



What's next?

- Learn how to upload your DAG folder to your Amazon S3 bucket in Adding or updating DAGs.

Upgrading the Apache Airflow version

Amazon MWAA supports minor version upgrades. This means you can upgrade your environment from version x**.4.**z to x**.5.**z. To perform a major version upgrade, for example from version **1.**y**.**z to **2.**y**.**z, you must create a new environment and migrate your resources. For more information on upgrading to a new major version of Apache Airflow, see Migrating to a new Amazon MWAA environment in the *Amazon MWAA Migration Guide*.

During the upgrade process, Amazon MWAA captures a snapshot of your environment metadata, upgrades the workers, schedulers, the web server to the new Apache Airflow version, and finally restores the metadata database using the snapshot.

> **Note**
>
> You cannot downgrade the Apache Airflow version for your environment.

Before you upgrade, make sure that your DAGs and other workflow resources are compatible with the new Apache Airflow version you are upgrading to. If you use a `requirements.txt` to manage dependencies, you must also ensure the dependencies you specify in your requirements are compatible with the new version.

Topics

- Upgrade your workflow resources
- Specify the new version

Upgrade your workflow resources

Whenever you're changing Apache Airflow versions, ensure that you reference the correct `--constraint` URL in your `requirements.txt`.

> **Warning**
>
> Specifying requirements that are incompatible with your target Apache Airflow version during an upgrade might result in a lengthy rollback process to the previous version of Apache Airflow with the previous requirements version.

To migrate your workflow resources

1. Create a fork of the aws-mwaa-local-runner repository, and clone a copy of the Amazon MWAA local runner.
2. Checkout to the branch of the aws-mwaa-local-runner repository that matches the version you are upgrading to.
3. Use the Amazon MWAA local runner CLI tool to build the Docker image and run Apache Airflow locally. For more information, see the local runner README in the GitHub repository.
4. To update your `requirements.txt`, follow the best practices we recommend in Managing Python dependencies, in the *Amazon MWAA User Guide*.

5. (Optional) To speed up the upgrade process, clean up the environment's metadata database. Environments with a large amount of metadata can take significantly longer to upgrade.
6. After you have successfully tested your workflow resources, copy your DAGs, `requirements.txt`, and plugins to your environment's Amazon S3 bucket.

You are now ready to edit the environment, specify a new Apache Airflow version, and start the update procedure.

Specify the new version

After you have completed updating your workflow resources to ensure compatibility with the new Apache Airflow version, do the following to edit environment details and specify the version of Apache Airflow that you want to upgrade to.

> **Note**
>
> When you perform an upgrade, all tasks currently running on the environment are terminated during the procedure. The update procedure can take up to two hours, during which time your environment will be unavailable.

To specify a new version using the console

1. Open the Environments page on the Amazon MWAA console.
2. From the **Environments** list, choose the environment that you want to upgrade.
3. On the environment page, choose **Edit** to edit the environment.
4. In the **Environment details** section, for **Airflow version**, choose the new Apache Airflow version number that you want to upgrade the environment to from the dropdown list.
5. Choose **Next** until you are on the **Review and save** page.
6. On the **Review and save** page, review your changes, then choose **Save**.

When you apply changes, your environment begins the upgrade procedure. During this period, the status of your environment indicates what actions Amazon MWAA is taking, and whether the procedure is successful.

In a successful upgrade scenario, the status will show `UPDATING`, then `CREATING_SNAPSHOT` as Amazon MWAA captures a backup of your metadata. Finally, the status will return first to `UPDATING`, then to `AVAILABLE` when the procedure is done.

If the environment fails to upgrade, your environment status will show `ROLLING_BACK`. If the rollback is successful, the status will first show `UPDATE_FAILED`, indicating that the update failed but the environment is available. If the rollback fails, the status will show `UNAVAILABLE`, indicating that you cannot access the environment.

Using a startup script with Amazon MWAA

A startup script is a shell (`.sh`) script that you host in your environment's Amazon S3 bucket similar to your DAGs, requirements, and plugins. Amazon MWAA runs this script during startup on every individual Apache Airflow component (worker, scheduler, and web server) before installing requirements and initializing the Apache Airflow process. Use a startup script to do the following:

- **Install runtimes** – Install Linux runtimes required by your workflows and connections.
- **Configure environment variables** – Set environment variables for each Apache Airflow component. Overwrite common variables such as `PATH`, `PYTHONPATH`, and `LD_LIBRARY_PATH`.
- **Manage keys and tokens** – Pass access tokens for custom repositories to `requirements.txt` and configure security keys.

The following topics describe how to configure a startup script to install Linux runtimes, set environment variables, and troubleshoot related issues using CloudWatch Logs.

Topics

- Configure a startup script
- Install Linux runtimes using a startup script
- Set environment variables using a startup script

Configure a startup script

To use a startup script with your existing Amazon MWAA environment, upload a `.sh` file to your environment's Amazon S3 bucket. Then, to associate the script with the environment, specify the following in your environment details:

- **The Amazon S3 URL path to the script** – The relative path to the script hosted in your bucket, for example, `s3://mwaa-environment/startup.sh`
- **The Amazon S3 version ID of the script** – The version of the startup shell script in your Amazon S3 bucket. You must specify the version ID that Amazon S3 assigns to the file every time you update the script. Version IDs are Unicode, UTF-8 encoded, URL-ready, opaque strings that are no more than 1,024 bytes long, for example, `3sL4kqtJlcpXroDTDmJ+rmSpXd3dIbrHY+MTRCxf3vjVBH40Nr8X8gdRQBpUMLUo`.

To complete the steps in this section, use the following sample script. The script outputs the value assigned to `MWAA_AIRFLOW_COMPONENT`. This environment variable identifies each Apache Airflow component that the script runs on.

Copy the code and save it locally as `startup.sh`.

```
#!/bin/sh

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

Next, upload the script to your Amazon S3 bucket.

AWS Management Console

To upload a shell script (console)

1. Sign in to the AWS Management Console and open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. From the **Buckets** list, choose the name of the bucket associated with your environment.
3. On the **Objects** tab, choose **Upload**.
4. On the **Upload** page, drag and drop the shell script you created.
5. Choose **Upload**.

The script appears in the list of **Objects**. Amazon S3 creates a new version ID for the file. If you update the script and upload it again using the same file name, a new version ID is assigned to the file.

AWS CLI

To create and upload a shell script (CLI)

1. Open a new command prompt, and run the Amazon S3 `ls` command to list and identify the bucket associated with your environment.

```
$ aws s3 ls
```

2. Navigate to the folder where you saved the shell script. Use `cp` in a new prompt window to upload the script to your bucket. Replace *your-s3-bucket* with your information.

```
$ aws s3 cp startup.sh s3://your-s3-bucket/startup.sh
```

If successful, Amazon S3 outputs the URL path to the object:

```
upload: ./startup.sh to s3://your-s3-bucket/startup.sh
```

3. Use the following command to retrieve the latest version ID for the script.

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

```
BbdVMmBRjtestta1EsVnbybZp1Wqh1J4
```

You specify this version ID when you associate the script with an environment.

Now, associate the script with your environment.

AWS Management Console

To associate the script with an environment (console)

1. Open the Environments page on the Amazon MWAA console.
2. Select the row for the environment you want to update, then choose **Edit**.
3. On the **Specify details** page, for **Startup script file - *optional***, enter the Amazon S3 URL for the script, for example: `s3://your-mwaa-bucket/startup-sh..`

4. Choose the latest version from the drop down list, or **Browse S3** to find the script.
5. Choose **Next**, then proceed to the **Review and save** page.
6. Review changes, then choose **Save**.

Environment updates can take between 10 to 30 minutes. Amazon MWAA runs the startup script as each component in your environment restarts.

AWS CLI

To associate the script with an environment (CLI)

- Open a command prompt and use `update-environment` to specify the Amazon S3 URL and version ID for the script.

```
$ aws mwaa update-environment \
    --name your-mwaa-environment \
    --startup-script-s3-path startup.sh \
    --startup-script-s3-object-version BbdVMmBRjteststa1EsVnbybZp1Wqh1J4
```

If successful, Amazon MWAA returns the Amazon Resource Name (ARN) for the environment:

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

Environment update can take between 10 to 30 minutes. Amazon MWAA runs the startup script as each component in your environment restarts.

Finally, retrieve log events to verify that the script is working as expected. When you activate logging for an each Apache Airflow component, Amazon MWAA creates a new log group and log stream. For more information, see Apache Airflow log types.

AWS Management Console

To check the Apache Airflow log stream (console)

1. Open the Environments page on the Amazon MWAA console.
2. Choose your environment.

3. In the **Monitoring** pane, choose the log group for which you want to view logs, for example, **Airflow scheduler log group** .
4. In the CloudWatch console, from the **Log streams** list, choose a stream with the following prefix: `startup_script_exection_ip`.
5. On the **Log events** pane, you will see the output of the command printing the value for `MWAA_AIRFLOW_COMPONENT`. For example, for scheduler logs, you will the following:

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

You can repeat the previous steps to view worker and web server logs.

Install Linux runtimes using a startup script

Use a startup script to update the operating system of an Apache Airflow component, and install additional runtime libraries to use with your workflows. For example, the following script runs `yum update` to update the operating system.

When running `yum update` in a startup script, you must exclude Python using `--exclude=python*` as shown in the example. For your environment to run, Amazon MWAA installs a specific version of Python compatible with your environment. Therefore, you can't update the environment's Python version using a startup script.

```
#!/bin/sh

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

To install runtimes on specific Apache Airflow component, use `MWAA_AIRFLOW_COMPONENT` and `if` and `fi` conditional statements. This example runs a single command to install the `libaio` library on the scheduler and worker, but not on the web server.

Important

- If you have configured a private web server, you must either use the following condition or provide all installation files locally in order to avoid installation timeouts.
- Use sudo to run operations that require administrative privileges.

```
#!/bin/sh

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

You can use a startup script to check the Python version.

```
#!/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 does not support overriding the default Python version, as this may lead to incompatibilities with the installed Apache Airflow libraries.

Set environment variables using a startup script

Use startup scripts to set environment variables and modify Apache Airflow configurations. The following defines a new variable, `ENVIRONMENT_STAGE`. You can reference this variable in a DAG or in your custom modules.

```
#!/bin/sh

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

Use startup scripts to overwrite common Apache Airflow or system variables. For example, you set `LD_LIBRARY_PATH` to instruct Python to look for binaries in the path you specify. This lets you provide custom binaries for your workflows using plugins:

```
#!/bin/sh

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

Reserved environment variables

Amazon MWAA reserves a set of critical environment variables. If you overwrite a *reserved* variable, Amazon MWAA restores it to its default. The following lists the reserved variables:

- `MWAA__AIRFLOW__COMPONENT` – Used to identify the Apache Airflow component with one of the following values: `scheduler`, `worker`, or `webserver`.
- `AIRFLOW__WEBSERVER__SECRET_KEY` – The secret key used for securely signing session cookies in the Apache Airflow web server.
- `AIRFLOW__CORE__FERNET_KEY` – The key used for encryption and decryption of sensitive data stored in the metadata database, for example, connection passwords.
- `AIRFLOW_HOME` – The path to the Apache Airflow home directory where configuration files and DAG files are stored locally.
- `AIRFLOW__CELERY__BROKER_URL` – The URL of the message broker used for communication between the Apache Airflow scheduler and the Celery worker nodes.
- `AIRFLOW__CELERY__RESULT_BACKEND` – The URL of the database used to store the results of Celery tasks.
- `AIRFLOW__CORE__EXECUTOR` – The executor class that Apache Airflow should use. In Amazon MWAA this is a `CeleryExecutor`
- `AIRFLOW__CORE__LOAD_EXAMPLES` – Used to activate, or deactivate, the loading of example DAGs.
- `AIRFLOW__METRICS__METRICS_BLOCK_LIST` – Used to manage which Apache Airflow metrics are emitted and captured by Amazon MWAA in CloudWatch.
- `SQL_ALCHEMY_CONN` – The connection string for the RDS for PostgreSQL database used to store Apache Airflow metadata in Amazon MWAA.
- `AIRFLOW__CORE__SQL_ALCHEMY_CONN` – Used for the same purpose as `SQL_ALCHEMY_CONN`, but following the new Apache Airflow naming convention.
- `AIRFLOW__CELERY__DEFAULT_QUEUE` – The default queue for Celery tasks in Apache Airflow.
- `AIRFLOW__OPERATORS__DEFAULT_QUEUE` – The default queue for tasks using specific Apache Airflow operators.

- AIRFLOW_VERSION – The Apache Airflow version installed in the Amazon MWAA environment.
- AIRFLOW_CONN_AWS_DEFAULT – The default AWS credentials used to integrate with other AWS services in.
- AWS_DEFAULT_REGION – Sets the default AWS Region used with default credentials to integrate with other AWS services.
- AWS_REGION – If defined, this environment variable overrides the values in the environment variable AWS_DEFAULT_REGION and the profile setting region.
- PYTHONUNBUFFERED – Used to send stdout and stderr streams to container logs.
- AIRFLOW__METRICS__STATSD_ALLOW_LIST – Used to configure an allow list of comma-separated prefixes to send the metrics that start with the elements of the list.
- AIRFLOW__METRICS__STATSD_ON – Activates sending metrics to StatsD.
- AIRFLOW__METRICS__STATSD_HOST – Used to connect to the StatSD daemon.
- AIRFLOW__METRICS__STATSD_PORT – Used to connect to the StatSD daemon.
- AIRFLOW__METRICS__STATSD_PREFIX – Used to connect to the StatSD daemon.
- AIRFLOW__CELERY__WORKER_AUTOSCALE – Sets the maximum and minimum concurrency.
- AIRFLOW__CORE__DAG_CONCURRENCY – Sets the number of task instances that can run concurrently by the scheduler in one DAG.
- AIRFLOW__CORE__MAX_ACTIVE_TASKS_PER_DAG – Sets the maximum number of active tasks per DAG.
- AIRFLOW__CORE__PARALLELISM – Defines the maximum number of task instances that can simultaneously.
- AIRFLOW__SCHEDULER__PARSING_PROCESSES – Sets the maximum number of processes parsed by the scheduler to schedule DAGs.
- AIRFLOW__CELERY_BROKER_TRANSPORT_OPTIONS__VISIBILITY_TIMEOUT – Defines the number of seconds a worker waits to acknowledge the task before the message is redelivered to another worker.
- AIRFLOW__CELERY_BROKER_TRANSPORT_OPTIONS__REGION – Sets the AWS Region for the underlying Celery transport.
- AIRFLOW__CELERY_BROKER_TRANSPORT_OPTIONS__PREDEFINED_QUEUES – Sets the queue for the underlying Celery transport.
- AIRFLOW_SCHEDULER_ALLOWED_RUN_ID_PATTERN – Used to verify the validity of your input for the run_id parameter when triggering a DAG.

- AIRFLOW__WEBSERVER__BASE_URL – The URL of the web server used to host the Apache Airflow UI.

Unreserved environment variables

You can use a startup script to overwrite *unreserved* environment variables. The following lists some of these common variables:

- PATH – Specifies a list of directories where the operating system searches for executable files and scripts. When a command runs in the command line, the system checks the directories in PATH in order to find and execute the command. When you create custom operators or tasks in Apache Airflow, you might need to rely on external scripts or executables. If the directories containing these files are not in the specified in the PATH variable, the tasks fail to run when the system is unable to locate them. By adding the appropriate directories to PATH, Apache Airflow tasks can find and run the required executables.
- PYTHONPATH – Used by the Python interpreter to determine which directories to search for imported modules and packages. It is a list of directories that you can add to the default search path. This lets the interpreter find and load Python libraries not included in the standard library, or installed in system directories. Use this variable to add your modules and custom Python packages and use them with your DAGs.
- LD_LIBRARY_PATH – An environment variable used by the dynamic linker and loader in Linux to find and load shared libraries. It specifies a list of directories containing shared libraries, which are searched before the default system library directories. Use this variable to specify your custom binaries.
- CLASSPATH – Used by the Java Runtime Environment (JRE) and Java Development Kit (JDK) to locate and load Java classes, libraries, and resources at runtime. It is a list of directories, JAR files, and ZIP archives that contain compiled Java code.

Working with DAGs on Amazon MWAA

To run Directed Acyclic Graphs (DAGs) on an Amazon Managed Workflows for Apache Airflow environment, you copy your files to the Amazon S3 storage bucket attached to your environment, then let Amazon MWAA know where your DAGs and supporting files are located on the Amazon MWAA console. Amazon MWAA takes care of synchronizing the DAGs among workers, schedulers, and the web server. This guide describes how to add or update your DAGs, and install custom plugins and Python dependencies on an Amazon MWAA environment.

Topics

- Amazon S3 bucket overview
- Adding or updating DAGs
- Installing custom plugins
- Installing Python dependencies
- Deleting files on Amazon S3

Amazon S3 bucket overview

An Amazon S3 bucket for an Amazon MWAA environment must have *Public Access Blocked*. By default, all Amazon S3 resources—buckets, objects, and related sub-resources (for example, lifecycle configuration)—are private.

- Only the resource owner, the AWS account that created the bucket, can access the resource. The resource owner (for example, your administrator) can grant access permissions to others by writing an access control policy.
- The access policy you set up must have permission to add DAGs, custom plugins in `plugins.zip`, and Python dependencies in `requirements.txt` to your Amazon S3 bucket. For an example policy that contains the required permissions, see AmazonMWAAFullConsoleAccess.

An Amazon S3 bucket for an Amazon MWAA environment must have *Versioning Enabled*. When Amazon S3 bucket versioning is enabled, anytime a new version is created, a new copy is created.

- Versioning is enabled for the custom plugins in a `plugins.zip`, and Python dependencies in a `requirements.txt` on your Amazon S3 bucket.

- You must specify the version of a `plugins.zip`, and `requirements.txt` on the Amazon MWAA console each time these files are updated on your Amazon S3 bucket.

Adding or updating DAGs

Directed Acyclic Graphs (DAGs) are defined within a Python file that defines the DAG's structure as code. You can use the AWS CLI, or the Amazon S3 console to upload DAGs to your environment. This topic describes the steps to add or update Apache Airflow DAGs on your Amazon Managed Workflows for Apache Airflow environment using the `dags` folder in your Amazon S3 bucket.

Sections

- Prerequisites
- How it works
- What's changed in v2
- Testing DAGs using the Amazon MWAA CLI utility
- Uploading DAG code to Amazon S3
- Specifying the path to your DAGs folder on the Amazon MWAA console (the first time)
- Viewing changes on your Apache Airflow UI
- What's next?

Prerequisites

You'll need the following before you can complete the steps on this page.

- **Permissions** — Your AWS account must have been granted access by your administrator to the AmazonMWAAFullConsoleAccess access control policy for your environment. In addition, your Amazon MWAA environment must be permitted by your execution role to access the AWS resources used by your environment.
- **Access** — If you require access to public repositories to install dependencies directly on the web server, your environment must be configured with **public network** web server access. For more information, see the section called “Apache Airflow access modes”.
- **Amazon S3 configuration** — The Amazon S3 bucket used to store your DAGs, custom plugins in `plugins.zip`, and Python dependencies in `requirements.txt` must be configured with *Public Access Blocked* and *Versioning Enabled*.

How it works

A Directed Acyclic Graph (DAG) is defined within a single Python file that defines the DAG's structure as code. It consists of the following:

- A DAG definition.
- Operators that describe how to run the DAG and the tasks to run.
- Operator relationships that describe the order in which to run the tasks.

To run an Apache Airflow platform on an Amazon MWAA environment, you need to copy your DAG definition to the `dags` folder in your storage bucket. For example, the DAG folder in your storage bucket may look like this:

Example DAG folder

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

Amazon MWAA automatically syncs new and changed objects from your Amazon S3 bucket to Amazon MWAA scheduler and worker containers' `/usr/local/airflow/dags` folder every 30 seconds, preserving the Amazon S3 source's file hierarchy, regardless of file type. The time that new DAGs take to appear in your Apache Airflow UI is controlled by `scheduler.dag_dir_list_interval`. Changes to existing DAGs will be picked up on the next DAG processing loop.

> **Note**
>
> You do not need to include the `airflow.cfg` configuration file in your DAG folder. You can override the default Apache Airflow configurations from the Amazon MWAA console. For more information, see Using Apache Airflow configuration options on Amazon MWAA.

What's changed in v2

- **New: Operators, Hooks, and Executors**. The import statements in your DAGs, and the custom plugins you specify in a `plugins.zip` on Amazon MWAA have changed between Apache Airflow v1 and Apache Airflow v2. For example, `from`

`airflow.contrib.hooks.aws_hook import AwsHook` in Apache Airflow v1 has changed to `from airflow.providers.amazon.aws.hooks.base_aws import AwsBaseHook` in Apache Airflow v2. To learn more, see Python API Reference in the *Apache Airflow reference guide*.

Testing DAGs using the Amazon MWAA CLI utility

- The command line interface (CLI) utility replicates an Amazon Managed Workflows for Apache Airflow environment locally.
- The CLI builds a Docker container image locally that's similar to an Amazon MWAA production image. This allows you to run a local Apache Airflow environment to develop and test DAGs, custom plugins, and dependencies before deploying to Amazon MWAA.
- To run the CLI, see the aws-mwaa-local-runner on GitHub.

Uploading DAG code to Amazon S3

You can use the Amazon S3 console or the AWS Command Line Interface (AWS CLI) to upload DAG code to your Amazon S3 bucket. The following steps assume you are uploading code (`.py`) to a folder named `dags` in your Amazon S3 bucket.

Using the AWS CLI

The AWS Command Line Interface (AWS CLI) is an open source tool that enables you to interact with AWS services using commands in your command-line shell. To complete the steps on this page, you need the following:

- AWS CLI – Install version 2.
- AWS CLI – Quick configuration with `aws configure`.

To upload using the AWS CLI

1. Use the following command to list all of your Amazon S3 buckets.

```
aws s3 ls
```

2. Use the following command to list the files and folders in the Amazon S3 bucket for your environment.

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

3. The following command uploads a `dag_def.py` file to a `dags` folder.

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

If a folder named `dags` does not already exist on your Amazon S3 bucket, this command creates the `dags` folder and uploads the file named `dag_def.py` to the new folder.

Using the Amazon S3 console

The Amazon S3 console is a web-based user interface that allows you to create and manage the resources in your Amazon S3 bucket. The following steps assume you have a DAGs folder named `dags`.

To upload using the Amazon S3 console

1. Open the Environments page on the Amazon MWAA console.
2. Choose an environment.
3. Select the **S3 bucket** link in the **DAG code in S3** pane to open your storage bucket on the Amazon S3 console.
4. Choose the `dags` folder.
5. Choose **Upload**.
6. Choose **Add file**.
7. Select the local copy of your `dag_def.py`, choose **Upload**.

Specifying the path to your DAGs folder on the Amazon MWAA console (the first time)

The following steps assume you are specifying the path to a folder on your Amazon S3 bucket named `dags`.

1. Open the Environments page on the Amazon MWAA console.
2. Choose the environment where you want to run DAGs.
3. Choose **Edit**.

4. On the **DAG code in Amazon S3** pane, choose **Browse S3** next to the **DAG folder** field.
5. Select your `dags` folder.
6. Choose **Choose**.
7. Choose **Next**, **Update environment**.

Viewing changes on your Apache Airflow UI

Logging into Apache Airflow

You need Apache Airflow UI access policy: AmazonMWAAWebServerAccess permissions for your AWS account in AWS Identity and Access Management (IAM) to view your Apache Airflow UI.

To access your Apache Airflow UI

1. Open the Environments page on the Amazon MWAA console.
2. Choose an environment.
3. Choose **Open Airflow UI**.

What's next?

- Test your DAGs, custom plugins, and Python dependencies locally using the aws-mwaa-local-runner on GitHub.

Installing custom plugins

Amazon Managed Workflows for Apache Airflow supports Apache Airflow's built-in plugin manager, allowing you to use custom Apache Airflow operators, hooks, sensors, or interfaces. This page describes the steps to install Apache Airflow custom plugins on your Amazon MWAA environment using a `plugins.zip` file.

Contents

- Prerequisites
- How it works
- When to use the plugins
- Custom plugins overview

- Custom plugins directory and size limits
- Examples of custom plugins
  - Example using a flat directory structure in plugins.zip
  - Example using a nested directory structure in plugins.zip
- Creating a plugins.zip file
  - Step one: Test custom plugins using the Amazon MWAA CLI utility
  - Step two: Create the plugins.zip file
- Uploading plugins.zip to Amazon S3
  - Using the AWS CLI
  - Using the Amazon S3 console
- Installing custom plugins on your environment
  - Specifying the path to plugins.zip on the Amazon MWAA console (the first time)
  - Specifying the plugins.zip version on the Amazon MWAA console
- Example use cases for plugins.zip
- What's next?

Prerequisites

You'll need the following before you can complete the steps on this page.

- **Permissions** — Your AWS account must have been granted access by your administrator to the AmazonMWAAFullConsoleAccess access control policy for your environment. In addition, your Amazon MWAA environment must be permitted by your execution role to access the AWS resources used by your environment.
- **Access** — If you require access to public repositories to install dependencies directly on the web server, your environment must be configured with **public network** web server access. For more information, see the section called “Apache Airflow access modes”.
- **Amazon S3 configuration** — The Amazon S3 bucket used to store your DAGs, custom plugins in `plugins.zip`, and Python dependencies in `requirements.txt` must be configured with *Public Access Blocked* and *Versioning Enabled*.

How it works

To run custom plugins on your environment, you must do three things:

1. Create a `plugins.zip` file locally.
2. Upload the local `plugins.zip` file to your Amazon S3 bucket.
3. Specify the version of this file in the **Plugins file** field on the Amazon MWAA console.

> **Note**
>
> If this is the first time you're uploading a `plugins.zip` to your Amazon S3 bucket, you also need to specify the path to the file on the Amazon MWAA console. You only need to complete this step once.

When to use the plugins

Plugins are required only for extending the Apache Airflow user interface, as outlined in the Apache Airflow documentation. Custom operators can be placed directly in the `/dags` folder alongside your DAG code.

If you need to create your own integrations with external systems, place them in the `/dags` folder or a subfolder within it, but not in the `plugins.zip` folder. In Apache Airflow 2.x, plugins are primarily used for extending the UI.

Similarly, other dependencies should not be placed in `plugins.zip`. Instead, they can be stored in a location under the Amazon S3 `/dags` folder, where they will be synchronized to each Amazon MWAA container before Apache Airflow starts.

> **Note**
>
> Any file in the `/dags` folder or in `plugins.zip` that does not explicitly define an Apache Airflow DAG object must be listed in an `.airflowignore` file.

Custom plugins overview

Apache Airflow's built-in plugin manager can integrate external features to its core by simply dropping files in an `$AIRFLOW_HOME/plugins` folder. It allows you to use custom Apache Airflow operators, hooks, sensors, or interfaces. The following section provides an example of flat and nested directory structures in a local development environment and the resulting import statements, which determines the directory structure within a plugins.zip.

Custom plugins directory and size limits

The Apache Airflow *Scheduler* and the *Workers* look for custom plugins during startup on the AWS-managed Fargate container for your environment at `/usr/local/airflow/plugins/*`.

- **Directory structure**. The directory structure (at `/*`) is based on the contents of your `plugins.zip` file. For example, if your `plugins.zip` contains the `operators` directory as a top-level directory, then the directory will be extracted to `/usr/local/airflow/plugins/operators` on your environment.
- **Size limit**. We recommend a `plugins.zip` file less than than 1 GB. The larger the size of a `plugins.zip` file, the longer the startup time on an environment. Although Amazon MWAA doesn't limit the size of a `plugins.zip` file explicitly, if dependencies can't be installed within ten minutes, the Fargate service will time-out and attempt to rollback the environment to a stable state.

> **Note**
>
> For environments using Apache Airflow v1.10.12 or Apache Airflow v2.0.2, Amazon MWAA limits outbound traffic on the Apache Airflow web server, and does not allow you to install plugins nor Python dependencies directly on the web server. Starting with Apache Airflow v2.2.2, Amazon MWAA can install plugins and dependencies directly on the web server.

Examples of custom plugins

The following section uses sample code in the *Apache Airflow reference guide* to show how to structure your local development environment.

Example using a flat directory structure in plugins.zip

Apache Airflow v2

The following example shows a `plugins.zip` file with a flat directory structure for Apache Airflow v2.

Example flat directory with PythonVirtualenvOperator plugins.zip

The following example shows the top-level tree of a plugins.zip file for the PythonVirtualenvOperator custom plugin in Creating a custom plugin for Apache Airflow PythonVirtualenvOperator.

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

Example plugins/virtual_python_plugin.py

The following example shows the PythonVirtualenvOperator custom plugin.

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

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

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

def _generate_virtualenv_cmd(tmp_dir: str, python_bin: str, system_site_packages:
 bool) -> List[str]:
    cmd = ['python3','/usr/local/airflow/.local/lib/python3.7/site-packages/
virtualenv', tmp_dir]
```

```
    if system_site_packages:
        cmd.append('--system-site-packages')
    if python_bin is not None:
        cmd.append(f'--python={python_bin}')
    return cmd

airflow.utils.python_virtualenv._generate_virtualenv_cmd=_generate_virtualenv_cmd

class VirtualPythonPlugin(AirflowPlugin):
    name = 'virtual_python_plugin'
```

Apache Airflow v1

The following example shows a `plugins.zip` file with a flat directory structure for Apache Airflow v1.

Example flat directory with PythonVirtualenvOperator plugins.zip

The following example shows the top-level tree of a plugins.zip file for the PythonVirtualenvOperator custom plugin in Creating a custom plugin for Apache Airflow PythonVirtualenvOperator.

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

Example plugins/virtual_python_plugin.py

The following example shows the PythonVirtualenvOperator custom plugin.

```
from airflow.plugins_manager import AirflowPlugin
from airflow.operators.python_operator import PythonVirtualenvOperator

def _generate_virtualenv_cmd(self, tmp_dir):
    cmd = ['python3','/usr/local/airflow/.local/lib/python3.7/site-packages/
virtualenv', tmp_dir]
    if self.system_site_packages:
        cmd.append('--system-site-packages')
    if self.python_version is not None:
        cmd.append('--python=python{}'.format(self.python_version))
    return cmd
PythonVirtualenvOperator._generate_virtualenv_cmd=_generate_virtualenv_cmd

class EnvVarPlugin(AirflowPlugin):
```

```
    name = 'virtual_python_plugin'
```

Example using a nested directory structure in plugins.zip

Apache Airflow v2

The following example shows a `plugins.zip` file with separate directories for `hooks`, `operators`, and a `sensors` directory for Apache Airflow v2.

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
```

The following example shows the import statements in the DAG (DAGs folder) that uses the custom plugins.

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]
```

The following examples show each of the import statements needed in the custom plugin files.

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
```

Follow the steps in Testing custom plugins using the Amazon MWAA CLI utility, and then Creating a plugins.zip file to zip the contents **within** your `plugins` directory. For example, `cd plugins`.

Apache Airflow v1

The following example shows a `plugins.zip` file with separate directories for `hooks`, `operators`, and a `sensors` directory for Apache Airflow v1.10.12.

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
```

The following example shows the import statements in the DAG (DAGs folder) that uses the custom plugins.

Example dags/your_dag.py

```
from airflow import DAG
from datetime import datetime, timedelta
from operators.my_operator import MyOperator
from sensors.my_sensor import MySensor
from operators.hello_operator import HelloOperator

default_args = {
 'owner': 'airflow',
 'depends_on_past': False,
 'start_date': datetime(2018, 1, 1),
 'email_on_failure': False,
 'email_on_retry': False,
 'retries': 1,
 'retry_delay': timedelta(minutes=5),
}


with DAG('customdag',
   max_active_runs=3,
   schedule_interval='@once',
   default_args=default_args) as dag:

 sens = MySensor(
  task_id='taskA'
 )

 op = MyOperator(
  task_id='taskB',
  my_field='some text'
 )

 hello_task = HelloOperator(task_id='sample-task', name='foo_bar')
```

```
 sens >> op >> hello_task
```

Example plugins/my_airflow_plugin.py

```
from airflow.plugins_manager import AirflowPlugin
from hooks.my_airflow_hook import *
from operators.my_airflow_operator import *
from utils.my_utils import *

class PluginName(AirflowPlugin):

    name = 'my_airflow_plugin'

    hooks = [MyHook]
    operators = [MyOperator]
    sensors = [MySensor]
```

The following examples show each of the import statements needed in the custom plugin files.

Example hooks/my_airflow_hook.py

```
from airflow.hooks.base_hook import BaseHook


class MyHook(BaseHook):

    def my_method(self):
        print("Hello World")
```

Example sensors/my_airflow_sensor.py

```
from airflow.sensors.base_sensor_operator import BaseSensorOperator
from airflow.utils.decorators import apply_defaults


class MySensor(BaseSensorOperator):

    @apply_defaults
    def __init__(self,
                 *args,
                 **kwargs):
```

```
        super(MySensor, self).__init__(*args, **kwargs)

    def poke(self, context):
        return True
```

Example operators/my_airflow_operator.py

```
from airflow.operators.bash_operator import BaseOperator
from airflow.utils.decorators import apply_defaults
from hooks.my_hook import MyHook


class MyOperator(BaseOperator):

    @apply_defaults
    def __init__(self,
                 my_field,
                 *args,
                 **kwargs):
        super(MyOperator, self).__init__(*args, **kwargs)
        self.my_field = my_field

    def execute(self, context):
        hook = MyHook('my_conn')
        hook.my_method()
```

Example operators/hello_operator.py

```
from airflow.models.baseoperator import BaseOperator
from airflow.utils.decorators import apply_defaults

class HelloOperator(BaseOperator):

    @apply_defaults
    def __init__(
            self,
            name: str,
            **kwargs) -> None:
        super().__init__(**kwargs)
        self.name = name

    def execute(self, context):
        message = "Hello {}".format(self.name)
```

```
        print(message)
        return message
```

Follow the steps in Testing custom plugins using the Amazon MWAA CLI utility, and then Creating a plugins.zip file to zip the contents **within** your `plugins` directory. For example, `cd plugins`.

Creating a plugins.zip file

The following steps describe the steps we recommend to create a plugins.zip file locally.

Step one: Test custom plugins using the Amazon MWAA CLI utility

- The command line interface (CLI) utility replicates an Amazon Managed Workflows for Apache Airflow environment locally.
- The CLI builds a Docker container image locally that's similar to an Amazon MWAA production image. This allows you to run a local Apache Airflow environment to develop and test DAGs, custom plugins, and dependencies before deploying to Amazon MWAA.
- To run the CLI, see the aws-mwaa-local-runner on GitHub.

Step two: Create the plugins.zip file

You can use a built-in ZIP archive utility, or any other ZIP utility (such as 7zip) to create a .zip file.

> **Note**
>
> The built-in zip utility for Windows OS may add subfolders when you create a .zip file. We recommend verifying the contents of the plugins.zip file before uploading to your Amazon S3 bucket to ensure no additional directories were added.

1. Change directories to your local Airflow plugins directory. For example:

```
myproject$ cd plugins
```

2. Run the following command to ensure that the contents have executable permissions (macOS and Linux only).

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

3. Zip the contents **within** your `plugins` folder.

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

Uploading `plugins.zip` to Amazon S3

You can use the Amazon S3 console or the AWS Command Line Interface (AWS CLI) to upload a `plugins.zip` file to your Amazon S3 bucket.

Using the AWS CLI

The AWS Command Line Interface (AWS CLI) is an open source tool that enables you to interact with AWS services using commands in your command-line shell. To complete the steps on this page, you need the following:

- AWS CLI – Install version 2.
- AWS CLI – Quick configuration with `aws configure`.

To upload using the AWS CLI

1. In your command prompt, navigate to the directory where your `plugins.zip` file is stored. For example:

```
cd plugins
```

2. Use the following command to list all of your Amazon S3 buckets.

```
aws s3 ls
```

3. Use the following command to list the files and folders in the Amazon S3 bucket for your environment.

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

4. Use the following command to upload the `plugins.zip` file to the Amazon S3 bucket for your environment.

```
aws s3 cp plugins.zip s3://YOUR_S3_BUCKET_NAME/plugins.zip
```

Using the Amazon S3 console

The Amazon S3 console is a web-based user interface that allows you to create and manage the resources in your Amazon S3 bucket.

To upload using the Amazon S3 console

1. Open the Environments page on the Amazon MWAA console.
2. Choose an environment.
3. Select the **S3 bucket** link in the **DAG code in S3** pane to open your storage bucket on the Amazon S3 console.
4. Choose **Upload**.
5. Choose **Add file**.
6. Select the local copy of your `plugins.zip`, choose **Upload**.

Installing custom plugins on your environment

This section describes how to install the custom plugins you uploaded to your Amazon S3 bucket by specifying the path to the plugins.zip file, and specifying the version of the plugins.zip file each time the zip file is updated.

Specifying the path to `plugins.zip` on the Amazon MWAA console (the first time)

If this is the first time you're uploading a `plugins.zip` to your Amazon S3 bucket, you also need to specify the path to the file on the Amazon MWAA console. You only need to complete this step once.

1. Open the Environments page on the Amazon MWAA console.
2. Choose an environment.
3. Choose **Edit**.
4. On the **DAG code in Amazon S3** pane, choose **Browse S3** next to the **Plugins file - optional** field.

5. Select the `plugins.zip` file on your Amazon S3 bucket.
6. Choose **Choose**.
7. Choose **Next**, **Update environment**.

Specifying the `plugins.zip` version on the Amazon MWAA console

You need to specify the version of your `plugins.zip` file on the Amazon MWAA console each time you upload a new version of your `plugins.zip` in your Amazon S3 bucket.

1. Open the Environments page on the Amazon MWAA console.
2. Choose an environment.
3. Choose **Edit**.
4. On the **DAG code in Amazon S3** pane, choose a `plugins.zip` version in the dropdown list.
5. Choose **Next**.

Example use cases for plugins.zip

- Learn how to create a custom plugin in Custom plugin with Apache Hive and Hadoop.
- Learn how to create a custom plugin in Custom plugin to patch PythonVirtualenvOperator .
- Learn how to create a custom plugin in Custom plugin with Oracle.
- Learn how to create a custom plugin in the section called “Changing a DAG's timezone”.

What's next?

- Test your DAGs, custom plugins, and Python dependencies locally using the aws-mwaa-local-runner on GitHub.

Installing Python dependencies

A Python dependency is any package or distribution that is not included in the Apache Airflow base install for your Apache Airflow version on your Amazon Managed Workflows for Apache Airflow environment. This topic describes the steps to install Apache Airflow Python dependencies on your Amazon MWAA environment using a `requirements.txt` file in your Amazon S3 bucket.

Contents

- Prerequisites
- How it works
- Python dependencies overview
  - Python dependencies location and size limits
- Creating a requirements.txt file
  - Step one: Test Python dependencies using the Amazon MWAA CLI utility
  - Step two: Create the requirements.txt
- Uploading requirements.txt to Amazon S3
  - Using the AWS CLI
  - Using the Amazon S3 console
- Installing Python dependencies on your environment
  - Specifying the path to requirements.txt on the Amazon MWAA console (the first time)
  - Specifying the requirements.txt version on the Amazon MWAA console
- Viewing logs for your requirements.txt
- What's next?

Prerequisites

You'll need the following before you can complete the steps on this page.

- **Permissions** — Your AWS account must have been granted access by your administrator to the AmazonMWAAFullConsoleAccess access control policy for your environment. In addition, your Amazon MWAA environment must be permitted by your execution role to access the AWS resources used by your environment.
- **Access** — If you require access to public repositories to install dependencies directly on the web server, your environment must be configured with **public network** web server access. For more information, see the section called "Apache Airflow access modes".
- **Amazon S3 configuration** — The Amazon S3 bucket used to store your DAGs, custom plugins in `plugins.zip`, and Python dependencies in `requirements.txt` must be configured with *Public Access Blocked* and *Versioning Enabled*.

How it works

On Amazon MWAA, you install all Python dependencies by uploading a `requirements.txt` file to your Amazon S3 bucket, then specifying the version of the file on the Amazon MWAA console each time you update the file. Amazon MWAA runs `pip3 install -r requirements.txt` to install the Python dependencies on the Apache Airflow scheduler and each of the workers.

To run Python dependencies on your environment, you must do three things:

1. Create a `requirements.txt` file locally.
2. Upload the local `requirements.txt` to your Amazon S3 bucket.
3. Specify the version of this file in the **Requirements file** field on the Amazon MWAA console.

> **Note**
>
> If this is the first time you're creating and uploading a `requirements.txt` to your Amazon S3 bucket, you also need to specify the path to the file on the Amazon MWAA console. You only need to complete this step once.

Python dependencies overview

You can install Apache Airflow extras and other Python dependencies from the Python Package Index (PyPi.org), Python wheels (`.whl`), or Python dependencies hosted on a private PyPi/PEP-503 Compliant Repo on your environment.

Python dependencies location and size limits

The Apache Airflow *Scheduler* and the *Workers* look for the packages in the `requirements.txt` file and the packages are installed on the environment at `/usr/local/airflow/.local/bin`.

- **Size limit**. We recommend a `requirements.txt` file that references libraries whose combined size is less than than 1 GB. The more libraries Amazon MWAA needs to install, the longer the *startup* time on an environment. Although Amazon MWAA doesn't limit the size of installed libraries explicitly, if dependencies can't be installed within ten minutes, the Fargate service will time-out and attempt to rollback the environment to a stable state.

Creating a requirements.txt file

The following steps describe the steps we recommend to create a requirements.txt file locally.

Step one: Test Python dependencies using the Amazon MWAA CLI utility

- The command line interface (CLI) utility replicates an Amazon Managed Workflows for Apache Airflow environment locally.
- The CLI builds a Docker container image locally that's similar to an Amazon MWAA production image. This allows you to run a local Apache Airflow environment to develop and test DAGs, custom plugins, and dependencies before deploying to Amazon MWAA.
- To run the CLI, see the aws-mwaa-local-runner on GitHub.

Step two: Create the `requirements.txt`

The following section describes how to specify Python dependencies from the Python Package Index in a `requirements.txt` file.

Apache Airflow v2

1. **Test locally**. Add additional libraries iteratively to find the right combination of packages and their versions, before creating a `requirements.txt` file. To run the Amazon MWAA CLI utility, see the aws-mwaa-local-runner on GitHub.
2. **Review the Apache Airflow package extras**. To view a list of the packages installed for Apache Airflow v2 on Amazon MWAA, see Amazon MWAA local runner `requirements.txt` on the GitHub website.
3. **Add a constraints statement**. Add the constraints file for your Apache Airflow v2 environment at the top of your `requirements.txt` file. Apache Airflow constraints files specify the provider versions available at the time of a Apache Airflow release.

   Beginning with Apache Airflow v2.7.2, your requirements file must include a `--constraint` statement. If you do not provide a constraint, Amazon MWAA will specify one for you to ensure the packages listed in your requirements are compatible with the version of Apache Airflow you are using.

   In the following example, replace *`{environment-version}`* with your environment's version number, and *`{Python-version}`* with the version of Python that's compatible with your environment.

For information on the version of Python compatible with your Apache Airflow environment, see Apache Airflow Versions.

```
--constraint "https://raw.githubusercontent.com/apache/airflow/
constraints-{Airflow-version}/constraints-{Python-version}.txt"
```

If the constraints file determines that `xyz==1.0` package is not compatible with other packages in your environment, `pip3 install` will fail in order to prevent incompatible libraries from being installed to your environment. If installation fails for any packages, you can view error logs for each Apache Airflow component (the scheduler, worker, and web server) in the corresponding log stream on CloudWatch Logs. For more information on log types, see the section called "Viewing Airflow logs".

4. **Apache Airflow packages**. Add the package extras and the version (==). This helps to prevent packages of the same name, but different version, from being installed on your environment.

```
apache-airflow[package-extra]==2.5.1
```

5. **Python libraries**. Add the package name and the version (==) in your `requirements.txt` file. This helps to prevent a future breaking update from PyPi.org from being automatically applied.

```
library == version
```

Example Boto3 and psycopg2-binary

This example is provided for demonstration purposes. The boto and psycopg2-binary libraries are included with the Apache Airflow v2 base install and don't need to be specified in a `requirements.txt` file.

```
boto3==1.17.54
boto==2.49.0
botocore==1.20.54
psycopg2-binary==2.8.6
```

If a package is specified without a version, Amazon MWAA installs the latest version of the package from PyPi.org. This version may conflict with other packages in your `requirements.txt`.

Uploading `requirements.txt` to Amazon S3

You can use the Amazon S3 console or the AWS Command Line Interface (AWS CLI) to upload a `requirements.txt` file to your Amazon S3 bucket.

Using the AWS CLI

The AWS Command Line Interface (AWS CLI) is an open source tool that enables you to interact with AWS services using commands in your command-line shell. To complete the steps on this page, you need the following:

- AWS CLI – Install version 2.
- AWS CLI – Quick configuration with `aws configure`.

To upload using the AWS CLI

1. Use the following command to list all of your Amazon S3 buckets.

```
aws s3 ls
```

2. Use the following command to list the files and folders in the Amazon S3 bucket for your environment.

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

3. The following command uploads a `requirements.txt` file to an Amazon S3 bucket.

```
aws s3 cp requirements.txt s3://YOUR_S3_BUCKET_NAME/requirements.txt
```

Using the Amazon S3 console

The Amazon S3 console is a web-based user interface that allows you to create and manage the resources in your Amazon S3 bucket.

To upload using the Amazon S3 console

1. Open the Environments page on the Amazon MWAA console.
2. Choose an environment.
3. Select the **S3 bucket** link in the **DAG code in S3** pane to open your storage bucket on the Amazon S3 console.
4. Choose **Upload**.
5. Choose **Add file**.
6. Select the local copy of your `requirements.txt`, choose **Upload**.

Installing Python dependencies on your environment

This section describes how to install the dependencies you uploaded to your Amazon S3 bucket by specifying the path to the requirements.txt file, and specifying the version of the requirements.txt file each time it's updated.

Specifying the path to `requirements.txt` on the Amazon MWAA console (the first time)

If this is the first time you're creating and uploading a `requirements.txt` to your Amazon S3 bucket, you also need to specify the path to the file on the Amazon MWAA console. You only need to complete this step once.

1. Open the Environments page on the Amazon MWAA console.
2. Choose an environment.
3. Choose **Edit**.
4. On the **DAG code in Amazon S3** pane, choose **Browse S3** next to the **Requirements file - optional** field.
5. Select the `requirements.txt` file on your Amazon S3 bucket.
6. Choose **Choose**.
7. Choose **Next**, **Update environment**.

You can begin using the new packages immediately after your environment finishes updating.

Specifying the `requirements.txt` version on the Amazon MWAA console

You need to specify the version of your `requirements.txt` file on the Amazon MWAA console each time you upload a new version of your `requirements.txt` in your Amazon S3 bucket.

1. Open the Environments page on the Amazon MWAA console.
2. Choose an environment.
3. Choose **Edit**.
4. On the **DAG code in Amazon S3** pane, choose a `requirements.txt` version in the dropdown list.
5. Choose **Next**, **Update environment**.

You can begin using the new packages immediately after your environment finishes updating.

Viewing logs for your `requirements.txt`

You can view Apache Airflow logs for the *Scheduler* scheduling your workflows and parsing your `dags` folder. The following steps describe how to open the log group for the *Scheduler* on the Amazon MWAA console, and view Apache Airflow logs on the CloudWatch Logs console.

To view logs for a `requirements.txt`

1. Open the Environments page on the Amazon MWAA console.
2. Choose an environment.
3. Choose the **Airflow scheduler log group** on the **Monitoring** pane.
4. Choose the `requirements_install_ip` log in **Log streams**.
5. You should see the list of packages that were installed on the environment at `/usr/local/airflow/.local/bin`. For example:

```
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. Review the list of packages and whether any of these encountered an error during installation. If something went wrong, you may see an error similar to the following:

```
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))
```

What's next?

- Test your DAGs, custom plugins, and Python dependencies locally using the aws-mwaa-local-runner on GitHub.

Deleting files on Amazon S3

This page describes how versioning works in an Amazon S3 bucket for an Amazon Managed Workflows for Apache Airflow environment, and the steps to delete a DAG, `plugins.zip`, or `requirements.txt` file.

Contents

- Prerequisites
- Versioning overview
- How it works
- Deleting a DAG on Amazon S3
- Removing a "current" requirements.txt or plugins.zip from an environment
- Deleting a "non-current" (previous) requirements.txt or plugins.zip version
- Using lifecycles to delete "non-current" (previous) versions and delete markers automatically
- Example lifecycle policy to delete requirements.txt "non-current" versions and delete markers automatically
- What's next?

Prerequisites

You'll need the following before you can complete the steps on this page.

- **Permissions** — Your AWS account must have been granted access by your administrator to the AmazonMWAAFullConsoleAccess access control policy for your environment. In addition, your Amazon MWAA environment must be permitted by your execution role to access the AWS resources used by your environment.
- **Access** — If you require access to public repositories to install dependencies directly on the web server, your environment must be configured with **public network** web server access. For more information, see the section called "Apache Airflow access modes".
- **Amazon S3 configuration** — The Amazon S3 bucket used to store your DAGs, custom plugins in `plugins.zip`, and Python dependencies in `requirements.txt` must be configured with *Public Access Blocked* and *Versioning Enabled*.

Versioning overview

The `requirements.txt` and `plugins.zip` in your Amazon S3 bucket are versioned. When Amazon S3 bucket versioning is enabled for an object, and an artifact (for example, plugins.zip) is deleted from an Amazon S3 bucket, the file doesn't get deleted entirely. Anytime an artifact is deleted on Amazon S3, a new copy of the file is created that is a 404 (Object not found) error/0k file that says "I'm not here." Amazon S3 calls this a *delete marker*. A delete marker is a "null" version of the file with a key name (or key) and version ID like any other object.

We recommend deleting file versions and delete markers periodically to reduce storage costs for your Amazon S3 bucket. To delete "non-current" (previous) file versions entirely, you must delete the versions of the file(s), and then the *delete marker* for the version.

How it works

Amazon MWAA runs a sync operation on your Amazon S3 bucket every thirty seconds. This causes any DAG deletions in an Amazon S3 bucket to be synced to the Airflow image of your Fargate container.

For `plugins.zip` and `requirements.txt` files, changes occur only after an environment update when Amazon MWAA builds a new Airflow image of your Fargate container with the custom plugins and Python dependencies. If you delete the *current* version of any of a `requirements.txt` or `plugins.zip` file, and then update your environment without providing a new version for the deleted file, then the update will fail with an error message, such as, "Unable to read version `{version}` of file `{file}`".

Deleting a DAG on Amazon S3

A DAG file (`.py`) is not versioned and can be deleted directly on the Amazon S3 console. The following steps describe how to delete a DAG on your Amazon S3 bucket.

To delete a DAG

1. Open the Environments page on the Amazon MWAA console.
2. Choose an environment.
3. Select the **S3 bucket** link in the **DAG code in S3** pane to open your storage bucket on the Amazon S3 console.
4. Choose the `dags` folder.
5. Select the DAG, **Delete**.
6. Under **Delete objects?**, type `delete`.
7. Choose **Delete objects**.

> **Note**
>
> Apache Airflow preserves historical DAG runs. After a DAG has been run in Apache Airflow, it remains in the Airflow DAGs list regardless of the file status, until you delete it in Apache Airflow. To delete a DAG in Apache Airflow, choose the red "delete" button under the **Links** column.

Removing a "current" requirements.txt or plugins.zip from an environment

Currently, there isn't a way to remove a plugins.zip or requirements.txt from an environment after they've been added, but we're working on the issue. In the interim, a workaround is to point to an empty text or zip file, respectively.

Deleting a "non-current" (previous) requirements.txt or plugins.zip version

The `requirements.txt` and `plugins.zip` files in your Amazon S3 bucket are versioned on Amazon MWAA. If you want to delete these files on your Amazon S3 bucket entirely, you must

retrieve the current version (121212) of the object (for example, plugins.zip), delete the version, and then remove the *delete marker* for the file version(s).

You can also delete "non-current" (previous) file versions on the Amazon S3 console; however, you'll still need to delete the *delete marker* using one of the following options.

- To retrieve the object version, see Retrieving object versions from a versioning-enabled bucket *in the Amazon S3 guide*.
- To delete the object version, see Deleting object versions from a versioning-enabled bucket *in the Amazon S3 guide*.
- To remove a delete marker, see Managing delete markers *in the Amazon S3 guide*.

Using lifecycles to delete "non-current" (previous) versions and delete markers automatically

You can configure a lifecycle policy for your Amazon S3 bucket to delete "non-current" (previous) versions of the plugins.zip and requirements.txt files in your Amazon S3 bucket after a certain number of days, or to remove an expired object's delete marker.

1. Open the Environments page on the Amazon MWAA console.
2. Choose an environment.
3. Under **DAG code in Amazon S3**, choose your Amazon S3 bucket.
4. Choose **Create lifecycle rule**.

Example lifecycle policy to delete requirements.txt "non-current" versions and delete markers automatically

The following example shows how to create a lifecycle rule that permanently deletes "non-current" versions of a requirements.txt file and their delete markers after thirty days.

1. Open the Environments page on the Amazon MWAA console.
2. Choose an environment.
3. Under **DAG code in Amazon S3**, choose your Amazon S3 bucket.
4. Choose **Create lifecycle rule**.

5. In **Lifecycle rule name**, type `Delete previous requirements.txt versions and delete markers after thirty days`.
6. In **Prefix**, **requirements**.
7. In **Lifecycle rule actions**, choose **Permanently delete previous versions of objects** and **Delete expired delete markers or incomplete multipart uploads**.
8. In **Number of days after objects become previous versions**, type `30`.
9. In **Expired object delete markers**, choose **Delete expired object delete markers, objects are permanently deleted after 30 days**.

What's next?

- Learn more about Amazon S3 delete markers in Managing delete markers.
- Learn more about Amazon S3 lifecycles in Expiring objects.

Networking

This guide describes the Amazon VPC network setup you'll need for an Amazon MWAA environment.

Sections

- About networking on Amazon MWAA
- Security in your VPC on Amazon MWAA
- Managing access to service-specific Amazon VPC endpoints on Amazon MWAA
- Creating the required VPC service endpoints in an Amazon VPC with private routing
- Managing your own Amazon VPC endpoints on Amazon MWAA

About networking on Amazon MWAA

An Amazon VPC is a virtual network that is linked to your AWS account. It gives you cloud security and the ability to scale dynamically by providing fine-grained control over your virtual infrastructure and network traffic segmentation. This page describes the Amazon VPC infrastructure with *public routing* or *private routing* that's needed to support an Amazon Managed Workflows for Apache Airflow environment.

Contents

- Terms
- What's supported
- VPC infrastructure overview
  - Public routing over the Internet
  - Private routing without Internet access
- Example use cases for an Amazon VPC and Apache Airflow access mode
  - Internet access is allowed - new Amazon VPC network
  - Internet access is not allowed - new Amazon VPC network
  - Internet access is not allowed - existing Amazon VPC network

Terms

Public routing

An Amazon VPC network that has access to the Internet.

Private routing

An Amazon VPC network **without** access to the Internet.

What's supported

The following table describes the types of Amazon VPCs Amazon MWAA supports.

Amazon VPC types	Supported	
An Amazon VPC owned by the account that is attemptin g to create the environment.	Yes	
A shared Amazon VPC where multiple AWS accounts create their AWS resources.	Yes	

VPC infrastructure overview

When you create an Amazon MWAA environment, Amazon MWAA creates between one to two VPC endpoints for your environment based on the Apache Airflow access mode you chose for your environment. These endpoints appear as Elastic Network Interfaces (ENIs) with private IPs in your Amazon VPC. After these endpoints are created, any traffic destined to these IPs is privately or publicly routed to the corresponding AWS services used by your environment.

The following section describes the Amazon VPC infrastructure required to route traffic publicly *over the Internet*, or privately *within your Amazon VPC*.

Public routing over the Internet

This section describes the Amazon VPC infrastructure of an environment with public routing. You'll need the following VPC infrastructure:

- **One VPC security group**. A VPC security group acts as a virtual firewall to control ingress (inbound) and egress (outbound) network traffic on an instance.
  - Up to 5 security groups can be specified.
  - The security group must specify a self-referencing inbound rule to itself.
  - The security group must specify an outbound rule for all traffic (`0.0.0.0/0`).
  - The security group must allow all traffic in the self-referencing rule. For example, (Recommended) Example all access self-referencing security group .
  - The security group can *optionally* restrict traffic further by specifying the port range for HTTPS port range `443` and a TCP port range `5432`. For example, (Optional) Example security group that restricts inbound access to port 5432 and (Optional) Example security group that restricts inbound access to port 443.
- **Two public subnets**. A public subnet is a subnet that's associated with a route table that has a route to an Internet gateway.
  - Two public subnets are required. This allows Amazon MWAA to build a new container image for your environment in your other availability zone, if one container fails.
  - The subnets must be in different Availability Zones. For example, `us-east-1a`, `us-east-1b`.
  - The subnets must route to a NAT gateway (or NAT instance) with an Elastic IP Address (EIP).
  - The subnets must have a route table that directs internet-bound traffic to an Internet gateway.
- **Two private subnets**. A private subnet is a subnet that's **not** associated with a route table that has a route to an Internet gateway.
  - Two private subnets are required. This allows Amazon MWAA to build a new container image for your environment in your other availability zone, if one container fails.
  - The subnets must be in different Availability Zones. For example, `us-east-1a`, `us-east-1b`.
  - The subnets *must* have a route table to a NAT device (gateway or instance).
  - The subnets **must not** route to an Internet gateway.
- **A network access control list (ACL)**. An NACL manages (by allow or deny rules) inbound and outbound traffic at the subnet level.
  - The NACL must have an inbound rule that allows all traffic (`0.0.0.0/0`).
  - The NACL must have an outbound rule that allows all traffic (`0.0.0.0/0`).
  - For example, (Recommended) Example ACLs.

- **Two NAT gateways (or NAT instances)**. A NAT device forwards traffic from the instances in the private subnet to the Internet or other AWS services, and then routes the response back to the instances.
  - The NAT device must be attached to a public subnet. (One NAT device per public subnet.)
  - The NAT device must have an Elastic IPv4 Address (EIP) attached to each public subnet.
- **An Internet gateway**. An Internet gateway connects an Amazon VPC to the Internet and other AWS services.
  - An Internet gateway must be attached to the Amazon VPC.

Private routing without Internet access

This section describes the Amazon VPC infrastructure of an environment with *private routing*. You'll need the following VPC infrastructure:

- **One VPC security group**. A VPC security group acts as a virtual firewall to control ingress (inbound) and egress (outbound) network traffic on an instance.
  - Up to 5 security groups can be specified.
  - The security group must specify a self-referencing inbound rule to itself.
  - The security group must specify an outbound rule for all traffic (`0.0.0.0/0`).
  - The security group must allow all traffic in the self-referencing rule. For example, (Recommended) Example all access self-referencing security group .
  - The security group can *optionally* restrict traffic further by specifying the port range for HTTPS port range 443 and a TCP port range 5432. For example, (Optional) Example security group that restricts inbound access to port 5432 and (Optional) Example security group that restricts inbound access to port 443.
- **Two private subnets**. A private subnet is a subnet that's **not** associated with a route table that has a route to an Internet gateway.
  - Two private subnets are required. This allows Amazon MWAA to build a new container image for your environment in your other availability zone, if one container fails.
  - The subnets must be in different Availability Zones. For example, `us-east-1a`, `us-east-1b`.
  - The subnets must have a route table to your VPC endpoints.
  - The subnets **must not** have a route table to a NAT device (gateway or instance), **nor** an Internet gateway.

- **A network access control list (ACL)**. An NACL manages (by allow or deny rules) inbound and outbound traffic at the subnet level.
  - The NACL must have an inbound rule that allows all traffic (`0.0.0.0/0`).
  - The NACL must have an outbound rule that denies all traffic (`0.0.0.0/0`).
  - For example, (Recommended) Example ACLs.
- **A local route table**. A local route table is a default route for communication within the VPC.
  - The local route table must be associated to your private subnets.
  - The local route table must enable instances in your VPC to communicate with your own network. For example, if you're using an AWS Client VPN to access the VPC interface endpoint for your Apache Airflow *Web server*, the route table must route to the VPC endpoint.
- **VPC endpoints** for each AWS service used by your environment, and Apache Airflow VPC endpoints in the same AWS Region and Amazon VPC as your Amazon MWAA environment.
  - A VPC endpoint for each AWS service used by the environment and VPC endpoints for Apache Airflow. For example, (Required) VPC endpoints.
  - The VPC endpoints must have private DNS enabled.
  - The VPC endpoints must be associated to your environment's two private subnets.
  - The VPC endpoints must be associated to your environment's security group.
  - The VPC endpoint policy for each endpoint should be configured to allow access to AWS services used by the environment. For example, (Recommended) Example VPC endpoint policy to allow all access.
  - A VPC endpoint policy for Amazon S3 should be configured to allow bucket access. For example, (Recommended) Example Amazon S3 gateway endpoint policy to allow bucket access.

Example use cases for an Amazon VPC and Apache Airflow access mode

This section descibes the different use cases for network access in your Amazon VPC and the Apache Airflow *Web server* access mode you should choose on the Amazon MWAA console.

Internet access is allowed - new Amazon VPC network

If Internet access in your VPC is allowed by your organization, *and* you would like users to access your Apache Airflow *Web server* over the Internet:

1. Create an Amazon VPC network *with Internet access*.

2. Create an environment with the **Public network** access mode for your Apache Airflow *Web server*.
3. **What we recommend**: We recommend using the AWS CloudFormation quick-start template that creates the Amazon VPC infrastructure, an Amazon S3 bucket, and an Amazon MWAA environment at the same time. To learn more, see Quick start tutorial for Amazon Managed Workflows for Apache Airflow.

If Internet access in your VPC is allowed by your organization, *and* you would like to limit Apache Airflow *Web server* access to users within your VPC:

1. Create an Amazon VPC network *with Internet access*.
2. Create a mechanism to access the VPC interface endpoint for your Apache Airflow *Web server* from your computer.
3. Create an environment with the **Private network** access mode for your Apache Airflow *Web server*.
4. **What we recommend**:
    a. We recommend using the Amazon MWAA console in Option one: Creating the VPC network on the Amazon MWAA console, or the AWS CloudFormation template in Option two: Creating an Amazon VPC network *with* Internet access.
    b. We recommend configuring access using an AWS Client VPN to your Apache Airflow *Web server* in Tutorial: Configuring private network access using an AWS Client VPN.

Internet access is not allowed - new Amazon VPC network

If Internet access in your VPC is **not allowed** by your organization:

1. Create an Amazon VPC network *without Internet access*.
2. Create a mechanism to access the VPC interface endpoint for your Apache Airflow *Web server* from your computer.
3. Create VPC endpoints for each AWS service used by your environment.
4. Create an environment with the **Private network** access mode for your Apache Airflow *Web server*.
5. **What we recommend**:

a. We recommend using the AWS CloudFormation template to create an Amazon VPC without Internet access and the VPC endpoints for each AWS service used by Amazon MWAA in Option three: Creating an Amazon VPC network *without* Internet access.

b. We recommend configuring access using an AWS Client VPN to your Apache Airflow *Web server* in Tutorial: Configuring private network access using an AWS Client VPN.

Internet access is not allowed - existing Amazon VPC network

If Internet access in your VPC is **not allowed** by your organization, *and* you already have the required Amazon VPC network *without Internet access*:

1. Create VPC endpoints for each AWS service used by your environment.
2. Create VPC endpoints for Apache Airflow.
3. Create a mechanism to access the VPC interface endpoint for your Apache Airflow *Web server* from your computer.
4. Create an environment with the **Private network** access mode for your Apache Airflow *Web server*.
5. **What we recommend**:

   a. We recommend creating and attaching the VPC endpoints needed for each AWS service used by Amazon MWAA, and the VPC endpoints needed for Apache Airflow in Creating the required VPC service endpoints in an Amazon VPC with private routing.

   b. We recommend configuring access using an AWS Client VPN to your Apache Airflow *Web server* in Tutorial: Configuring private network access using an AWS Client VPN.

Security in your VPC on Amazon MWAA

This page describes the Amazon VPC components used to secure your Amazon Managed Workflows for Apache Airflow environment and the configurations needed for these components.

Contents

- Terms
- Security overview
- Network access control lists (ACLs)

- (Recommended) Example ACLs
- VPC security groups
  - (Recommended) Example all access self-referencing security group
  - (Optional) Example security group that restricts inbound access to port 5432
  - (Optional) Example security group that restricts inbound access to port 443
- VPC endpoint policies (private routing only)
  - (Recommended) Example VPC endpoint policy to allow all access
  - (Recommended) Example Amazon S3 gateway endpoint policy to allow bucket access

Terms

Public routing

An Amazon VPC network that has access to the Internet.

Private routing

An Amazon VPC network **without** access to the Internet.

Security overview

Security groups and access control lists (ACLs) provide ways to control the network traffic across the subnets and instances in your Amazon VPC using rules you specify.

- Network traffic to and from a subnet can be controlled by Access Control Lists (ACLs). You only need one ACL, and the same ACL can be used on multiple environments.
- Network traffic to and from an instance can be controlled by an Amazon VPC security group. You can use between one to five security groups per environment.
- Network traffic to and from an instance can also be controlled by VPC endpoint policies. If Internet access within your Amazon VPC is not allowed by your organization and you're using an Amazon VPC network with *private routing*, a VPC endpoint policy is required for the AWS VPC endpoints and Apache Airflow VPC endpoints.

Network access control lists (ACLs)

A network access control list (ACL) can manage (by allow or deny rules) inbound and outbound traffic at the *subnet* level. An ACL is stateless, which means that inbound and outbound rules must be specified separately and explicitly. It is used to specify the types of network traffic that are allowed in or out from the instances in a VPC network.

Every Amazon VPC has a default ACL that allows all inbound and outbound traffic. You can edit the default ACL rules, or create a custom ACL and attach it to your subnets. A subnet can only have one ACL attached to it at any time, but one ACL can be attached to multiple subnets.

(Recommended) Example ACLs

The following example shows the *inbound* and *outbound* ACL rules that can be used for an Amazon VPC with *public routing* or *private routing*.

Rule number	Type	Protocol	Port range	Source	Allow/Deny
100	All IPv4 traffic	All	All	0.0.0.0/0	Allow
*	All IPv4 traffic	All	All	0.0.0.0/0	Deny

VPC security groups

A VPC security group acts as a virtual firewall that controls the network traffic at the *instance* level. A security group is stateful, which means that when an inbound connection is permitted, it is allowed to reply. It is used to specify the types of network traffic that are allowed in from the instances in a VPC network.

Every Amazon VPC has a default security group. By default, it has no inbound rules. It has an outbound rule that allows all outbound traffic. You can edit the default security group rules, or create a custom security group and attach it to your Amazon VPC. On Amazon MWAA, you need to configure inbound and outbound rules to direct traffic on your NAT gateways.

(Recommended) Example all access self-referencing security group

The following example shows the *inbound* security group rules that allows all traffic for an Amazon VPC with *public routing* or *private routing*. The security group in this example is a self-referencing rule to itself.

Type	Protocol	Source Type	Source		
All traffic	All	All	sg-0909e8e81919 / my-mwaa-vpc-security-group		

The following example shows the *outbound* security group rules.

Type	Protocol	Source Type	Source		
All traffic	All	All	0.0.0.0/0		

(Optional) Example security group that restricts inbound access to port 5432

The following example shows the *inbound* security group rules that allow all HTTPS traffic on port 5432 for the Amazon Aurora PostgreSQL metadata database (owned by Amazon MWAA) for your environment.

> **Note**
>
> If you choose to restrict traffic using this rule, you'll need to add another rule to allow TCP traffic on port 443.

Type	Protocol	Port range	Source type	Source	
Custom TCP	TCP	5432	Custom	sg-0909e8e81919 /	

Type	Protocol	Port range	Source type	Source	
				my-mwaa-vpc-security-group	

(Optional) Example security group that restricts inbound access to port 443

The following example shows the *inbound* security group rules that allow all TCP traffic on port 443 for the Apache Airflow *Web server*.

Type	Protocol	Port range	Source type	Source	
HTTPS	TCP	443	Custom	sg-0909e8e81919 / my-mwaa-vpc-security-group	

VPC endpoint policies (private routing only)

A VPC endpoint (AWS PrivateLink) policy controls access to AWS services from your private subnet. A VPC endpoint policy is an IAM resource policy that you attach to your VPC gateway or interface endpoint. This section describes the permissions needed for the VPC endpoint policies for each VPC endpoint.

We recommend using a VPC interface endpoint policy for each of the VPC endpoints you created that allows full access to all AWS services, and using your execution role exclusively for AWS permissions.

(Recommended) Example VPC endpoint policy to allow all access

The following example shows a VPC interface endpoint policy for an Amazon VPC with *private routing*.

```
{
    "Statement": [
        {
```

```
            "Action": "*",
            "Effect": "Allow",
            "Resource": "*",
            "Principal": "*"
        }
    ]
}
```

(Recommended) Example Amazon S3 gateway endpoint policy to allow bucket access

The following example shows a VPC gateway endpoint policy that provides access to the Amazon S3 buckets required for Amazon ECR operations for an Amazon VPC with *private routing*. This is required for your Amazon ECR image to be retrieved, in addition to the bucket where your DAGs and supporting files are stored.

```
{
  "Statement": [
    {
      "Sid": "Access-to-specific-bucket-only",
      "Principal": "*",
      "Action": [
        "s3:GetObject"
      ],
      "Effect": "Allow",
      "Resource": ["arn:aws:s3:::prod-region-starport-layer-bucket/*"]
    }
  ]
}
```

Managing access to service-specific Amazon VPC endpoints on Amazon MWAA

A VPC endpoint (AWS PrivateLink) enables you to privately connect your VPC to services hosted on AWS without requiring an Internet gateway, a NAT device, VPN, or firewall proxies. These endpoints are horizontally scalable and highly available virtual devices that allow communication between instances in your VPC and AWS services. This page describes the VPC endpoints created by Amazon MWAA, and how to access the VPC endpoint for your Apache Airflow *Web server* if you've chosen the **Private network** access mode on Amazon Managed Workflows for Apache Airflow.

Contents

- Pricing
- VPC endpoint overview
  - Public network access mode
  - Private network access mode
- Permission to use other AWS services
- Viewing VPC endpoints
  - Viewing VPC endpoints on the Amazon VPC console
  - Identifying the private IP addresses of your Apache Airflow Web server and its VPC endpoint
- Accessing the VPC endpoint for your Apache Airflow Web server (private network access)
  - Using an AWS Client VPN
  - Using a Linux Bastion Host
  - Using a Load Balancer (advanced)

Pricing

- AWS PrivateLink Pricing

VPC endpoint overview

When you create an Amazon MWAA environment, Amazon MWAA creates between one to two VPC endpoints for your environment. These endpoints appear as Elastic Network Interfaces (ENIs) with private IPs in your Amazon VPC. After these endpoints are created, any traffic destined to these IPs is privately or publicly routed to the corresponding AWS services used by your environment.

Public network access mode

If you chose the **Public network** access mode for your Apache Airflow *Web server*, network traffic is publicly routed *over the Internet*.

- Amazon MWAA creates a VPC interface endpoint for your Amazon Aurora PostgreSQL metadata database. The endpoint is created in the Availability Zones mapped to your private subnets and is independent from other AWS accounts.

- Amazon MWAA then binds an IP address from your private subnets to the interface endpoints. This is designed to support the best practice of binding a single IP from each Availability Zone of the Amazon VPC.

Private network access mode

If you chose the **Private network** access mode for your Apache Airflow *Web server*, network traffic is privately routed *within your Amazon VPC*.

- Amazon MWAA creates a VPC interface endpoint for your Apache Airflow *Web server*, and an interface endpoint for your Amazon Aurora PostgreSQL metadata database. The endpoints are created in the Availability Zones mapped to your private subnets and is independent from other AWS accounts.
- Amazon MWAA then binds an IP address from your private subnets to the interface endpoints. This is designed to support the best practice of binding a single IP from each Availability Zone of the Amazon VPC.

Permission to use other AWS services

The interface endpoints use the execution role for your environment in AWS Identity and Access Management (IAM) to manage permission to AWS resources used by your environment. As more AWS services are enabled for an environment, each service will require you to configure permission using your environment's execution role. To add permissions, see Amazon MWAA execution role.

If you've chosen the **Private network** access mode for your Apache Airflow *Web server*, you must also allow permission in the VPC endpoint policy for each endpoint. To learn more, see the section called “VPC endpoint policies (private routing only)”.

Viewing VPC endpoints

This section describes how to view the VPC endpoints created by Amazon MWAA, and how to identify the private IP addresses for your Apache Airflow VPC endpoint.

Viewing VPC endpoints on the Amazon VPC console

The following section shows the steps to view the VPC endpoint(s) created by Amazon MWAA, and any VPC endpoints you may have created if you're using *private routing* for your Amazon VPC.

To view the VPC endpoint(s)

1. Open the Endpoints page on the Amazon VPC console.
2. Use the AWS Region selector to select your region.
3. You should see the VPC interface endpoint(s) created by Amazon MWAA, and any VPC endpoints you may have created if you're using *private routing* in your Amazon VPC.

To learn more about the VPC service endpoints that are required for an Amazon VPC with *private routing*, see Creating the required VPC service endpoints in an Amazon VPC with private routing.

Identifying the private IP addresses of your Apache Airflow Web server and its VPC endpoint

The following steps describe how to retrieve the host name of your Apache Airflow Web server and its VPC interface endpoint, and their private IP addresses.

1. Use the following AWS Command Line Interface (AWS CLI) command to retrieve the host name for your Apache Airflow *Web server*.

   ```
   aws mwaa get-environment --name YOUR_ENVIRONMENT_NAME --query
    'Environment.WebserverUrl'
   ```

   You should see something similar to the following response:

   ```
   "99aa99aa-55aa-44a1-a91f-f4552cf4e2f5-vpce.c10.us-west-2.airflow.amazonaws.com"
   ```

2. Run a *dig* command on the host name returned in the response of the previous command. For example:

   ```
   dig CNAME +short 99aa99aa-55aa-44a1-a91f-f4552cf4e2f5-vpce.c10.us-
   west-2.airflow.amazonaws.com
   ```

   You should see something similar to the following response:

   ```
   vpce-0699aa333a0a0a0-bf90xjtr.vpce-svc-00bb7c2ca2213bc37.us-
   west-2.vpce.amazonaws.com.
   ```

3. Use the following AWS Command Line Interface (AWS CLI) command to retrieve the VPC endpoint DNS name returned in the response of the previous command. For example:

```
aws ec2 describe-vpc-endpoints | grep vpce-0699aa333a0a0a0-bf90xjtr.vpce-
svc-00bb7c2ca2213bc37.us-west-2.vpce.amazonaws.com.
```

You should see something similar to the following response:

```
"DnsName": "vpce-066777a0a0a0-bf90xjtr.vpce-svc-00bb7c2ca2213bc37.us-
west-2.vpce.amazonaws.com",
```

4. Run either an *nslookup* or *dig* command on your Apache Airflow host name and its VPC endpoint DNS name to retrieve the IP addresses. For example:

```
dig +short YOUR_AIRFLOW_HOST_NAME YOUR_AIRFLOW_VPC_ENDPOINT_DNS
```

You should see something similar to the following response:

```
192.0.5.1
  192.0.6.1
```

Accessing the VPC endpoint for your Apache Airflow Web server (private network access)

If you've chosen the **Private network** access mode for your Apache Airflow *Web server*, you'll need to create a mechanism to access the VPC interface endpoint for your Apache Airflow *Web server*. You must use the same Amazon VPC, VPC security group, and private subnets as your Amazon MWAA environment for these resources.

Using an AWS Client VPN

AWS Client VPN is a managed client-based VPN service that enables you to securely access your AWS resources and resources in your on-premises network. It provides a secure TLS connection from any location using the OpenVPN client.

We recommend following the Amazon MWAA tutorial to configure a Client VPN: Tutorial: Configuring private network access using an AWS Client VPN.

Using a Linux Bastion Host

A bastion host is a server whose purpose is to provide access to a private network from an external network, such as over the Internet from your computer. Linux instances are in a public subnet, and they are set up with a security group that allows SSH access from the security group attached to the underlying Amazon EC2 instance running the bastion host.

We recommend following the Amazon MWAA tutorial to configure a Linux Bastion Host: Tutorial: Configuring private network access using a Linux Bastion Host.

Using a Load Balancer (advanced)

The following section shows the configurations you'll need to apply to an Application Load Balancer.

1. **Target groups**. You'll need to use target groups that point to the private IP addresses for your Apache Airflow *Web server*, and its VPC interface endpoint. We recommend specifying both private IP addresses as your registered targets, as using only one can reduce availability. For more information on how to identify the private IP addresses, see the section called "Identifying the private IP addresses of your Apache Airflow Web server and its VPC endpoint".
2. **Status codes**. We recommend using `200` and `302` status codes in your target group settings. Otherwise, the targets may be flagged as unhealthy if the VPC endpoint for the Apache Airflow *Web server* responds with a `302 Redirect` error.
3. **HTTPS Listener**. You'll need to specify the target port for the Apache Airflow *Web server*. For example:

Protocol	Port
HTTPS	443

4. **ACM new domain**. If you want to associate an SSL/TLS certificate in AWS Certificate Manager, you'll need to create a new domain for the HTTPS listener for your load balancer.
5. **ACM certificate region**. If you want to associate an SSL/TLS certificate in AWS Certificate Manager, you'll need to upload to the same AWS Region as your environment. For example:

- **Example region to upload certificate**

```
aws acm import-certificate --certificate fileb://Certificate.pem --certificate-
chain fileb://CertificateChain.pem --private-key fileb://PrivateKey.pem --
region us-west-2
```

Creating the required VPC service endpoints in an Amazon VPC with private routing

An existing Amazon VPC network *without Internet access* needs additional VPC service endpoints (AWS PrivateLink) to use Apache Airflow on Amazon Managed Workflows for Apache Airflow. This page describes the VPC endpoints required for the AWS services used by Amazon MWAA, the VPC endpoints required for Apache Airflow, and how to create and attach the VPC endpoints to an existing Amazon VPC with private routing.

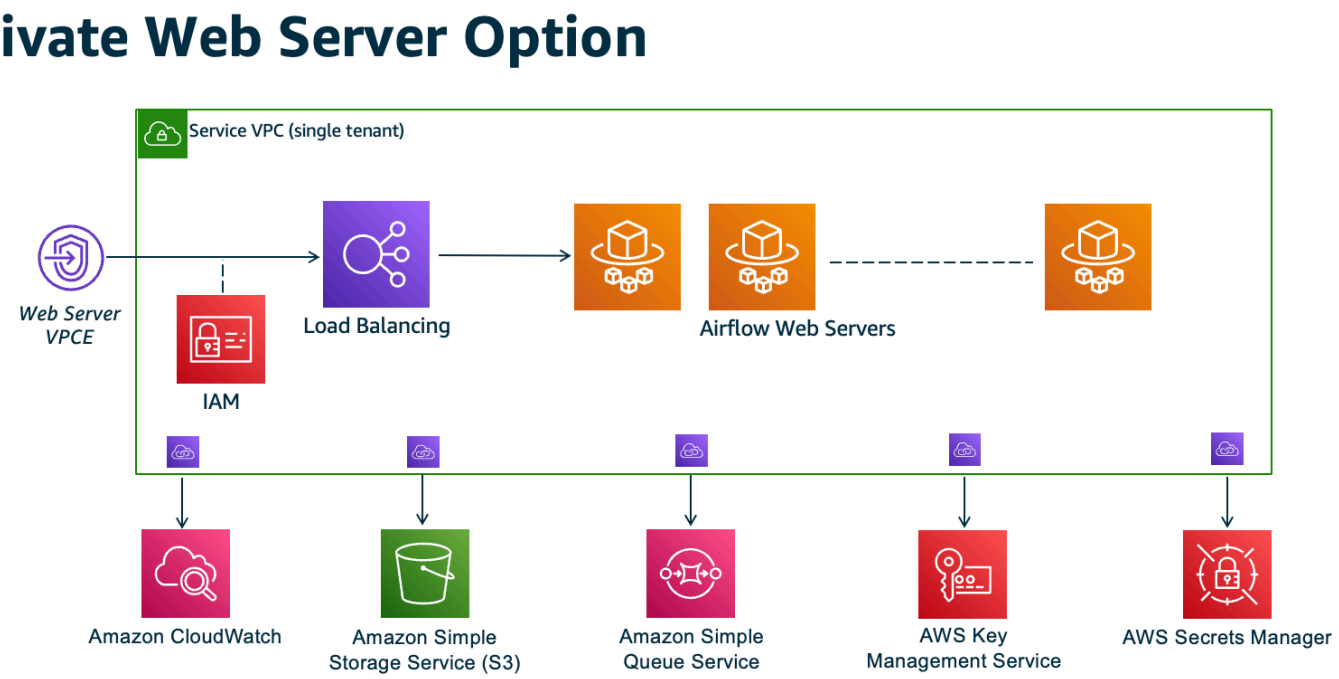
Contents

- Pricing
- Private network and private routing
- (Required) VPC endpoints
- Attaching the required VPC endpoints
  - VPC endpoints required for AWS services
  - VPC endpoints required for Apache Airflow
- (Optional) Enable private IP addresses for your Amazon S3 VPC interface endpoint
  - Using Route 53
  - VPCs with custom DNS

Pricing

- AWS PrivateLink Pricing

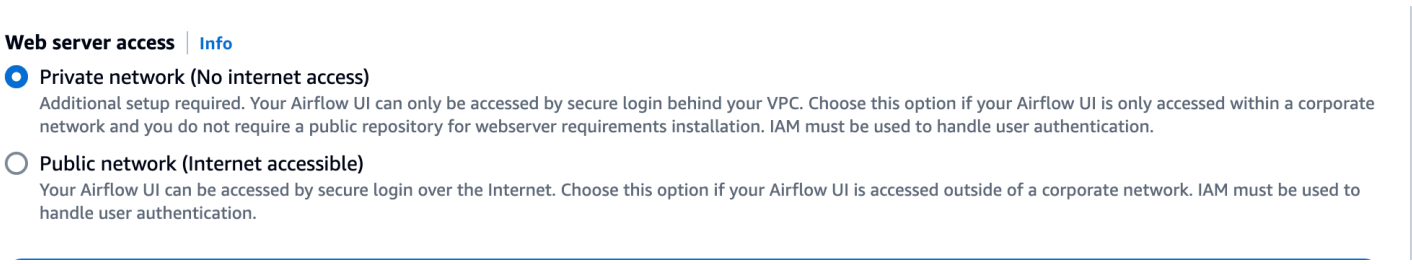
Private network and private routing



The private network access mode limits access to the Apache Airflow UI to users *within your Amazon VPC* that have been granted access to the IAM policy for your environment.

When you create an environment with private web server access, you must package all of your dependencies in a Python wheel archive (`.whl`), then reference the `.whl` in your `requirements.txt`. For instructions on packaging and installing your dependencies using wheel, see Managing dependencies using Python wheel.

The following image shows where to find the **Private network** option on the Amazon MWAA console.



- **Private routing**. An Amazon VPC *without Internet access* limits network traffic within the VPC. This page assumes your Amazon VPC does not have Internet access and requires VPC endpoints

for each AWS service used by your environment, and VPC endpoints for Apache Airflow in the same AWS Region and Amazon VPC as your Amazon MWAA environment.

(Required) VPC endpoints

The following section shows the required VPC endpoints needed for an Amazon VPC *without Internet access*. It lists the VPC endpoints for each AWS service used by Amazon MWAA, including the VPC endpoints needed for Apache Airflow.

```
com.amazonaws.YOUR_REGION.s3
com.amazonaws.YOUR_REGION.monitoring
com.amazonaws.YOUR_REGION.logs
com.amazonaws.YOUR_REGION.sqs
com.amazonaws.YOUR_REGION.kms
```

Note

When using Transit Gateway or any other routing that does not go directly to the AWS API endpoints, we recommend you to add AWS PrivateLink to your Amazon MWAA private subnets for the following services:

- Amazon S3
- Amazon SQS
- CloudWatch Logs
- CloudWatch metrics
- AWS KMS (if applicable)

This ensures that your Amazon MWAA environment can securely and efficiently communicate with these services without routing traffic through the public internet, thereby improving security and performance.

Attaching the required VPC endpoints

This section describes the steps to attach the required VPC endpoints for an Amazon VPC with private routing.

VPC endpoints required for AWS services

The following section shows the steps to attach the VPC endpoints for the AWS services used by an environment to an existing Amazon VPC.

To attach VPC endpoints to your private subnets

1. Open the Endpoints page on the Amazon VPC console.
2. Use the AWS Region selector to select your region.
3. Create the endpoint for Amazon S3:
    a. Choose **Create Endpoint**.
    b. In the *Filter by attributes or search by keyword* text field, type: **`.s3`**, then press *Enter* on your keyboard.
    c. We recommend choosing the service endpoint listed for the **Gateway** type.

    For example, **`com.amazonaws.us-west-2.s3 amazon Gateway`**
    d. Choose your environment's Amazon VPC in **VPC**.
    e. Ensure that your two private subnets in different Availability Zones are selected, and that that private DNS is enabled by selecting **Enable DNS name**.
    f. Choose your environment's Amazon VPC security group(s).
    g. Choose **Full Access** in **Policy**.
    h. Choose **Create endpoint**.
4. Create the endpoint for CloudWatch Logs:
    a. Choose **Create Endpoint**.
    b. In the *Filter by attributes or search by keyword* text field, type: **`.logs`**, then press *Enter* on your keyboard.
    c. Select the service endpoint.
    d. Choose your environment's Amazon VPC in **VPC**.
    e. Ensure that your two private subnets in different Availability Zones are selected, and that **Enable DNS name** is enabled.
    f. Choose your environment's Amazon VPC security group(s).
    g. Choose **Full Access** in **Policy**.
    h. Choose **Create endpoint**.

5. Create the endpoint for CloudWatch Monitoring:

    a. Choose **Create Endpoint**.

    b. In the *Filter by attributes or search by keyword* text field, type: **`.monitoring`**, then press *Enter* on your keyboard.

    c. Select the service endpoint.

    d. Choose your environment's Amazon VPC in **VPC**.

    e. Ensure that your two private subnets in different Availability Zones are selected, and that **Enable DNS name** is enabled.

    f. Choose your environment's Amazon VPC security group(s).

    g. Choose **Full Access** in **Policy**.

    h. Choose **Create endpoint**.

6. Create the endpoint for Amazon SQS:

    a. Choose **Create Endpoint**.

    b. In the *Filter by attributes or search by keyword* text field, type: **`.sqs`**, then press *Enter* on your keyboard.

    c. Select the service endpoint.

    d. Choose your environment's Amazon VPC in **VPC**.

    e. Ensure that your two private subnets in different Availability Zones are selected, and that **Enable DNS name** is enabled.

    f. Choose your environment's Amazon VPC security group(s).

    g. Choose **Full Access** in **Policy**.

    h. Choose **Create endpoint**.

7. Create the endpoint for AWS KMS:

    a. Choose **Create Endpoint**.

    b. In the *Filter by attributes or search by keyword* text field, type: **`.kms`**, then press *Enter* on your keyboard.

    c. Select the service endpoint.

    d. Choose your environment's Amazon VPC in **VPC**.

    e. Ensure that your two private subnets in different Availability Zones are selected, and that **Enable DNS name** is enabled.

f. Choose your environment's Amazon VPC security group(s).

g. Choose **Full Access** in **Policy**.

h. Choose **Create endpoint**.

VPC endpoints required for Apache Airflow

The following section shows the steps to attach the VPC endpoints for Apache Airflow to an existing Amazon VPC.

To attach VPC endpoints to your private subnets

1. Open the Endpoints page on the Amazon VPC console.
2. Use the AWS Region selector to select your region.
3. Create the endpoint for the Apache Airflow API:

    a. Choose **Create Endpoint**.

    b. In the *Filter by attributes or search by keyword* text field, type: **`.airflow.api`**, then press *Enter* on your keyboard.

    c. Select the service endpoint.

    d. Choose your environment's Amazon VPC in **VPC**.

    e. Ensure that your two private subnets in different Availability Zones are selected, and that **Enable DNS name** is enabled.

    f. Choose your environment's Amazon VPC security group(s).

    g. Choose **Full Access** in **Policy**.

    h. Choose **Create endpoint**.
4. Create the first endpoint for the Apache Airflow environment:

    a. Choose **Create Endpoint**.

    b. In the *Filter by attributes or search by keyword* text field, type: **`.airflow.env`**, then press *Enter* on your keyboard.

    c. Select the service endpoint.

    d. Choose your environment's Amazon VPC in **VPC**.

    e. Ensure that your two private subnets in different Availability Zones are selected, and that **Enable DNS name** is enabled.

f. Choose your environment's Amazon VPC security group(s).

g. Choose **Full Access** in **Policy**.

h. Choose **Create endpoint**.

5. Create the second endpoint for Apache Airflow operations:

a. Choose **Create Endpoint**.

b. In the *Filter by attributes or search by keyword* text field, type: **`.airflow.ops`**, then press *Enter* on your keyboard.

c. Select the service endpoint.

d. Choose your environment's Amazon VPC in **VPC**.

e. Ensure that your two private subnets in different Availability Zones are selected, and that **Enable DNS name** is enabled.

f. Choose your environment's Amazon VPC security group(s).

g. Choose **Full Access** in **Policy**.

h. Choose **Create endpoint**.

(Optional) Enable private IP addresses for your Amazon S3 VPC interface endpoint

Amazon S3 **Interface** endpoints don't support private DNS. The S3 endpoint requests still resolves to a *public* IP address. To resolve the S3 address to a *private* IP address, you need to add a private hosted zone in Route 53 for the S3 regional endpoint.

Using Route 53

This section describes the steps to enable private IP addresses for an S3 **Interface** endpoint using Route 53.

1. Create a Private Hosted Zone for your Amazon S3 VPC interface endpoint (such as, s3.eu-west-1.amazonaws.com) and associate it with your Amazon VPC.
2. Create an ALIAS A record for your Amazon S3 VPC interface endpoint (such as, s3.eu-west-1.amazonaws.com) that resolves to your VPC Interface Endpoint DNS name.
3. Create an ALIAS A wildcard record for your Amazon S3 interface endpoint (such as, *.s3.eu-west-1.amazonaws.com) that resolves to the VPC Interface Endpoint DNS name.

VPCs with custom DNS

If your Amazon VPC uses custom DNS routing, you need to make the changes in your DNS resolver (not Route 53, typically an EC2 instance running a DNS server) by creating a CNAME record. For example:

```
Name: s3.us-west-2.amazonaws.com
Type: CNAME
Value:  *.vpce-0f67d23e37648915c-e2q2e2j3.s3.us-west-2.vpce.amazonaws.com
```

Managing your own Amazon VPC endpoints on Amazon MWAA

Amazon MWAA uses Amazon VPC endpoints to integrate with various AWS services necessary to set up an Apache Airflow environment. Managing your own endpoints has two primary use-cases:

1. It means you can create Apache Airflow environments in a shared Amazon VPC when you use an AWS Organizations to manage multiple AWS accounts and share resources.
2. It let's you use more restrictive access policies by narrowing down your permissions to the specific resources that use your endpoints.

If you choose to manage your own VPC endpoints, you are responsible for creating your own endpoints for the environment RDS for PostgreSQL database, and for the environment web server.

For more information about how Amazon MWAA deploys Apache Airflow in the cloud, see the Amazon MWAA architecture diagram.

Creating an environment in a shared Amazon VPC

If you use AWS Organizations to manage multiple AWS accounts that share resources, you can use customer managed VPC endpoints with Amazon MWAA to share environment resources with another account in your organization.

When you configure shared VPC access, the account that owns the main Amazon VPC (*owner*) shares the two private subnets required by Amazon MWAA with other accounts (*participants*) that belong to the same organization. Participant accounts that share those subnets can view, create, modify, and delete environments in the shared Amazon VPC.

Assume you have an account, `Owner`, which acts as the `Root` account in the organization and owns the Amazon VPC resources, and a participant account, `Participant`, a member of the same organization. When `Participant` creates a new Amazon MWAA in Amazon VPC it shares with `Owner`, Amazon MWAA will first create the service VPC resources, then enter a `PENDING` state for up to 72 hours.

After the environment status changes from `CREATING` to `PENDING`, a principal acting on behalf of `Owner` creates the required endpoints. To do this, Amazon MWAA lists the database and web server endpoint in the Amazon MWAA console. You can also call the `GetEnvironment` API action to get the service endpoints.

> **Note**
>
> If the Amazon VPC you use to share resources is a private Amazon VPC, you must still complete the steps described in the section called "Managing access to VPC endpoints". The topic covers setting up a different set of Amazon VPC endpoints related to other AWS services that AWS integrates with, such as Amazon ECR, Amazon ECS, and Amazon SQS. These services are essential in operating, and managing, your Apache Airflow environment in the cloud.

Prerequisites

Before you create an Amazon MWAA environment in a shared VPC, you need the following resources:

- An AWS account, `Owner` to be used as the account that owns the Amazon VPC.
- An AWS Organizations organization unit, `MyOrganization` created as a *root*.
- A second AWS account, `Participant`, under `MyOrganization` to serve the participant account that creates the new environment.

In addition, we recommend that you familiarize yourself with the responsibilities and permissions for owners and participants when sharing resources in Amazon VPC.

Create the Amazon VPC

First, create a new Amazon VPC that the owner and participant accounts will share:

1. Sign in to the console using `Owner`, then, open the AWS CloudFormation console. Use the following template to create a stack. This stack provisions a number of networking resources including a Amazon VPC, and the subnets that the two accounts will share in this scenario.

```
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. After the new Amazon VPC resources have been provisioned, navigate to the AWS Resource Access Manager console, then choose **Create resource share**.
3. Choose the subnets you created in the first step from the list of available subnets you can share with `Participant`.

Create the environment

Complete the following steps to create an Amazon MWAA environment with customer-managed Amazon VPC endpoints.

1. Sign in using `Participant`, and open the Amazon MWAA console. Complete **Step one: Specify details** to specify an Amazon S3 bucket, a DAG folder, and dependencies for your new environment. For more information, see getting started.
2. On the **Configure advanced settings** page, under **Networking**, choose the subnets from the shared Amazon VPC.
3. Under **Endpoint management** choose **CUSTOMER** from the dropdown list.
4. Keep the default for the remaining options on the page, then, choose **Create environment** on the **Review and create** page.

The environment begins in a `CREATING` state, then changes to `PENDING`. When the environment is `PENDING`, write down the **Database endpoint service name** and **Web server endpoint service name** (if you set up a private web server) using the console.

When you create a new environment using the Amazon MWAA console. Amazon MWAA creates a new security group with the required inbound and outbound rules. Write down the security group ID.

In the next section, `Owner` will use the service endpoints and the security group ID to create new Amazon VPC endpoints in the shared Amazon VPC.

Create the Amazon VPC endpoints

Complete the following steps to create the required Amazon VPC endpoints for your environment.

1. Sign in to the AWS Management Console using `Owner`, the open https://console.aws.amazon.com/vpc/.
2. Choose **Security groups** from the left navigation panel, then create a new security group in the shared Amazon VPC using the following inbound, and outbound, rules:

	Type	Protocol	Source type	Source
Inbound	All traffic	All	All	Your environment security group
Outbound	All traffic	All	All	`0.0.0.0/0`

> **Warning**
>
> The `Owner` account must set up a security group in the `Owner` account to allow traffic from the new environment to the shared Amazon VPC. You can do this by creating a new security group in `Owner`, or editing an existing one.

3. Choose **Endpoints**, then create new endpoints for the environment database and the web server (if in private mode) using the endpoint service names from the previous steps. Choose the shared Amazon VPC, the subnets you used for the environment, and the environment's security group.

If successful, the environment will change from `PENDING` back to `CREATING`, then finally to `AVAILABLE`. When it is `AVAILABLE`, you can sign in to the Apache Airflow console.

Shared Amazon VPC Troubleshooting

Use the following reference to resolve issues you encounter when creating environments in a shared Amazon VPC.

Environment in `CREATE_FAILED` after `PENDING` status

- Verify that `Owner` is sharing the subnets with `Participant` using AWS Resource Access Manager.
- Verify that the Amazon VPC endpoints for the database and web server are created in the same subnets associated with the environment.
- Verify that the security group used with your endpoints allows traffic from the security groups used for the environment. The `Owner` account creates rules that reference the security group in `Participant` as *`account-number`*/*`security-group-id`*:.

Type	Protocol	Source type	Source
All traffic	All	All	*123456789012* /*sg-0909e8e81919*

For more information, see Responsibilities and permissions for owners and participants

Environment stuck in `PENDING` status

Verify each VPC endpoint status to ensure it is `Available`. If you configure an environment with a private web server, you must also create an endpoint for the web server. If the environment is stuck in `PENDING`, this might indicate that the private web server endpoint is missing.

Received `The Vpc Endpoint Service '`*`vpce-service-name`*`' does not exist` error

If you see the following error, verify that the account creating the endpoints in the `Owner` account that owns the shared VPC:

```
ClientError: An error occurred (InvalidServiceName) when calling the
 CreateVpcEndpoint operation:

The Vpc Endpoint Service 'vpce-service-name' does not exist
```

Tutorials for Amazon Managed Workflows for Apache Airflow

This guide includes step-by-step tutorials to using and configuring an Amazon Managed Workflows for Apache Airflow environment.

Topics

- Tutorial: Configuring private network access using an AWS Client VPN
- Tutorial: Configuring private network access using a Linux Bastion Host
- Tutorial: Restricting an Amazon MWAA user's access to a subset of DAGs
- Tutorial: Automate managing your own environment endpoints on Amazon MWAA

Tutorial: Configuring private network access using an AWS Client VPN

This tutorial walks you through the steps to create a VPN tunnel from your computer to the Apache Airflow *Web server* for your Amazon Managed Workflows for Apache Airflow environment. To connect to the Internet through a VPN tunnel, you'll first need to create a AWS Client VPN endpoint. Once set up, a Client VPN endpoint acts as a VPN server allowing a secure connection from your computer to the resources in your VPC. You'll then connect to the Client VPN from your computer using the AWS Client VPN for Desktop.

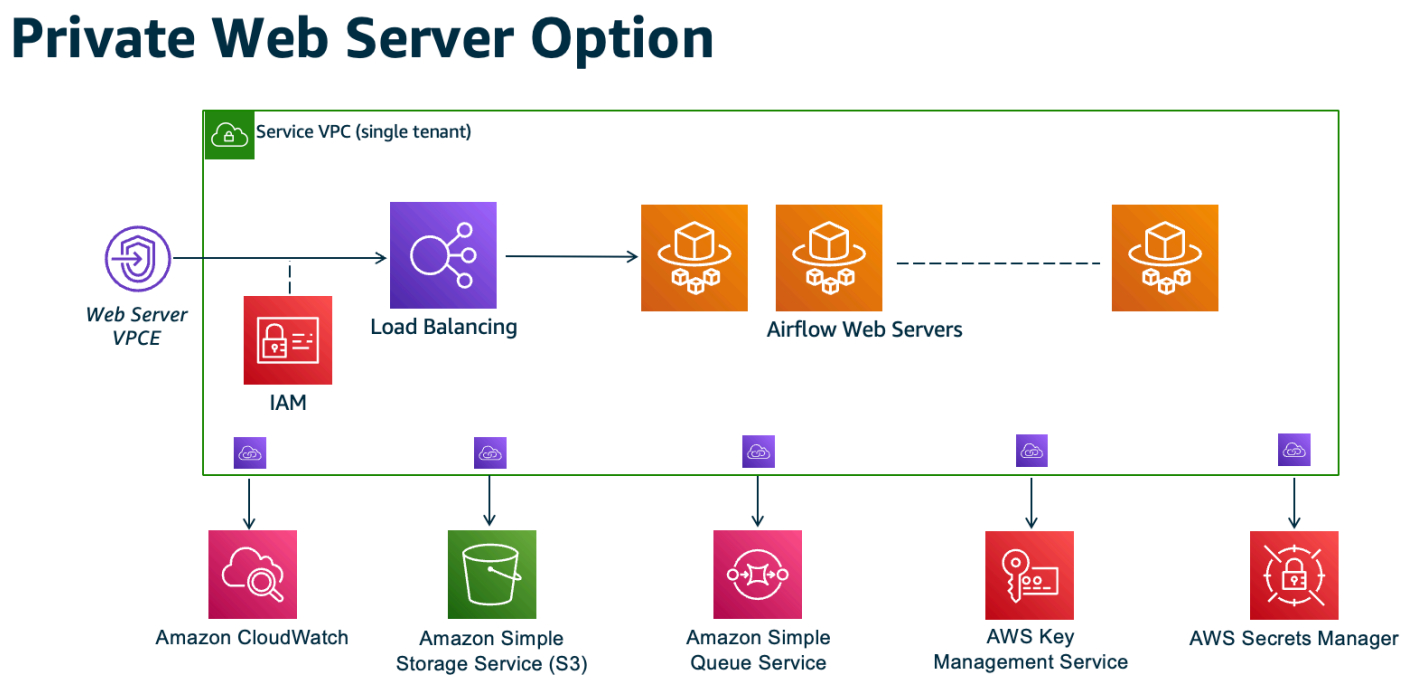
Sections

- Private network
- Use cases
- Before you begin
- Objectives
- (Optional) Step one: Identify your VPC, CIDR rules, and VPC security(s)
- Step two: Create the server and client certificates
- Step three: Save the AWS CloudFormation template locally
- Step four: Create the Client VPN AWS CloudFormation stack
- Step five: Associate subnets to your Client VPN

- Step six: Add an authorization ingress rule to your Client VPN
- Step seven: Download the Client VPN endpoint configuration file
- Step eight: Connect to the AWS Client VPN
- What's next?

Private network

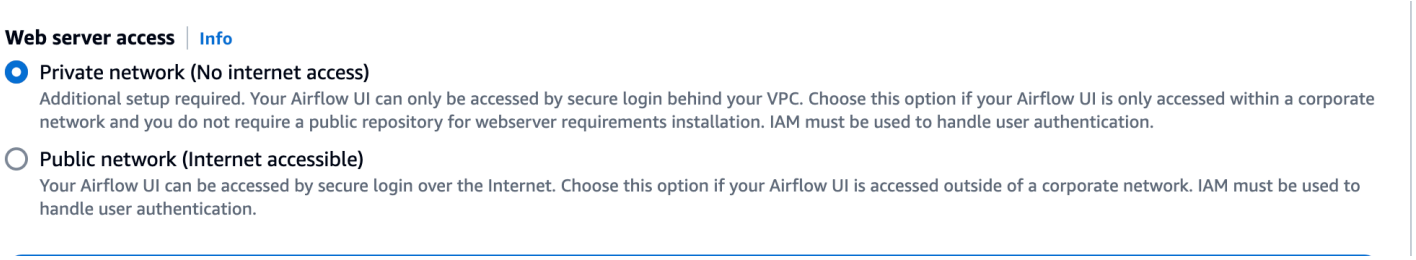
This tutorial assumes you've chosen the **Private network** access mode for your Apache Airflow *Web server*.



The private network access mode limits access to the Apache Airflow UI to users *within your Amazon VPC* that have been granted access to the IAM policy for your environment.

When you create an environment with private web server access, you must package all of your dependencies in a Python wheel archive (`.whl`), then reference the `.whl` in your `requirements.txt`. For instructions on packaging and installing your dependencies using wheel, see Managing dependencies using Python wheel.

The following image shows where to find the **Private network** option on the Amazon MWAA console.



Use cases

You can use this tutorial before or after you've created an Amazon MWAA environment. You must use the same Amazon VPC, VPC security group(s), and private subnets as your environment. If you use this tutorial after you've created an Amazon MWAA environment, once you've completed the steps, you can return to the Amazon MWAA console and change your Apache Airflow *Web server* access mode to **Private network**.

Before you begin

1. Check for user permissions. Be sure that your account in AWS Identity and Access Management (IAM) has sufficient permissions to create and manage VPC resources.
2. Use your Amazon MWAA VPC. This tutorial assumes that you are associating the Client VPN to an existing VPC. The Amazon VPC must be in the same AWS Region as an Amazon MWAA environment and have two private subnets. If you haven't created an Amazon VPC, use the AWS CloudFormation template in Option three: Creating an Amazon VPC network *without* Internet access.

Objectives

In this tutorial, you'll do the following:

1. Create a AWS Client VPN endpoint using a AWS CloudFormation template for an existing Amazon VPC.
2. Generate server and client certificates and keys, and then upload the server certificate and key to AWS Certificate Manager in the same AWS Region as an Amazon MWAA environment.
3. Download and modify a Client VPN endpoint configuration file for your Client VPN, and use the file to create a VPN profile to connect using the Client VPN for Desktop.

(Optional) Step one: Identify your VPC, CIDR rules, and VPC security(s)

The following section describes how to find IDs for your Amazon VPC, VPC security group, and a way to identify the CIDR rules you'll need to create your Client VPN in subsequent steps.

Identify your CIDR rules

The following section shows how to identify the CIDR rules, which you'll need to create your Client VPN.

To identify the CIDR for your Client VPN

1. Open the Your Amazon VPCs page on the Amazon VPC console.
2. Use the region selector in the navigation bar to choose the same AWS Region as an Amazon MWAA environment.
3. Choose your Amazon VPC.
4. Assuming the CIDRs for your private subnets are:

   - Private Subnet 1: 10.192.10.0/24
   - Private Subnet 2: 10.192.11.0/24

   If the CIDR for your Amazon VPC is 10.192.0.0/16, then the **Client IPv4 CIDR** you'd specify for your Client VPN would be 10.192.0.0/22.
5. Save this CIDR value, and the value of your VPC ID for subsequent steps.

Identify your VPC and security group(s)

The following section shows how to find the ID of your Amazon VPC and security group(s), which you'll need to create your Client VPN.

> **Note**
>
> You may be using more than one security group. You'll need to specify all of your VPC's security groups in subsequent steps.

To identify the security group(s)

1. Open the Security Groups page on the Amazon VPC console.
2. Use the region selector in the navigation bar to choose the AWS Region.
3. Look for the Amazon VPC in **VPC ID**, and identify the security groups associated with the VPC.
4. Save the ID of your security group(s) and VPC for subsequent steps.

Step two: Create the server and client certificates

A Client VPN endpoint supports 1024-bit and 2048-bit RSA key sizes only. The following section shows how to use OpenVPN easy-rsa to generate the server and client certificates and keys, and then upload the certificates to ACM using the AWS Command Line Interface (AWS CLI).

To create the client certificates

1. Follow these quick steps to create and upload the certificates to ACM via the AWS CLI in Client authentication and authorization: Mutual authentication.
2. In these steps, you **must** specify the same AWS Region as an Amazon MWAA environment in the AWS CLI command when uploading your server and client certificates. Here's some examples of how to specify the region in these commands:

   a. **Example region for server certificate**

   ```
   aws acm import-certificate --certificate fileb://server.crt --private-key
    fileb://server.key --certificate-chain fileb://ca.crt --region us-west-2
   ```

   b. **Example region for client certificate**

   ```
   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. After these steps, save the value returned in the AWS CLI response for the server certificate and client certificate ARNs. You'll be specifying these ARNs in your AWS CloudFormation template to create the Client VPN.
3. In these steps, a client certificate and a private key are saved to your computer. Here's an example of where to find these credentials:

a. **Example on macOS**

On macOS the contents are saved at `/Users/youruser/custom_folder`. If you list all (`ls -a`) contents of this directory, you should see something similar to the following:

```
.
..
ca.crt
client1.domain.tld.crt
client1.domain.tld.key
server.crt
server.key
```

b. After these steps, save the contents or note the location of the client certificate in `client1.domain.tld.crt`, and the private key in `client1.domain.tld.key`. You'll be adding these values to the configuration file for your Client VPN.

Step three: Save the AWS CloudFormation template locally

The following section contains the AWS CloudFormation template to create the Client VPN. You must specify the same Amazon VPC, VPC security group(s), and private subnets as your Amazon MWAA environment.

- Copy the contents of the following template and save locally as `mwaa_vpn_client.yaml`. You can also download the template.

  Substitute the following values:

  - **YOUR_CLIENT_ROOT_CERTIFICATE_ARN** – The ARN for your **client1.domain.tld** certificate in `ClientRootCertificateChainArn`.
  - **YOUR_SERVER_CERTIFICATE_ARN** – The ARN for your **server** certificate in `ServerCertificateArn`.
  - The Client IPv4 CIDR rule in `ClientCidrBlock`. A CIDR rule of `10.192.0.0/22` is provided.
  - Your Amazon VPC ID in `VpcId`. A VPC of `vpc-010101010101` is provided.
  - Your VPC security group ID(s) in `SecurityGroupIds`. A security group of `sg-0101010101` is provided.

```
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

If you're using more than one security group for your environment, you can specify multiple security groups in the following format:

```
SecurityGroupIds:
      - sg-0112233445566778b
```

```
    - sg-0223344556677889f
```

Step four: Create the Client VPN AWS CloudFormation stack

To create the AWS Client VPN

1. Open the AWS CloudFormation console.
2. Choose **Template is ready**, **Upload a template file**.
3. Choose **Choose file**, and select your `mwaa_vpn_client.yaml` file.
4.
5. Choose **Next**, **Next**.
6. Select the acknowledgement, and then choose **Create stack**.

Step five: Associate subnets to your Client VPN

To associate private subnets to the AWS Client VPN

1. Open the Amazon VPC console.
2. Choose the **Client VPN Endpoints** page.
3. Select your Client VPN, and then choose the **Associations** tab, **Associate**.
4. Choose the following in the dropdown list:
   - Your Amazon VPC in **VPC**.
   - One of your private subnets in **Choose a subnet to associate**.
5. Choose **Associate**.

> **Note**
>
> It takes several minutes for the VPC and subnet to be associated to the Client VPN.

Step six: Add an authorization ingress rule to your Client VPN

You need to add an authorization ingress rule using the CIDR rule for your VPC to your Client VPN. If you want to authorize specific users or groups from your Active Directory Group or SAML-based Identity Provider (IdP), see the Authorization rules in the *Client VPN guide*.

To add the CIDR to the AWS Client VPN

1. Open the Amazon VPC console.
2. Choose the **Client VPN Endpoints** page.
3. Select your Client VPN, and then choose the **Authorization** tab, **Authorize Ingress**.
4. Specify the following:
   - Your Amazon VPC's CIDR rule in **Destination network to enable**. For example:

     ```
     10.192.0.0/16
     ```
   - Choose **Allow access to all users** in **Grant access to**.
   - Enter a descriptive name in **Description**.
5. Choose **Add Authorization rule**.

> **Note**
>
> Depending on the networking components for your Amazon VPC, you may also need to this authorization ingress rule to your network access control list (NACL).

Step seven: Download the Client VPN endpoint configuration file

To download the configuration file

1. Follow these quick steps to download the Client VPN configuration file at Download the Client VPN endpoint configuration file.
2. In these steps, you're asked to prepend a string to your Client VPN endpoint DNS name. Here's an example:

- **Example endpoint DNS name**

  If your Client VPN endpoint DNS name looks like this:

  ```
  remote cvpn-endpoint-0909091212aaee1.prod.clientvpn.us-west-1.amazonaws.com 443
  ```

  You can add a string to identify your Client VPN endpoint like this:

  ```
  remote mwaavpn.cvpn-endpoint-0909091212aaee1.prod.clientvpn.us-
  west-1.amazonaws.com 443
  ```

3. In these steps, you're asked to add the contents of the client certificate between a new set of `<cert></cert>` tags and the contents of the private key between a new set of `<key></key>` tags. Here's an example:

   a. Open a command prompt and change directories to the location of your client certificate and private key.

   b. **Example macOS client1.domain.tld.crt**

   To show the contents of the `client1.domain.tld.crt` file on macOS, you can use `cat client1.domain.tld.crt`.

   Copy the value from terminal and paste in `downloaded-client-config.ovpn` like this:

   ```
   ZZZ1111dddaBBB
   -----END CERTIFICATE-----
   </ca>
   <cert>
   -----BEGIN CERTIFICATE-----
   YOUR client1.domain.tld.crt
   -----END CERTIFICATE-----
   </cert>
   ```

   c. **Example macOS client1.domain.tld.key**

   To show the contents of the `client1.domain.tld.key`, you can use `cat client1.domain.tld.key`.

   Copy the value from terminal and paste in `downloaded-client-config.ovpn` like this:

```
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>
```

Step eight: Connect to the AWS Client VPN

The client for AWS Client VPN is provided free of charge. You can connect your computer directly to AWS Client VPN for an end-to-end VPN experience.

To connect to the Client VPN

1. Download and install the AWS Client VPN for Desktop.
2. Open the AWS Client VPN.
3. Choose **File**, **Managed profiles** in the VPN client menu.
4. Choose **Add profile**, and then choose the `downloaded-client-config.ovpn`.
5. Enter a descriptive name in **Display Name**.
6. Choose **Add profile**, **Done**.
7. Choose **Connect**.

After you connect to the Client VPN, you'll need to disconnect from other VPNs to view any of the resources in your Amazon VPC.

> **Note**
>
> You may need to quit the client, and start again before you're able to get connected.

What's next?

- Learn how to create an Amazon MWAA environment in Get started with Amazon Managed Workflows for Apache Airflow. You must create an environment in the same AWS Region as the Client VPN, and using the same VPC, private subnets, and security group as the Client VPN.

Tutorial: Configuring private network access using a Linux Bastion Host

This tutorial walks you through the steps to create an SSH tunnel from your computer to the to the Apache Airflow *Web server* for your Amazon Managed Workflows for Apache Airflow environment. It assumes you've already created an Amazon MWAA environment. Once set up, a Linux Bastion Host acts as a jump server allowing a secure connection from your computer to the resources in your VPC. You'll then use a SOCKS proxy management add-on to control the proxy settings in your browser to access your Apache Airflow UI.

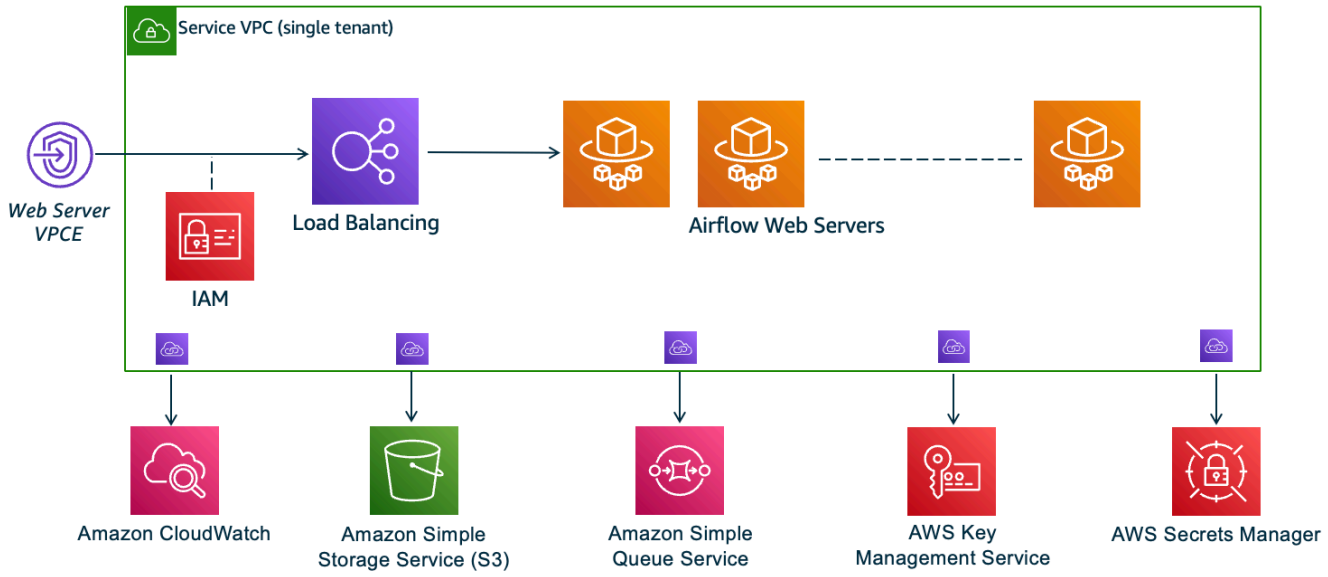
Sections

- Private network
- Use cases
- Before you begin
- Objectives
- Step one: Create the bastion instance
- Step two: Create the ssh tunnel
- Step three: Configure the bastion security group as an inbound rule
- Step four: Copy the Apache Airflow URL
- Step five: Configure proxy settings
- Step six: Open the Apache Airflow UI
- What's next?

Private network

This tutorial assumes you've chosen the **Private network** access mode for your Apache Airflow *Web server*.

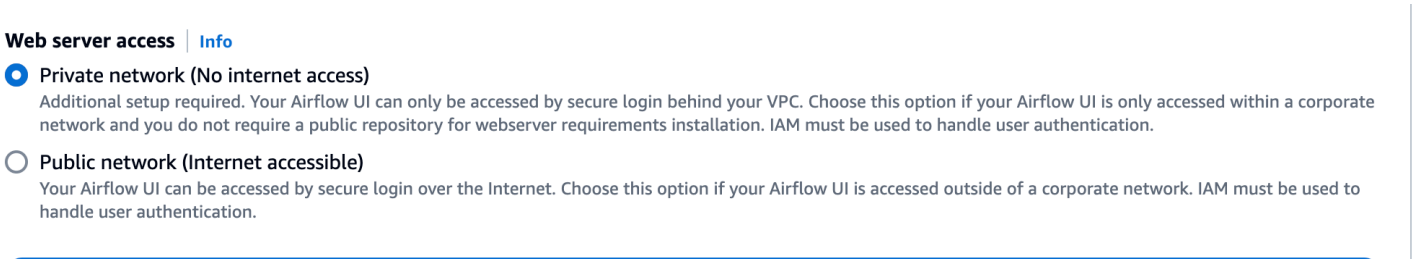
Private Web Server Option



The private network access mode limits access to the Apache Airflow UI to users *within your Amazon VPC* that have been granted access to the IAM policy for your environment.

When you create an environment with private web server access, you must package all of your dependencies in a Python wheel archive (`.whl`), then reference the `.whl` in your `requirements.txt`. For instructions on packaging and installing your dependencies using wheel, see Managing dependencies using Python wheel.

The following image shows where to find the **Private network** option on the Amazon MWAA console.



Use cases

You can use this tutorial after you've created an Amazon MWAA environment. You must use the same Amazon VPC, VPC security group(s), and public subnets as your environment.

Before you begin

1. Check for user permissions. Be sure that your account in AWS Identity and Access Management (IAM) has sufficient permissions to create and manage VPC resources.
2. Use your Amazon MWAA VPC. This tutorial assumes that you are associating the bastion host to an existing VPC. The Amazon VPC must be in the same region as your Amazon MWAA environment and have two private subnets, as defined in Create the VPC network.
3. Create an SSH key. You need to create an Amazon EC2 SSH key (**.pem**) in the same Region as your Amazon MWAA environment to connect to the virtual servers. If you don't have an SSH key, see Create or import a key pair in the *Amazon EC2 User Guide*.

Objectives

In this tutorial, you'll do the following:

1. Create a Linux Bastion Host instance using a AWS CloudFormation template for an existing VPC.
2. Authorize inbound traffic to the bastion instance's security group using an ingress rule on port 22.
3. Authorize inbound traffic from an Amazon MWAA environment's security group to the bastion instance's security group.
4. Create an SSH tunnel to the bastion instance.
5. Install and configure the FoxyProxy add-on for the Firefox browser to view the Apache Airflow UI.

Step one: Create the bastion instance

The following section describes the steps to create the linux bastion instance using a AWS CloudFormation template for an existing VPC on the AWS CloudFormation console.

To create the Linux Bastion Host

1. Open the Deploy Quick Start page on the AWS CloudFormation console.
2. Use the region selector in the navigation bar to choose the same AWS Region as your Amazon MWAA environment.

3. Choose **Next**.
4. Type a name in the **Stack name** text field, such as `mwaa-linux-bastion`.
5. On the **Parameters**, **Network configuration** pane, choose the following options:
   a. Choose your Amazon MWAA environment's **VPC ID**.
   b. Choose your Amazon MWAA environment's **Public subnet 1 ID**.
   c. Choose your Amazon MWAA environment's **Public subnet 2 ID**.
   d. Enter the narrowest possible address range (for example, an internal CIDR range) in **Allowed bastion external access CIDR**.

   > **Note**
   >
   > The simplest way to identify a range is to use the same CIDR range as your public subnets. For example, the public subnets in the AWS CloudFormation template on the Create the VPC network page are `10.192.10.0/24` and `10.192.11.0/24`.

6. On the **Amazon EC2 configuration** pane, choose the following:
   a. Choose your SSH key in the dropdown list in **Key pair name**.
   b. Enter a name in **Bastion Host Name**.
   c. Choose **true** for **TCP forwarding**.

   > **Warning**
   >
   > TCP forwarding must be set to **true** in this step. Otherwise, you won't be able to create an SSH tunnel in the next step.

7. Choose **Next**, **Next**.
8. Select the acknowledgement, and then choose **Create stack**.

To learn more about the architecture of your Linux Bastion Host, see Linux Bastion Hosts on the AWS Cloud: Architecture.

Step two: Create the ssh tunnel

The following steps describe how to create the ssh tunnel to your linux bastion. An SSH tunnel recieves the request from your local IP address to the linux bastion, which is why TCP forwarding for the linux bastion was set to `true` in previous steps.

macOS/Linux

To create a tunnel via command line

1. Open the Instances page on the Amazon EC2 console.
2. Choose an instance.
3. Copy the address in **Public IPv4 DNS**. For example, `ec2-4-82-142-1.compute-1.amazonaws.com`.
4. In your command prompt, navigate to the directory where your SSH key is stored.
5. Run the following command to connect to the bastion instance using ssh. Substitute the sample value with your SSH key name in `mykeypair.pem`.

```
ssh -i mykeypair.pem -N -D 8157 ec2-user@YOUR_PUBLIC_IPV4_DNS
```

Windows (PuTTY)

To create a tunnel using PuTTY

1. Open the Instances page on the Amazon EC2 console.
2. Choose an instance.
3. Copy the address in **Public IPv4 DNS**. For example, `ec2-4-82-142-1.compute-1.amazonaws.com`.
4. Open PuTTY, select **Session**.
5. Enter the host name in **Host Name** as ec2-user@*YOUR_PUBLIC_IPV4_DNS* and the **port** as 22.
6. Expand the **SSH** tab, select **Auth**. In **Private Key file for authentication**, choose your local "ppk" file.
7. Under SSH, choose the **Tunnels** tab, and then select the *Dynamic* and *Auto* options.

8. In **Source Port**, add the `8157` port (or any other unused port), and then leave the **Destination** port blank. Choose **Add**.
9. Choose the **Session** tab and enter a session name. For example `SSH Tunnel`.
10. Choose **Save**, **Open**.

> **Note**
>
> You may need to enter a pass phrase for your public key.

> **Note**
>
> If you receive a `Permission denied (publickey)` error, we recommend using the AWSSupport-TroubleshootSSH tool, and choose **Run this Automation (console)** to troubleshoot your SSH setup.

Step three: Configure the bastion security group as an inbound rule

Access to the servers and regular internet access from the servers is allowed with a special maintenance security group attached to those servers. The following steps describe how to configure the bastion security group as an inbound source of traffic to an environment's VPC security group.

1. Open the Environments page on the Amazon MWAA console.
2. Choose an environment.
3. On the **Networking** pane, choose **VPC security group**.
4. Choose **Edit inbound rules**.
5. Choose **Add rule**.
6. Choose your VPC security group ID in the **Source** dropdown list.
7. Leave the remaining options blank, or set to their default values.
8. Choose **Save rules**.

Step four: Copy the Apache Airflow URL

The following steps describe how to open the Amazon MWAA console and copy the URL to the Apache Airflow UI.

1. Open the Environments page on the Amazon MWAA console.
2. Choose an environment.
3. Copy the URL in **Airflow UI** for subsequent steps.

Step five: Configure proxy settings

If you use an SSH tunnel with dynamic port forwarding, you must use a SOCKS proxy management add-on to control the proxy settings in your browser. For example, you can use the `--proxy-server` feature of Chromium to kick off a browser session, or use the FoxyProxy extension in the Mozilla FireFox browser.

Option one: Setup an SSH Tunnel using local port forwarding

If you do not wish to use a SOCKS proxy, you can set up an SSH tunnel using local port forwarding. The following example command accesses the Amazon EC2 *ResourceManager* web interface by forwarding traffic on local port 8157.

1. Open a new command prompt window.
2. Type the following command to open an SSH tunnel.

```
ssh -i mykeypair.pem -N -L 8157:YOUR_VPC_ENDPOINT_ID-
vpce.YOUR_REGION.airflow.amazonaws.com:443
 ubuntu@YOUR_PUBLIC_IPV4_DNS.YOUR_REGION.compute.amazonaws.com
```

   `-L` signifies the use of local port forwarding which allows you to specify a local port used to forward data to the identified remote port on the node's local web server.

3. Type `http://localhost:8157/` in your browser.

> **Note**
>
> You may need to use `https://localhost:8157/`.

Option two: Proxies via command line

Most web browsers allow you to configure proxies via a command line or configuration parameter. For example, with Chromium you can start the browser with the following command:

```
chromium --proxy-server="socks5://localhost:8157"
```

This starts a browser session which uses the ssh tunnel you created in previous steps to proxy its requests. You can open your Private Amazon MWAA environment URL (with *https://*) as follows:

```
https://YOUR_VPC_ENDPOINT_ID-vpce.YOUR_REGION.airflow.amazonaws.com/home.
```

Option three: Proxies using FoxyProxy for Mozilla Firefox

The following example demonstrates a FoxyProxy Standard (version 7.5.1) configuration for Mozilla Firefox. FoxyProxy provides a set of proxy management tools. It lets you use a proxy server for URLs that match patterns corresponding to domains used by the Apache Airflow UI.

1. In Firefox, open the FoxyProxy Standard extension page.
2. Choose **Add to Firefox**.
3. Choose **Add**.
4. Choose the FoxyProxy icon in your browser's toolbar, choose **Options**.
5. Copy the following code and save locally as `mwaa-proxy.json`. Substitute the sample value in *YOUR_HOST_NAME* with your **Apache Airflow URL**.

```
{
  "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. On the **Import Settings from FoxyProxy 6.0+** pane, choose **Import Settings** and select the `mwaa-proxy.json` file.
7. Choose **OK**.

Step six: Open the Apache Airflow UI

The following steps describe how to open your Apache Airflow UI.

1. Open the Environments page on the Amazon MWAA console.
2. Choose **Open Airflow UI**.

What's next?

- Learn how to run Airflow CLI commands on an SSH tunnel to a bastion host in Apache Airflow CLI command reference.
- Learn how to upload DAG code to your Amazon S3 bucket in Adding or updating DAGs.

Tutorial: Restricting an Amazon MWAA user's access to a subset of DAGs

Amazon MWAA manages access to your environment by mapping your IAM principals to one or more of Apache Airflow's default roles. The following tutorial shows how you can restrict individual Amazon MWAA users to only view and interact with a specific DAG or a set of DAGs.

> **Note**
>
> The steps in this tutorial can be completed using federated access, as long as the IAM roles can be assumed.

Topics

- Prerequisites
- Step one: Provide Amazon MWAA web server access to your IAM principal with the default Public Apache Airflow role.
- Step two: Create a new Apache Airflow custom role
- Step three: Assign the role you created to your Amazon MWAA user
- Next steps
- Related resources

Prerequisites

To complete the steps in this tutorial, you'll need the following:

- An Amazon MWAA environment with multiple DAGs
- An IAM principal, `Admin` with AdministratorAccess permissions, and an IAM user, `MWAAUser`, as the principal for which you can limit DAG access. For more information about admin roles, see Administrator job function in the *IAM User Guide*

  > **Note**
  >
  > Do not attach permission policies directly to your IAM users. We recommend setting up IAM roles that users can assume to gain temporary access to your Amazon MWAA resources.

- AWS Command Line Interface version 2 installed.

Step one: Provide Amazon MWAA web server access to your IAM principal with the default `Public` Apache Airflow role.

To grant permission using the AWS Management Console

1. Sign in to your AWS account with an `Admin` role and open the IAM console.
2. In the left navigation pane, choose **Users**, then choose your Amazon MWAA IAM user from the users table.
3. On the user details page, under **Summary**, choose the **Permissions** tab, then choose **Permissions policies** to expand the card and choose **Add permissions**.
4. In the **Grant permissions** section, choose **Attach existing policies directly**, then choose **Create policy** to create and attach your own custom permissions policy.
5. On the **Create policy** page, choose **JSON**, then copy and paste the following JSON permissions policy in the policy editor. Tha policy grants web server access to the user with the default `Public` Apache Airflow role.

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

```
        "Effect": "Allow",
        "Action": "airflow:CreateWebLoginToken",
        "Resource": [

"arn:aws:airflow:YOUR_REGION:YOUR_ACCOUNT_ID:role/YOUR_ENVIRONMENT_NAME/Public"
            ]
        }
    ]
}
```

Step two: Create a new Apache Airflow custom role

To create a new role using the Apache Airflow UI

1. Using your administrator IAM role, open the Amazon MWAA console and launch your environment's Apache Airflow UI.
2. From the navigation pane at the top, hover on **Security** to open the dropdown list, then choose **List Roles** to view the default Apache Airflow roles.
3. From the roles list, select **User**, then at the top of the page choose **Actions** to open the dropdown. Choose **Copy Role**, and confirm **Ok**

> **Note**
>
> Copy the **Ops** or **Viewer** roles to grant more or less access, respectively.

4. Locate the new role you created in the table and choose **Edit record**.
5. On the **Edit Role** page, do the following:

   - For **Name**, type a new name for the role in the text field. For example, **`Restricted`**.
   - For the list of **Permissions**, remove `can read on DAGs` and `can edit on DAGs`, then add read and write permissions for the set of DAGs you want to provide access to. For example, for a DAG, `example_dag.py`, add **`can read on DAG:example_dag`** and **`can edit on DAG:example_dag`**.

   Choose **Save**. You should now have a new role that limits access to a subset of DAGs available in your Amazon MWAA environment. You can now assign this role to any existing Apache Airflow users.

Step three: Assign the role you created to your Amazon MWAA user

To assign the new role

1. Using access credentials for `MWAAUser`, run the following CLI command to retrieve your environment's web server URL.

   ```
   $ aws mwaa get-environment --name YOUR_ENVIRONMENT_NAME | jq
    '.Environment.WebserverUrl'
   ```

   If successful, you'll see the following output:

   ```
   "ab1b2345-678a-90a1-a2aa-34a567a8a901.c13.us-west-2.airflow.amazonaws.com"
   ```

2. With `MWAAUser` signed in to the AWS Management Console, open a new browser window and access the following URl. Replace `Webserver-URL` with your information.

   ```
   https://<Webserver-URL>/home
   ```

   If successful, you'll see a `Forbidden` error page because `MWAAUser` has not been granted permission to access the Apache Airflow UI yet.

3. With `Admin` signed in to the AWS Management Console, open the Amazon MWAA console again and launch your environment's Apache Airflow UI.
4. From the UI dashboard, expand the **Security** dropdown, and this time choose **List Users**.
5. In the users table, find the new Apache Airflow user and choose **Edit record**. The user's first name will match your IAM user name in the following pattern: `user/mwaa-user`.
6. On the **Edit User** page, in the **Role** section, add the new custom role you created, then choose **Save**.

   > **Note**
   >
   > The **Last Name** field is required, but a space satisfies the requirement.

   The IAM `Public` principal grants the `MWAAUser` permission to access the Apache Airflow UI, while the new role provides the additional permissions needed to see their DAGs.

Important

Any of the 5 default roles (such as `Admin`) not authorized by IAM which are added using the Apache Airflow UI will be removed on next user login.

Next steps

- To learn more about managing access to your Amazon MWAA environment, and to see sample JSON IAM policies you can use for your environment users, see the section called "Accessing an Amazon MWAA environment"

Related resources

- Access Control (Apache Airflow Documentation) – Learn more about the default Apache Airflow roles on the Apache Airflow documentation website.

Tutorial: Automate managing your own environment endpoints on Amazon MWAA

If you use AWS Organizations to manage multiple AWS accounts that share resources, Amazon MWAA lets you create, and manage, your own Amazon VPC endpoints. This means you can use stricter security policies that allow access only the resources required by your environment.

When you create an environment in a shared Amazon VPC, the account that owns the main Amazon VPC (*owner*) shares the two private subnets required by Amazon MWAA with other accounts (*participants*) that belong to the same organization. Participant accounts that share those subnets can then view, create, modify, and delete environments in the shared VPC.

When you create an environment in a shared, or otherwise policy-restricted, Amazon VPC, Amazon MWAA will first create the service VPC resources, then enter a `PENDING` state for up to 72 hours.

When the environment status changes from `CREATING` to `PENDING`, Amazon MWAA sends an Amazon EventBridge notification of the change in state. This lets the owner account create the required endpoints on behalf of participants based on endpoint service information from the Amazon MWAA console or API, or programmatically In the following, we create new Amazon VPC

endpoints using an Lambda function and an EventBridge rule that listens to Amazon MWAA state change notifications.

Here, we create the new endpoints in the same Amazon VPC as the environment. To set up a shared Amazon VPC, create the EventBridge rule and Lambda function would in the owner account, and the Amazon MWAA environment in the participant account.

Topics

- Prerequisites
- Create the Amazon VPC
- Create the Lambda function
- Create the EventBridge rule
- Create the Amazon MWAA environment

Prerequisites

To complete the steps in this tutorial, you will need the following:

- ...

Create the Amazon VPC

Use the following AWS CloudFormation template and AWS CLI command to create a new Amazon VPC. The template sets up the Amazon VPC resources and modifies the endpoint policy to restrict access to a specific queue.

1. Download the AWS CloudFormation template, then unzip the `.yml` file.
2. In a new command prompt window, navigate to the folder where you saved the template, then use `create-stack` to create the stack. The `--template-body` flag specifies the path to the template.

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

In the next section, you'll create the Lambda function.

Create the Lambda function

Use the following Python code and IAM JSON policy to create a new Lambda function and execution role. This function creates Amazon VPC endpoints for a private Apache Airflow web server and an Amazon SQS queue. Amazon MWAA uses Amazon SQS to queue tasks with Celery among multiple workers when scaling your environment.

1. Download the Python function code.
2. Download the IAM permission policy, then unzip the file.
3. Open a command prompt, then navigate to the folder where you saved the JSON permission policy. Use the IAM `create-role` command to create the new role.

   ```
   $ aws iam create-role --role-name function-role \
   --assume-role-policy-document file://lambda-mwaa-vpce-policy.json
   ```

   Note the role ARN from the AWS CLI response. In the next step, we specify this new role as the function's execution role using its ARN.
4. Navigate to the folder where you saved the function code, then use the`create-function` command to create a new function.

   ```
   $ 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
   ```

   Note the function ARN from the AWS CLI response. In the next step we specify the ARN to configure the function as a target for a new EventBridge rule.

In the next section, you will create the EventBridge rule that invokes this function when the environment enters a `PENDING` state.

Create the EventBridge rule

Do the following to create a new rule that listens for Amazon MWAA notifications and targets your new Lambda function.

1. Use the EventBridge `put-rule` command to create a new EventBridge rule.

   ```
   $ aws events put-rule --name "mwaa-lambda-rule" \
   ```

```
--event-pattern "{\"source\":[\"aws.airflow\"],\"detail-type\":[\"MWAA Environment
 Status Change\"]}"
```

The event pattern listens for notifications that Amazon MWAA sends whenever an environment status changes.

```
{
  "source": ["aws.airflow"],
  "detail-type": ["MWAA Environment Status Change"]
}
```

2. Use the `put-targets` command to add the Lambda function as a target for the new rule.

```
$ aws events put-targets --rule "mwaa-lambda-rule" \
--targets "Id"="1","Arn"="arn:aws::lambda:region:123456789012:function:mwaa-vpce-
lambda"
```

You're ready to create a new Amazon MWAA environment with customer-managed Amazon VPC endpoints.

Create the Amazon MWAA environment

Use the Amazon MWAA console to create a new environment with customer-managed Amazon VPC endpoints.

1. Open the Amazon MWAA console, and choose **Create an environment**.
2. For **Name** enter a unique name.
3. For **Airflow version** choose the latest version.
4. Choose an **Amazon S3 bucket** and a **DAGs folder**, such as `dags/` to use with the environment, then choose **Next**.
5. On the **Configure advanced settings** page, do the following:
   a. For **Virtual Private Cloud**, choose the Amazon VPC you created in the previous step.
   b. For **Web server access**, choose **Public network (Internet accessible)**.
   c. For **Security groups**, choose the security group you created with AWS CloudFormation. Because the security groups for the AWS PrivateLink endpoints from the earlier step are self-referencing, you must choose the same security group for your environment.

   d. For **Endpoint management**, choose **Customer managed endpoints**.
6. Keep the remaining default settings, then choose **Next**.
7. Review your selections, then choose **Create environment**.

> **Tip**
>
> For more information about setting up a new environment, see Getting started with Amazon MWAA.

When the environment is `PENDING`, Amazon MWAA sends a notification that matches the event pattern you set for your rule. The rule invokes your Lambda function. The function parses the notification event and gets the required endpoint information for the web server and the Amazon SQS queue. It then creates the endpoints in your Amazon VPC.

When the endpoints are available, Amazon MWAA resumes creating your environment. When ready, the environment status changes to `AVAILABLE` and you can access the Apache Airflow web server using the Amazon MWAA console.

Code examples for Amazon Managed Workflows for Apache Airflow

This guide contains code samples, including DAGs and custom plugins, that you can use on an Amazon Managed Workflows for Apache Airflow environment. For more examples of using Apache Airflow with AWS services, see the `dags` directory in the Apache Airflow GitHub repository.

Samples

- Using a DAG to import variables in the CLI
- Creating an SSH connection using the SSHOperator
- Using a secret key in AWS Secrets Manager for an Apache Airflow Snowflake connection
- Using a DAG to write custom metrics in CloudWatch
- Aurora PostgreSQL database cleanup on an Amazon MWAA environment
- Exporting environment metadata to CSV files on Amazon S3
- Using a secret key in AWS Secrets Manager for an Apache Airflow variable
- Using a secret key in AWS Secrets Manager for an Apache Airflow connection
- Creating a custom plugin with Oracle
- Creating a custom plugin that generates runtime environment variables
- Changing a DAG's timezone on Amazon MWAA
- Refreshing a CodeArtifact token
- Creating a custom plugin with Apache Hive and Hadoop
- Creating a custom plugin for Apache Airflow PythonVirtualenvOperator
- Invoking DAGs with a Lambda function
- Invoking DAGs in different Amazon MWAA environments
- Using Amazon MWAA with Amazon RDS for Microsoft SQL Server
- Using Amazon MWAA with Amazon EMR
- Using Amazon MWAA with Amazon EKS
- Connecting to Amazon ECS using the ECSOperator
- Using dbt with Amazon MWAA
- AWS blogs and tutorials

Using a DAG to import variables in the CLI

The following sample code imports variables using the CLI on Amazon Managed Workflows for Apache Airflow.

Topics

- Version
- Prerequisites
- Permissions
- Dependencies
- Code sample
- What's next?

Version

- You can use the code example on this page with **Apache Airflow v2** in Python 3.10.

Prerequisites

- No additional permissions are required to use the code example on this page.

Permissions

Your AWS account needs access to the `AmazonMWAAAirflowCliAccess` policy. To learn more, see Apache Airflow CLI policy: AmazonMWAAAirflowCliAccess.

Dependencies

- To use this code example with Apache Airflow v2, no additional dependencies are required. The code uses the Apache Airflow v2 base install on your environment.

Code sample

The following sample code takes three inputs: your Amazon MWAA environment name (in `mwaa_env`), the AWS Region of your environment (in `aws_region`), and the local file that contains the variables you want to import (in `var_file`).

```
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)
```

What's next?

- Learn how to upload the DAG code in this example to the `dags` folder in your Amazon S3 bucket in Adding or updating DAGs.

Creating an SSH connection using the `SSHOperator`

The following example describes how you can use the `SSHOperator` in a directed acyclic graph (DAG) to connect to a remote Amazon EC2 instance from your Amazon Managed Workflows for

Apache Airflow environment. You can use a similar approach to connect to any remote instance with SSH access.

In the following example, you upload a SSH secret key (`.pem`) to your environment's `dags` directory on Amazon S3. Then, you install the necessary dependencies using `requirements.txt` and create a new Apache Airflow connection in the UI. Finally, you write a DAG that creates an SSH connection to the remote instance.

Topics

- Version
- Prerequisites
- Permissions
- Requirements
- Copy your secret key to Amazon S3
- Create a new Apache Airflow connection
- Code sample

Version

- You can use the code example on this page with **Apache Airflow v2** in Python 3.10.

Prerequisites

To use the sample code on this page, you'll need the following:

- An Amazon MWAA environment.
- An SSH secret key. The code sample assumes you have an Amazon EC2 instance and a `.pem` in the same Region as your Amazon MWAA environment. If you don't have a key, see Create or import a key pair in the *Amazon EC2 User Guide*.

Permissions

- No additional permissions are required to use the code example on this page.

Requirements

Add the following parameter to `requirements.txt` to install the `apache-airflow-providers-ssh` package on the web server. Once your environment updates and Amazon MWAA successfully installs the dependency, you will see a new **SSH** connection type in the UI.

```
-c https://raw.githubusercontent.com/apache/airflow/constraints-Airflow-version/
constraints-Python-version.txt
apache-airflow-providers-ssh
```

> **Note**
>
> `-c` defines the constraints URL in `requirements.txt`. This ensures that Amazon MWAA installs the correct package version for your environemnt.

Copy your secret key to Amazon S3

Use the following AWS Command Line Interface command to copy your `.pem` key to your environment's `dags` directory in Amazon S3.

```
$ aws s3 cp your-secret-key.pem s3://your-bucket/dags/
```

Amazon MWAA copies the content in `dags`, including the `.pem` key, to the local `/usr/local/airflow/dags/` directory, By doing this, Apache Airflow can access the key.

Create a new Apache Airflow connection

To create a new SSH connection using the Apache Airflow UI

1. Open the Environments page on the Amazon MWAA console.
2. From the list of environments, choose **Open Airflow UI** for your environment.
3. On the Apache Airflow UI page, choose **Admin** from the top navigation bar to expand the dropdown list, then choose **Connections**.
4. On the **List Connections** page, choose **+**, or **Add a new record** button to add a new connection.
5. On the **Add Connection** page, add the following information:

a. For **Connection Id**, enter **ssh_new**.

b. For **Connection Type**, choose **SSH** from the dropdown list.

> **Note**
>
> If the **SSH** connection type is not available in the list, Amazon MWAA hasn't installed the required `apache-airflow-providers-ssh` package. Update your `requirements.txt` file to include this package, then try again.

c. For **Host**, enter the IP address for the Amazon EC2 instance that you want to connect to. For example, **12.345.67.89**.

d. For **Username**, enter **ec2-user** if you are connecting to an Amazon EC2 instance. Your username might be different, depending on the type of remote instance you want Apache Airflow to connect to.

e. For **Extra**, enter the following key-value pair in JSON format:

```
{ "key_file": "/usr/local/airflow/dags/your-secret-key.pem" }
```

This key-value pair instructs Apache Airflow to look for the secret key in the local `/dags` directory.

Code sample

The following DAG uses the `SSHOperator` to connect to your target Amazon EC2 instance, then runs the `hostname` Linux command to print the name of the instance. You can modify the DAG to run any command or script on the remote instance.

1. Open a terminal, and navigate to the directory where your DAG code is stored. For example:

```
cd dags
```

2. Copy the contents of the following code sample and save locally as `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. Run the following AWS CLI command to copy the DAG to your environment's bucket, then trigger the DAG using the Apache Airflow UI.

```
$ aws s3 cp your-dag.py s3://your-environment-bucket/dags/
```

4. If successful, you'll see output similar to the following in the task logs for `ssh_task` in the `ssh_operator_example` DAG:

```
[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
```

Using a secret key in AWS Secrets Manager for an Apache Airflow Snowflake connection

The following sample calls AWS Secrets Manager to get a secret key for an Apache Airflow Snowflake connection on Amazon Managed Workflows for Apache Airflow. It assumes you've completed the steps in Configuring an Apache Airflow connection using a AWS Secrets Manager secret.

Topics

- Version
- Prerequisites
- Permissions
- Requirements
- Code sample
- What's next?

Version

- You can use the code example on this page with **Apache Airflow v2** in Python 3.10.

Prerequisites

To use the sample code on this page, you'll need the following:

- The Secrets Manager backend as an Apache Airflow configuration option as shown in Configuring an Apache Airflow connection using a AWS Secrets Manager secret.
- An Apache Airflow connection string in Secrets Manager as shown in Configuring an Apache Airflow connection using a AWS Secrets Manager secret.

Permissions

- Secrets Manager permissions as shown in Configuring an Apache Airflow connection using a AWS Secrets Manager secret.

Requirements

To use the sample code on this page, add the following dependencies to your `requirements.txt`. To learn more, see Installing Python dependencies.

```
apache-airflow-providers-snowflake==1.3.0
```

Code sample

The following steps describe how to create the DAG code that calls Secrets Manager to get the secret.

1. In your command prompt, navigate to the directory where your DAG code is stored. For example:

   ```
   cd dags
   ```

2. Copy the contents of the following code sample and save locally as `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",
    )
```

What's next?

- Learn how to upload the DAG code in this example to the `dags` folder in your Amazon S3 bucket in Adding or updating DAGs.

Using a DAG to write custom metrics in CloudWatch

You can use the following code example to write a directed acyclic graph (DAG) that runs a `PythonOperator` to retrieve OS-level metrics for an Amazon MWAA environment. The DAG then publishes the data as custom metrics to Amazon CloudWatch.

Custom OS-level metrics provide you with additional visibility about how your environment workers are utilizing resources such as virtual memory and CPU. You can use this information to select the environment class that best suits your workload.

Topics

- Version
- Prerequisites
- Permissions
- Dependencies
- Code example

Version

- You can use the code example on this page with **Apache Airflow v2** in Python 3.10.

Prerequisites

To use the code example on this page, you need the following:

- An Amazon MWAA environment.

Permissions

- No additional permissions are required to use the code example on this page.

Dependencies

- No additional dependencies are required to use the code example on this page.

Code example

1. In your command prompt, navigate to the folder where your DAG code is stored. For example:

```
cd dags
```

2. Copy the contents of the following code example and save it locally as `dag-custom-metrics.py`. Replace `MWAA-ENV-NAME` with your environment 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. Run the following AWS CLI command to copy the DAG to your environment's bucket, then trigger the DAG using the Apache Airflow UI.

```
$ aws s3 cp your-dag.py s3://your-environment-bucket/dags/
```

4. If the DAG runs successfully, you should see something similar to the following in your Apache Airflow logs:

```
[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
```

Aurora PostgreSQL database cleanup on an Amazon MWAA environment

Amazon Managed Workflows for Apache Airflow uses an Aurora PostgreSQL database as the Apache Airflow metadata database, where DAG runs and task instances are stored. The following sample code periodically clears out entries from the dedicated Aurora PostgreSQL database for your Amazon MWAA environment.

Topics

- Version
- Prerequisites
- Dependencies
- Code sample

Version

- You can use the code example on this page with **Apache Airflow v2** in Python 3.10.

Prerequisites

To use the sample code on this page, you'll need the following:

- An Amazon MWAA environment.

Dependencies

- To use this code example with Apache Airflow v2, no additional dependencies are required. The code uses the Apache Airflow v2 base install on your environment.

Code sample

The following DAG cleans the metadata database for the tables specified in `TABLES_TO_CLEAN`. The example deletes data from the specified tables that is older than 30 days. To adjust how far back the entries are deleted, set `MAX_AGE_IN_DAYS` to a different value.

Apache Airflow v2.4 and later

```
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 may 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
    )
```

Apache Airflow v2.2 and earlier

```
from airflow import settings
from airflow.utils.dates import days_ago
from airflow.models import DagTag, DagModel, DagRun, ImportError, Log, SlaMiss,
 RenderedTaskInstanceFields, TaskInstance, TaskReschedule, XCom
from airflow.decorators import dag, task
from airflow.utils.dates import days_ago
from time import sleep
```

```
from airflow.version import version
major_version, minor_version = int(version.split('.')[0]), int(version.split('.')
[1])
if major_version >= 2 and minor_version >= 6:
    from airflow.jobs.job import Job
else:
    # The BaseJob class was renamed as of Apache Airflow v2.6
    from airflow.jobs.base_job import BaseJob as Job

# Delete entries for the past 30 days. Adjust MAX_AGE_IN_DAYS to set how far back
 this DAG cleans the database.
MAX_AGE_IN_DAYS = 30
MIN_AGE_IN_DAYS = 0
DECREMENT = -7

# This is a list of (table, time) tuples.
# table = the table to clean in the metadata database
# time  = the column in the table associated to the timestamp of an entry
#         or None if not applicable.
TABLES_TO_CLEAN = [[Job, Job.latest_heartbeat],
    [TaskInstance, TaskInstance.execution_date],
    [TaskReschedule, TaskReschedule.execution_date],
    [DagTag, None],
    [DagModel, DagModel.last_parsed_time],
    [DagRun, DagRun.execution_date],
    [ImportError, ImportError.timestamp],
    [Log, Log.dttm],
    [SlaMiss, SlaMiss.execution_date],
    [RenderedTaskInstanceFields, RenderedTaskInstanceFields.execution_date],
    [XCom, XCom.execution_date],
]

@task()
def cleanup_db_fn(x):
    session = settings.Session()

    if x[1]:
        for oldest_days_ago in range(MAX_AGE_IN_DAYS, MIN_AGE_IN_DAYS, DECREMENT):
            earliest_days_ago = max(oldest_days_ago + DECREMENT, MIN_AGE_IN_DAYS)
            print(f"deleting {str(x[0])} entries between {earliest_days_ago} and
 {oldest_days_ago} days old...")
            earliest_date = days_ago(earliest_days_ago)
            oldest_date = days_ago(oldest_days_ago)
```

```
            query = session.query(x[0]).filter(x[1] >= earliest_date).filter(x[1] <=
 oldest_date)
            query.delete(synchronize_session= False)
            session.commit()
            sleep(5)
    else:
        # No time column specified for the table. Delete all entries
        print("deleting", str(x[0]), "...")
        query = session.query(x[0])
        query.delete(synchronize_session= False)
        session.commit()

    session.close()


@dag(
    dag_id="cleanup_db",
    schedule_interval="@weekly",
    start_date=days_ago(7),
    catchup=False,
    is_paused_upon_creation=False
)

def clean_db_dag_fn():
    t_last=None
    for x in TABLES_TO_CLEAN:
        t=cleanup_db_fn(x)
        if t_last:
            t_last >> t
        t_last = t

clean_db_dag = clean_db_dag_fn()
```

Exporting environment metadata to CSV files on Amazon S3

The following code example shows how you can create a directed acyclic graph (DAG) that queries the database for a range of DAG run information, and writes the data to `.csv` files stored on Amazon S3.

You might want to export information from the your environment's Aurora PostgreSQL database in order to inspect the data locally, archive them in object storage, or combine them with tools like

the Amazon S3 to Amazon Redshift operator and the database cleanup, in order to move Amazon MWAA metadata out of the environment, but preserve them for future analysis.

You can query the database for any of the objects listed in Apache Airflow models. This code sample uses three models, `DagRun`, `TaskFail`, and `TaskInstance`, which provide information relevant to DAG runs.

Topics

- Version
- Prerequisites
- Permissions
- Requirements
- Code sample

Version

- You can use the code example on this page with **Apache Airflow v2** in Python 3.10.

Prerequisites

To use the sample code on this page, you'll need the following:

- An Amazon MWAA environment.
- A new Amazon S3 bucket where you want to export your metadata information.

Permissions

Amazon MWAA needs permission for the Amazon S3 action `s3:PutObject` to write the queried metadata information to Amazon S3. Add the following policy statement to your environment's execution role.

```
{
  "Effect": "Allow",
  "Action": "s3:PutObject*",
  "Resource": "arn:aws:s3:::your-new-export-bucket"
}
```

This policy limits write access to only *your-new-export-bucket*.

Requirements

- To use this code example with Apache Airflow v2, no additional dependencies are required. The code uses the Apache Airflow v2 base install on your environment.

Code sample

The following steps describe how you can create a DAG that queries the Aurora PostgreSQL and writes the result to your new Amazon S3 bucket.

1. In your terminal, navigate to the directory where your DAG code is stored. For example:

```
cd dags
```

2. Copy the contents of the following code example and save it locally as `metadata_to_csv.py`. You can change the value assigned to `MAX_AGE_IN_DAYS` to control the age of the oldest records your DAG queries from the metadata database.

```
from airflow.decorators import dag, task
from airflow import settings
import os
import boto3
from airflow.utils.dates import days_ago
from airflow.models import DagRun, TaskFail, TaskInstance
import csv, re
from io import StringIO

DAG_ID = os.path.basename(__file__).replace(".py", "")

MAX_AGE_IN_DAYS = 30
S3_BUCKET = '<your-export-bucket>'
S3_KEY = 'files/export/{0}.csv'

# You can add other objects to export from the metadatabase,
OBJECTS_TO_EXPORT = [
    [DagRun,DagRun.execution_date],
    [TaskFail,TaskFail.execution_date],
    [TaskInstance, TaskInstance.execution_date],
]
```

```
@task()
def export_db_task(**kwargs):
    session = settings.Session()
    print("session: ",str(session))

    oldest_date = days_ago(MAX_AGE_IN_DAYS)
    print("oldest_date: ",oldest_date)

    s3 = boto3.client('s3')

    for x in OBJECTS_TO_EXPORT:
        query = session.query(x[0]).filter(x[1] >= days_ago(MAX_AGE_IN_DAYS))
        print("type",type(query))
        allrows=query.all()
        name=re.sub("[<>']", "", str(x[0]))
        print(name,": ",str(allrows))

        if len(allrows) > 0:
            outfileStr=""
            f = StringIO(outfileStr)
            w = csv.DictWriter(f, vars(allrows[0]).keys())
            w.writeheader()
            for y in allrows:
                w.writerow(vars(y))
            outkey = S3_KEY.format(name[6:])
            s3.put_object(Bucket=S3_BUCKET, Key=outkey, Body=f.getvalue())

@dag(
    dag_id=DAG_ID,
    schedule_interval=None,
    start_date=days_ago(1),
    )
def export_db():
    t = export_db_task()

metadb_to_s3_test = export_db()
```

3. Run the following AWS CLI command to copy the DAG to your environment's bucket, then trigger the DAG using the Apache Airflow UI.

```
$ aws s3 cp your-dag.py s3://your-environment-bucket/dags/
```

4. If successful, you'll output similar to the following in the task logs for the `export_db` task:

```
[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
```

You can now access and download the exported `.csv` files in your new Amazon S3 bucket in `/files/export/`.

Using a secret key in AWS Secrets Manager for an Apache Airflow variable

The following sample calls AWS Secrets Manager to get a secret key for an Apache Airflow variable on Amazon Managed Workflows for Apache Airflow. It assumes you've completed the steps in Configuring an Apache Airflow connection using a AWS Secrets Manager secret.

Topics

- Version
- Prerequisites
- Permissions
- Requirements

- Code sample
- What's next?

Version

- The sample code on this page can be used with **Apache Airflow v1** in Python 3.7.
- You can use the code example on this page with **Apache Airflow v2** in Python 3.10.

Prerequisites

To use the sample code on this page, you'll need the following:

- The Secrets Manager backend as an Apache Airflow configuration option as shown in Configuring an Apache Airflow connection using a AWS Secrets Manager secret.
- An Apache Airflow variable string in Secrets Manager as shown in Configuring an Apache Airflow connection using a AWS Secrets Manager secret.

Permissions

- Secrets Manager permissions as shown in Configuring an Apache Airflow connection using a AWS Secrets Manager secret.

Requirements

- To use this code example with Apache Airflow v1, no additional dependencies are required. The code uses the Apache Airflow v1 base install on your environment.
- To use this code example with Apache Airflow v2, no additional dependencies are required. The code uses the Apache Airflow v2 base install on your environment.

Code sample

The following steps describe how to create the DAG code that calls Secrets Manager to get the secret.

1. In your command prompt, navigate to the directory where your DAG code is stored. For example:

```
cd dags
```

2. Copy the contents of the following code sample and save locally as `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
    )
```

What's next?

- Learn how to upload the DAG code in this example to the `dags` folder in your Amazon S3 bucket in Adding or updating DAGs.

Using a secret key in AWS Secrets Manager for an Apache Airflow connection

The following sample calls AWS Secrets Manager to get a secret key for an Apache Airflow connection on Amazon Managed Workflows for Apache Airflow. It assumes you've completed the steps in Configuring an Apache Airflow connection using a AWS Secrets Manager secret.

Topics

- Version
- Prerequisites
- Permissions
- Requirements
- Code sample
- What's next?

Version

- The sample code on this page can be used with **Apache Airflow v1** in Python 3.7.

- You can use the code example on this page with **Apache Airflow v2** in Python 3.10.

Prerequisites

To use the sample code on this page, you'll need the following:

- The Secrets Manager backend as an Apache Airflow configuration option as shown in Configuring an Apache Airflow connection using a AWS Secrets Manager secret.
- An Apache Airflow connection string in Secrets Manager as shown in Configuring an Apache Airflow connection using a AWS Secrets Manager secret.

Permissions

- Secrets Manager permissions as shown in Configuring an Apache Airflow connection using a AWS Secrets Manager secret.

Requirements

- To use this code example with Apache Airflow v1, no additional dependencies are required. The code uses the Apache Airflow v1 base install on your environment.

- To use this code example with Apache Airflow v2, no additional dependencies are required. The code uses the Apache Airflow v2 base install on your environment.

Code sample

The following steps describe how to create the DAG code that calls Secrets Manager to get the secret.

Apache Airflow v2

1. In your command prompt, navigate to the directory where your DAG code is stored. For example:

```
cd dags
```

2. Copy the contents of the following code sample and save locally as `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
    )
```

Apache Airflow v1

1. In your command prompt, navigate to the directory where your DAG code is stored. For example:

```
cd dags
```

2. Copy the contents of the following code sample and save locally as `secrets-manager.py`.

```
from airflow import DAG, settings, secrets
from airflow.operators.python_operator import PythonOperator
from airflow.utils.dates import days_ago
from airflow.contrib.hooks.aws_hook import AwsHook

from datetime import timedelta
import os

### The steps to create this secret key can be found at: https://
docs.aws.amazon.com/mwaa/latest/userguide/connections-secrets-manager.html
sm_secretId_name = 'airflow/connections/myconn'

default_args = {
    'owner': 'airflow',
    'start_date': days_ago(1),
    'depends_on_past': False
}


### Gets the secret myconn from Secrets Manager
```

```
def read_from_aws_sm_fn(**kwargs):
    ### set up Secrets Manager
    hook = AwsHook()
    client = hook.get_client_type('secretsmanager')
    response = client.get_secret_value(SecretId=sm_secretId_name)
    myConnSecretString = response["SecretString"]

    return myConnSecretString

### 'os.path.basename(__file__).replace(".py", "")' uses the file name secrets-
manager.py for a DAG ID of secrets-manager
with DAG(
        dag_id=os.path.basename(__file__).replace(".py", ""),
        default_args=default_args,
        dagrun_timeout=timedelta(hours=2),
        start_date=days_ago(1),
        schedule_interval=None
) as dag:
    write_all_to_aws_sm = PythonOperator(
        task_id="read_from_aws_sm",
        python_callable=read_from_aws_sm_fn,
        provide_context=True
    )
```

What's next?

- Learn how to upload the DAG code in this example to the dags folder in your Amazon S3 bucket in Adding or updating DAGs.

Creating a custom plugin with Oracle

The following sample walks you through the steps to create a custom plugin using Oracle for Amazon MWAA and can be combined with other custom plugins and binaries in your plugins.zip file.

Contents

- Version
- Prerequisites
- Permissions

- Requirements
- Code sample
- Create the custom plugin
  - Download dependencies
  - Custom plugin
  - Plugins.zip
- Airflow configuration options
- What's next?

Version

- The sample code on this page can be used with **Apache Airflow v1** in Python 3.7.

- You can use the code example on this page with **Apache Airflow v2** in Python 3.10.

Prerequisites

To use the sample code on this page, you'll need the following:

- An Amazon MWAA environment.
- *Worker* logging enabled at any log level, `CRITICAL` or above, for your environment. For more information about Amazon MWAA log types and how to manage your log groups, see the section called “Viewing Airflow logs”

Permissions

- No additional permissions are required to use the code example on this page.

Requirements

To use the sample code on this page, add the following dependencies to your `requirements.txt`. To learn more, see Installing Python dependencies.

Apache Airflow v2

```
-c https://raw.githubusercontent.com/apache/airflow/constraints-2.0.2/
constraints-3.7.txt
cx_Oracle
apache-airflow-providers-oracle
```

Apache Airflow v1

```
cx_Oracle==8.1.0
apache-airflow[oracle]==1.10.12
```

Code sample

The following steps describe how to create the DAG code that will test the custom plugin.

1. In your command prompt, navigate to the directory where your DAG code is stored. For example:

```
cd dags
```

2. Copy the contents of the following code sample and save locally as `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
    )
```

Create the custom plugin

This section describes how to download the dependencies, create the custom plugin and the plugins.zip.

Download dependencies

Amazon MWAA will extract the contents of plugins.zip into `/usr/local/airflow/plugins` on each Amazon MWAA scheduler and worker container. This is used to add binaries to your environment. The following steps describe how to assemble the files needed for the custom plugin.

Pull the Amazon Linux container image

1. In your command prompt, pull the Amazon Linux container image, and run the container locally. For example:

   ```
   docker pull amazonlinux
   docker run -it amazonlinux:latest /bin/bash
   ```

   Your command prompt should invoke a bash command line. For example:

   ```
   bash-4.2#
   ```

2. Install the Linux-native asynchronous I/O facility (libaio).

   ```
   yum -y install libaio
   ```

3. Keep this window open for subsequent steps. We'll be copying the following files locally: `lib64/libaio.so.1`, `lib64/libaio.so.1.0.0`, `lib64/libaio.so.1.0.1`.

Download client folder

1. Install the unzip package locally. For example:

   ```
   sudo yum install unzip
   ```

2. Create an `oracle_plugin` directory. For example:

```
mkdir oracle_plugin
cd oracle_plugin
```

3. Use the following curl command to download the instantclient-basic-linux.x64-18.5.0.0.0dbru.zip from Oracle Instant Client Downloads for Linux x86-64 (64-bit).

```
curl https://download.oracle.com/otn_software/linux/instantclient/185000/
instantclient-basic-linux.x64-18.5.0.0.0dbru.zip > client.zip
```

4. Unzip the `client.zip` file. For example:

```
unzip *.zip
```

Extract files from Docker

1. In a new command prompt, display and write down your Docker container ID. For example:

```
docker container ls
```

Your command prompt should return all containers and their IDs. For example:

```
debc16fd6970
```

2. In your `oracle_plugin` directory, extract the `lib64/libaio.so.1`, `lib64/libaio.so.1.0.0`, `lib64/libaio.so.1.0.1` files to the local `instantclient_18_5` folder. For example:

```
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/
```

Custom plugin

Apache Airflow will execute the contents of Python files in the plugins folder at startup. This is used to set and modify environment variables. The following steps describe the sample code for the custom plugin.

- Copy the contents of the following code sample and save locally as `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

The following steps show how to create the `plugins.zip`. The contents of this example can be combined with your other plugins and binaries into a single `plugins.zip` file.

Zip the contents of the plugin directory

1. In your command prompt, navigate to the `oracle_plugin` directory. For example:

```
cd oracle_plugin
```

2. Zip the `instantclient_18_5` directory in plugins.zip. For example:

```
zip -r ../plugins.zip ./
```

3. You should see the following in your command prompt:

```
oracle_plugin$ ls
client.zip  instantclient_18_5
```

4. Remove the `client.zip` file. For example:

```
rm client.zip
```

Zip the env_var_plugin_oracle.py file

1. Add the `env_var_plugin_oracle.py` file to the root of the plugins.zip. For example:

```
zip plugins.zip env_var_plugin_oracle.py
```

2. Your plugins.zip should now include the following:

```
env_var_plugin_oracle.py
instantclient_18_5/
```

Airflow configuration options

If you're using Apache Airflow v2, add `core.lazy_load_plugins : False` as an Apache Airflow configuration option. To learn more, see Using configuration options to load plugins in 2.

What's next?

- Learn how to upload the `requirements.txt` file in this example to your Amazon S3 bucket in Installing Python dependencies.
- Learn how to upload the DAG code in this example to the `dags` folder in your Amazon S3 bucket in Adding or updating DAGs.
- Learn more about how to upload the `plugins.zip` file in this example to your Amazon S3 bucket in Installing custom plugins.

Creating a custom plugin that generates runtime environment variables

The following sample walks you through the steps to create a custom plugin that generates environment variables at runtime on an Amazon Managed Workflows for Apache Airflow environment.

Topics

- Version
- Prerequisites
- Permissions
- Requirements
- Custom plugin

- Plugins.zip
- Airflow configuration options
- What's next?

Version

- The sample code on this page can be used with **Apache Airflow v1** in Python 3.7.

Prerequisites

To use the sample code on this page, you'll need the following:

- An Amazon MWAA environment.

Permissions

- No additional permissions are required to use the code example on this page.

Requirements

- To use this code example with Apache Airflow v1, no additional dependencies are required. The code uses the Apache Airflow v1 base install on your environment.

Custom plugin

Apache Airflow will execute the contents of Python files in the plugins folder at startup. This is used to set and modify environment variables. The following steps describe the sample code for the custom plugin.

1. In your command prompt, navigate to the directory where your plugins are stored. For example:

```
cd plugins
```

2. Copy the contents of the following code sample and save locally as `env_var_plugin.py` in the above folder.

```
from airflow.plugins_manager import AirflowPlugin
import os

os.environ["PATH"] = os.getenv("PATH") + ":/usr/local/airflow/.local/lib/python3.7/
site-packages"
os.environ["JAVA_HOME"]="/usr/lib/jvm/java-1.8.0-
openjdk-1.8.0.272.b10-1.amzn2.0.1.x86_64"

class EnvVarPlugin(AirflowPlugin):
     name = 'env_var_plugin'
```

Plugins.zip

The following steps show how to create `plugins.zip`. The contents of this example can be combined with other plugins and binaries into a single `plugins.zip` file.

1. In your command prompt, navigate to the `hive_plugin` directory from the previous step. For example:

```
cd plugins
```

2. Zip the contents within your `plugins` folder.

```
zip -r ../plugins.zip ./
```

Airflow configuration options

If you're using Apache Airflow v2, add `core.lazy_load_plugins : False` as an Apache Airflow configuration option. To learn more, see Using configuration options to load plugins in 2.

What's next?

- Learn how to upload the `requirements.txt` file in this example to your Amazon S3 bucket in Installing Python dependencies.
- Learn how to upload the DAG code in this example to the `dags` folder in your Amazon S3 bucket in Adding or updating DAGs.

- Learn more about how to upload the `plugins.zip` file in this example to your Amazon S3 bucket in Installing custom plugins.

Changing a DAG's timezone on Amazon MWAA

Apache Airflow schedules your directed acyclic graph (DAG) in UTC+0 by default. The following steps show how you can change the timezone in which Amazon MWAA runs your DAGs with Pendulum. Optionally, this topic demonstrates how you can create a custom plugin to change the timezone for your environment's Apache Airflow logs.

Topics

- Version
- Prerequisites
- Permissions
- Create a plugin to change the timezone in Airflow logs
- Create a plugins.zip
- Code sample
- What's next?

Version

- You can use the code example on this page with **Apache Airflow v2** in Python 3.10.

Prerequisites

To use the sample code on this page, you'll need the following:

- An Amazon MWAA environment.

Permissions

- No additional permissions are required to use the code example on this page.

Create a plugin to change the timezone in Airflow logs

Apache Airflow will run the Python files in the `plugins` directory at start-up. With the following plugin, you can override the executor's timezone, which modifies the timezone in which Apache Airflow writes logs.

1. Create a directory named `plugins` for your custom plugin, and navigate to the directory. For example:

```
$ mkdir plugins
$ cd plugins
```

2. Copy the contents of the following code sample and save locally as `dag-timezone-plugin.py` in the `plugins` folder.

```
import time
import os

os.environ['TZ'] = 'America/Los_Angeles'
time.tzset()
```

3. In the `plugins` directory, create an empty Python file named `__init__.py`. Your `plugins` directory should be similar to the following:

```
plugins/
|-- __init__.py
|-- dag-timezone-plugin.py
```

Create a `plugins.zip`

The following steps show how to create `plugins.zip`. The content of this example can be combined with other plugins and binaries into a single `plugins.zip` file.

1. In your command prompt, navigate to the `plugins` directory from the previous step. For example:

```
cd plugins
```

2. Zip the contents within your `plugins` directory.

```
zip -r ../plugins.zip ./
```

3. Upload `plugins.zip` to your S3 bucket

```
$ aws s3 cp plugins.zip s3://your-mwaa-bucket/
```

Code sample

To change the default timezone (UTC+0) in which the DAG runs, we'll use a library called Pendulum, a Python library for working with timezone-aware datetime.

1. In your command prompt, navigate to the directory where your DAGs are stored. For example:

```
$ cd dags
```

2. Copy the content of the following example and save as `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. Run the following AWS CLI command to copy the DAG to your environment's bucket, then trigger the DAG using the Apache Airflow UI.

```
$ aws s3 cp your-dag.py s3://your-environment-bucket/dags/
```

4. If successful, you'll output similar to the following in the task logs for the `tz_aware_task` in the `tz_test` DAG:

```
[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
```

What's next?

- Learn more about how to upload the `plugins.zip` file in this example to your Amazon S3 bucket in Installing custom plugins.

Refreshing a CodeArtifact token

If you're using CodeArtifact to install Python dependencies, Amazon MWAA requires an active token. To allow Amazon MWAA to access a CodeArtifact repository at runtime, you can use a startup script and set the `PIP_EXTRA_INDEX_URL` with the token.

The following topic describes how you can create a startup script that uses the `get_authorization_token` CodeArtifact API operation to retrieve a fresh token every time your environment starts up, or updates.

Topics

- Version

- Prerequisites
- Permissions
- Code sample
- What's next?

Version

- You can use the code example on this page with **Apache Airflow v2** in Python 3.10.

Prerequisites

To use the sample code on this page, you'll need the following:

- An Amazon MWAA environment.
- A CodeArtifact repository where you store dependencies for your environment.

Permissions

To refresh the CodeArtifact token and write the result to Amazon S3 Amazon MWAA must have the following permissions in the execution role.

- The `codeartifact:GetAuthorizationToken` action allows Amazon MWAA to retrieve a new token from CodeArtifact. The following policy grants permission for every CodeArtifact domain you create. You can further restrict access to your domains by modifying the resource value in the statement, and specifying only the domains that you want your environment to access.

```
{
  "Effect": "Allow",
  "Action": "codeartifact:GetAuthorizationToken",
  "Resource": "arn:aws:codeartifact:us-west-2:*:domain/*"
}
```

- The `sts:GetServiceBearerToken` action is required to call the CodeArtifact `GetAuthorizationToken` API operation. This operation returns a token that must be used when using a package manager such as `pip` with CodeArtifact. To use a package manager with a CodeArtifact repository, your environment's execution role role must allow `sts:GetServiceBearerToken` as shown in the following policy statement.

```
{
  "Sid": "AllowServiceBearerToken",
  "Effect": "Allow",
  "Action": "sts:GetServiceBearerToken",
  "Resource": "*"
}
```

Code sample

The following steps describe how you can create a start up script that updates the CodeArtifact token.

1. Copy the contents of the following code sample and save locally as `code_artifact_startup_script.sh`.

```
#!/bin/sh

# Startup script for MWAA, see https://docs.aws.amazon.com/mwaa/latest/userguide/
using-startup-script.html

set -eu

# setup code artifact endpoint and token
# https://pip.pypa.io/en/stable/cli/pip_install/#cmdoption-0
# https://docs.aws.amazon.com/mwaa/latest/userguide/samples-code-artifact.html
DOMAIN="amazon"
DOMAIN_OWNER="112233445566"
REGION="us-west-2"
REPO_NAME="MyRepo"
echo "Getting token for CodeArtifact with args: --domain $DOMAIN --region $REGION
 --domain-owner $DOMAIN_OWNER"
TOKEN=$(aws codeartifact get-authorization-token --domain $DOMAIN --region $REGION
 --domain-owner $DOMAIN_OWNER | jq -r '.authorizationToken')
echo "Setting Pip env var for '--index-url' to point to CodeArtifact"
export PIP_EXTRA_INDEX_URL="https://aws:$TOKEN@$DOMAIN-
$DOMAIN_OWNER.d.codeartifact.$REGION.amazonaws.com/pypi/$REPO_NAME/simple/"
echo "CodeArtifact startup setup complete"
```

2. Navigate to the folder where you saved the script. Use `cp` in a new prompt window to upload the script to your bucket. Replace *`your-s3-bucket`* with your information.

```
$ aws s3 cp code_artifact_startup_script.sh s3://your-s3-bucket/
code_artifact_startup_script.sh
```

If successful, Amazon S3 outputs the URL path to the object:

```
upload: ./code_artifact_startup_script.sh to s3://your-s3-bucket/
code_artifact_startup_script.sh
```

After you upload the script, your environment updates and runs the script at startup.

What's next?

- Learn how to use startup scripts to customize your environment in the section called "Using a startup script".
- Learn how to upload the DAG code in this example to the `dags` folder in your Amazon S3 bucket in Adding or updating DAGs.
- Learn more about how to upload the `plugins.zip` file in this example to your Amazon S3 bucket in Installing custom plugins.

Creating a custom plugin with Apache Hive and Hadoop

Amazon MWAA extracts the contents of a `plugins.zip` to `/usr/local/airflow/plugins`. This can be used to add binaries to your containers. In addition, Apache Airflow executes the contents of Python files in the `plugins` folder at *startup*—enabling you to set and modify environment variables. The following sample walks you through the steps to create a custom plugin using Apache Hive and Hadoop on an Amazon Managed Workflows for Apache Airflow environment and can be combined with other custom plugins and binaries.

Topics

- Version
- Prerequisites
- Permissions
- Requirements
- Download dependencies

- Custom plugin
- Plugins.zip
- Code sample
- Airflow configuration options
- What's next?

Version

- The sample code on this page can be used with **Apache Airflow v1** in Python 3.7.
- You can use the code example on this page with **Apache Airflow v2** in Python 3.10.

Prerequisites

To use the sample code on this page, you'll need the following:

- An Amazon MWAA environment.

Permissions

- No additional permissions are required to use the code example on this page.

Requirements

To use the sample code on this page, add the following dependencies to your `requirements.txt`. To learn more, see Installing Python dependencies.

Apache Airflow v2

```
-c https://raw.githubusercontent.com/apache/airflow/constraints-2.0.2/
constraints-3.7.txt
apache-airflow-providers-amazon[apache.hive]
```

Apache Airflow v1

```
apache-airflow[hive]==1.10.12
```

Download dependencies

Amazon MWAA will extract the contents of plugins.zip into `/usr/local/airflow/plugins` on each Amazon MWAA scheduler and worker container. This is used to add binaries to your environment. The following steps describe how to assemble the files needed for the custom plugin.

1. In your command prompt, navigate to the directory where you would like to create your plugin. For example:

   ```
   cd plugins
   ```

2. Download Hadoop from a mirror, for example:

   ```
   wget https://downloads.apache.org/hadoop/common/hadoop-3.3.0/hadoop-3.3.0.tar.gz
   ```

3. Download Hive from a mirror, for example:

   ```
   wget https://downloads.apache.org/hive/hive-3.1.2/apache-hive-3.1.2-bin.tar.gz
   ```

4. Create a directory. For example:

   ```
   mkdir hive_plugin
   ```

5. Extract Hadoop.

   ```
   tar -xvzf hadoop-3.3.0.tar.gz -C hive_plugin
   ```

6. Extract Hive.

   ```
   tar -xvzf apache-hive-3.1.2-bin.tar.gz -C hive_plugin
   ```

Custom plugin

Apache Airflow will execute the contents of Python files in the plugins folder at startup. This is used to set and modify environment variables. The following steps describe the sample code for the custom plugin.

1. In your command prompt, navigate to the `hive_plugin` directory. For example:

```
cd hive_plugin
```

2. Copy the contents of the following code sample and save locally as `hive_plugin.py` in the `hive_plugin` directory.

```
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. Cope the content of the following text and save locally as `.airflowignore` in the `hive_plugin` directory.

```
hadoop-3.3.0
apache-hive-3.1.2-bin
```

Plugins.zip

The following steps show how to create `plugins.zip`. The contents of this example can be combined with other plugins and binaries into a single `plugins.zip` file.

1. In your command prompt, navigate to the `hive_plugin` directory from the previous step. For example:

```
cd hive_plugin
```

2. Zip the contents within your `plugins` folder.

```
zip -r ../hive_plugin.zip ./
```

Code sample

The following steps describe how to create the DAG code that will test the custom plugin.

1. In your command prompt, navigate to the directory where your DAG code is stored. For example:

```
cd dags
```

2. Copy the contents of the following code sample and save locally as `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 configuration options

If you're using Apache Airflow v2, add `core.lazy_load_plugins : False` as an Apache Airflow configuration option. To learn more, see Using configuration options to load plugins in 2.

What's next?

- Learn how to upload the `requirements.txt` file in this example to your Amazon S3 bucket in Installing Python dependencies.

- Learn how to upload the DAG code in this example to the `dags` folder in your Amazon S3 bucket in Adding or updating DAGs.
- Learn more about how to upload the `plugins.zip` file in this example to your Amazon S3 bucket in Installing custom plugins.

Creating a custom plugin for Apache Airflow PythonVirtualenvOperator

The following sample shows how to patch the Apache Airflow PythonVirtualenvOperator with a custom plugin on Amazon Managed Workflows for Apache Airflow.

Topics

- Version
- Prerequisites
- Permissions
- Requirements
- Custom plugin sample code
- Plugins.zip
- Code sample
- Airflow configuration options
- What's next?

Version

- The sample code on this page can be used with **Apache Airflow v1** in Python 3.7.
- You can use the code example on this page with **Apache Airflow v2** in Python 3.10.

Prerequisites

To use the sample code on this page, you'll need the following:

- An Amazon MWAA environment.

Permissions

- No additional permissions are required to use the code example on this page.

Requirements

To use the sample code on this page, add the following dependencies to your `requirements.txt`. To learn more, see Installing Python dependencies.

```
virtualenv
```

Custom plugin sample code

Apache Airflow will execute the contents of Python files in the plugins folder at startup. This plugin will patch the built-in `PythonVirtualenvOperator` during that startup process to make it compatible with Amazon MWAA. The following steps show the sample code for the custom plugin.

Apache Airflow v2

1. In your command prompt, navigate to the `plugins` directory above. For example:

   ```
   cd plugins
   ```

2. Copy the contents of the following code sample and save locally as `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'
```

Apache Airflow v1

1. In your command prompt, navigate to the `plugins` directory above. For example:

```
cd plugins
```

2. Copy the contents of the following code sample and save locally as `virtual_python_plugin.py`.

```
from airflow.plugins_manager import AirflowPlugin
from airflow.operators.python_operator import PythonVirtualenvOperator

def _generate_virtualenv_cmd(self, tmp_dir):
    cmd = ['python3','/usr/local/airflow/.local/lib/python3.7/site-packages/
virtualenv', tmp_dir]
    if self.system_site_packages:
        cmd.append('--system-site-packages')
    if self.python_version is not None:
        cmd.append('--python=python{}'.format(self.python_version))
    return cmd
PythonVirtualenvOperator._generate_virtualenv_cmd=_generate_virtualenv_cmd
```

```
class EnvVarPlugin(AirflowPlugin):
    name = 'virtual_python_plugin'
```

Plugins.zip

The following steps show how to create the `plugins.zip`.

1. In your command prompt, navigate to the directory containing `virtual_python_plugin.py` above. For example:

```
cd plugins
```

2. Zip the contents within your `plugins` folder.

```
zip plugins.zip virtual_python_plugin.py
```

Code sample

The following steps describe how to create the DAG code for the custom plugin.

Apache Airflow v2

1. In your command prompt, navigate to the directory where your DAG code is stored. For example:

```
cd dags
```

2. Copy the contents of the following code sample and save locally as `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,
    )
```

Apache Airflow v1

1. In your command prompt, navigate to the directory where your DAG code is stored. For example:

```
cd dags
```

2. Copy the contents of the following code sample and save locally as `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_operator import PythonVirtualenvOperator
from airflow.utils.dates import days_ago
import os

os.environ["PATH"] = os.getenv("PATH") + ":/usr/local/airflow/.local/bin"

def virtualenv_fn():
    import boto3
    print("boto3 version ",boto3.__version__)

with DAG(dag_id="virtualenv_test", schedule_interval=None, catchup=False,
 start_date=days_ago(1)) as dag:
    virtualenv_task = PythonVirtualenvOperator(
        task_id="virtualenv_task",
        python_callable=virtualenv_fn,
        requirements=["boto3>=1.17.43"],
        system_site_packages=False,
        dag=dag,
    )
```

Airflow configuration options

If you're using Apache Airflow v2, add `core.lazy_load_plugins : False` as an Apache Airflow configuration option. To learn more, see Using configuration options to load plugins in 2.

What's next?

- Learn how to upload the `requirements.txt` file in this example to your Amazon S3 bucket in Installing Python dependencies.
- Learn how to upload the DAG code in this example to the `dags` folder in your Amazon S3 bucket in Adding or updating DAGs.
- Learn more about how to upload the `plugins.zip` file in this example to your Amazon S3 bucket in Installing custom plugins.

Invoking DAGs with a Lambda function

The following code example uses an AWS Lambda function to get an Apache Airflow CLI token and invoke a directed acyclic graph (DAG) in an Amazon MWAA environment.

Topics

- Version
- Prerequisites
- Permissions
- Dependencies
- Code example

Version

- You can use the code example on this page with **Apache Airflow v2** in Python 3.10.

Prerequisites

To use this code example, you must:

- Use the public network access mode for your Amazon MWAA environment.
- Have a Lambda function using the latest Python runtime.

Note

If the Lambda function and your Amazon MWAA environment are in the same VPC, you can use this code on a private network. For this configuration, the Lambda function's execution role needs permission to call the Amazon Elastic Compute Cloud (Amazon EC2) **CreateNetworkInterface** API operation. You can provide this permission using the `AWSLambdaVPCAccessExecutionRole` AWS managed policy.

Permissions

To use the code example on this page, your Amazon MWAA environment's execution role needs access to perform the `airflow:CreateCliToken` action. You can provide this permission using the `AmazonMWAAAirflowCliAccess` AWS managed policy:

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

For more information, see Apache Airflow CLI policy: AmazonMWAAAirflowCliAccess.

Dependencies

- To use this code example with Apache Airflow v2, no additional dependencies are required. The code uses the Apache Airflow v2 base install on your environment.

Code example

1. Open the AWS Lambda console at https://console.aws.amazon.com/lambda/.
2. Choose your Lambda function from the **Functions** list.

3. On the function page, copy the following code and replace the following with the names of your resources:

   - YOUR_ENVIRONMENT_NAME – The name of your Amazon MWAA environment.
   - YOUR_DAG_NAME – The name of the DAG that you want to invoke.

```
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. Choose **Deploy**.
5. Choose **Test** to invoke your function using the Lambda console.
6. To verify that your Lambda successfully invoked your DAG, use the Amazon MWAA console to navigate to your environment's Apache Airflow UI, then do the following:

   a. On the **DAGs** page, locate your new target DAG in the list of DAGs.

b. Under **Last Run**, check the timestamp for the latest DAG run. This timestamp should closely match the latest timestamp for `invoke_dag` in your other environment.

c. Under **Recent Tasks**, check that the last run was successful.

Invoking DAGs in different Amazon MWAA environments

The following code example creates an Apache Airflow CLI token. The code then uses a directed acyclic graph (DAG) in one Amazon MWAA environment to invoke a DAG in a different Amazon MWAA environment.

Topics

- Version
- Prerequisites
- Permissions
- Dependencies
- Code example

Version

- You can use the code example on this page with **Apache Airflow v2** in Python 3.10.

Prerequisites

To use the code example on this page, you need the following:

- Two Amazon MWAA environments with **public network** web server access, including your current environment.
- A sample DAG uploaded to your target environment's Amazon Simple Storage Service (Amazon S3) bucket.

Permissions

To use the code example on this page, your environment's execution role must have permission to create an Apache Airflow CLI token. You can attach the AWS managed policy `AmazonMWAAAirflowCliAccess` to grant this permission.

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

For more information, see Apache Airflow CLI policy: AmazonMWAAAirflowCliAccess.

Dependencies

- To use this code example with Apache Airflow v2, no additional dependencies are required. The code uses the Apache Airflow v2 base install on your environment.

Code example

The following code example assumes that you're using a DAG in your current environment to invoke a DAG in another environment.

1. In your terminal, navigate to the directory where your DAG code is stored. For example:

   ```
   cd dags
   ```

2. Copy the content of the following code example and save it locally as `invoke_dag.py`. Replace the following values with your information.

   - `your-new-environment-name` – The name of the other environment where you want to invoke the DAG.

- `your-target-dag-id` – The ID of the DAG in the other environment that you want to invoke.

```
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. Run the following AWS CLI command to copy the DAG to your environment's bucket, then trigger the DAG using the Apache Airflow UI.

```
$ aws s3 cp your-dag.py s3://your-environment-bucket/dags/
```

4. If the DAG runs successfully, you'll see output similar to the following in the task logs for `invoke_dag_task`.

```
[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
```

To verify that your DAG was successfully invoked, navigate to the Apache Airflow UI for your new environment, then do the following:

a. On the **DAGs** page, locate your new target DAG in the list of DAGs.

b. Under **Last Run**, check the timestamp for the latest DAG run. This timestamp should closely match the latest timestamp for `invoke_dag` in your other environment.

c. Under **Recent Tasks**, check that the last run was successful.

Using Amazon MWAA with Amazon RDS for Microsoft SQL Server

You can use Amazon Managed Workflows for Apache Airflow to connect to an RDS for SQL Server. The following sample code uses DAGs on an Amazon Managed Workflows for Apache Airflow environment to connect to and execute queries on an Amazon RDS for Microsoft SQL Server.

Topics

- Version
- Prerequisites
- Dependencies
- Apache Airflow v2 connection
- Code sample
- What's next?

Version

- The sample code on this page can be used with **Apache Airflow v1** in Python 3.7.

- You can use the code example on this page with **Apache Airflow v2** in Python 3.10.

Prerequisites

To use the sample code on this page, you'll need the following:

- An Amazon MWAA environment.
- Amazon MWAA and the RDS for SQL Server are running in the same Amazon VPC/
- VPC security groups of Amazon MWAA and the server are configured with the following connections:
  - An inbound rule for the port 1433 open for Amazon RDS in Amazon MWAA's security group
  - Or an outbound rule for the port of 1433 open from Amazon MWAA to RDS
- Apache Airflow Connection for RDS for SQL Server reflects the hostname, port, username and password from the Amazon RDS SQL server database created in previous process.

Dependencies

To use the sample code in this section, add the following dependency to your `requirements.txt`. To learn more, see Installing Python dependencies

Apache Airflow v2

```
apache-airflow-providers-microsoft-mssql==1.0.1
apache-airflow-providers-odbc==1.0.1
pymssql==2.2.1
```

Apache Airflow v1

```
apache-airflow[mssql]==1.10.12
```

Apache Airflow v2 connection

If you're using a connection in Apache Airflow v2, ensure the Airflow connection object includes the following key-value pairs:

1. **Conn Id:** mssql_default

2. **Conn Type:** Amazon Web Services
3. **Host:** YOUR_DB_HOST
4. **Schema:**
5. **Login:** admin
6. **Password:**
7. **Port:** 1433
8. **Extra:**

Code sample

1. In your command prompt, navigate to the directory where your DAG code is stored. For example:

```
cd dags
```

2. Copy the contents of the following code sample and save locally as `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
```

What's next?

- Learn how to upload the `requirements.txt` file in this example to your Amazon S3 bucket in Installing Python dependencies.
- Learn how to upload the DAG code in this example to the `dags` folder in your Amazon S3 bucket in Adding or updating DAGs.
- Explore example scripts and other pymssql module examples.
- Learn more about executing SQL code in a specific Microsoft SQL database using the mssql_operator in the *Apache Airflow reference guide*.

Using Amazon MWAA with Amazon EMR

The following code sample demonstrates how to enable an integration using Amazon EMR and Amazon Managed Workflows for Apache Airflow.

Topics

- Version
- Code sample

Version

- The sample code on this page can be used with **Apache Airflow v1** in Python 3.7.

Code sample

```
"""
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.amazon.aws.operators.emr import EmrAddStepsOperator
from airflow.providers.amazon.aws.sensors.emr import EmrStepSensor
from airflow.providers.amazon.aws.operators.emr import EmrCreateJobFlowOperator

from airflow.utils.dates import days_ago
from datetime import timedelta
import os
```

```
DAG_ID = os.path.basename(__file__).replace(".py", "")

DEFAULT_ARGS = {
    'owner': 'airflow',
    'depends_on_past': False,
    'email': ['airflow@example.com'],
    'email_on_failure': False,
    'email_on_retry': False,
}

SPARK_STEPS = [
    {
        'Name': 'calculate_pi',
        'ActionOnFailure': 'CONTINUE',
        'HadoopJarStep': {
            'Jar': 'command-runner.jar',
            'Args': ['/usr/lib/spark/bin/run-example', 'SparkPi', '10'],
        },
    }
]

JOB_FLOW_OVERRIDES = {
    'Name': 'my-demo-cluster',
    'ReleaseLabel': 'emr-5.30.1',
    'Applications': [
        {
            'Name': 'Spark'
        },
    ],
    'Instances': {
        'InstanceGroups': [
            {
                'Name': "Master nodes",
                'Market': 'ON_DEMAND',
                'InstanceRole': 'MASTER',
                'InstanceType': 'm5.xlarge',
                'InstanceCount': 1,
            },
            {
                'Name': "Slave nodes",
                'Market': 'ON_DEMAND',
                'InstanceRole': 'CORE',
                'InstanceType': 'm5.xlarge',
```

```
                    'InstanceCount': 2,
                }
            ],
            'KeepJobFlowAliveWhenNoSteps': False,
            'TerminationProtected': False,
            'Ec2KeyName': 'mykeypair',
        },
        'VisibleToAllUsers': True,
        'JobFlowRole': 'EMR_EC2_DefaultRole',
        'ServiceRole': 'EMR_DefaultRole'
    }

    with DAG(
        dag_id=DAG_ID,
        default_args=DEFAULT_ARGS,
        dagrun_timeout=timedelta(hours=2),
        start_date=days_ago(1),
        schedule_interval='@once',
        tags=['emr'],
    ) as dag:

        cluster_creator = EmrCreateJobFlowOperator(
            task_id='create_job_flow',
            job_flow_overrides=JOB_FLOW_OVERRIDES
        )

        step_adder = EmrAddStepsOperator(
            task_id='add_steps',
            job_flow_id="{{ task_instance.xcom_pull(task_ids='create_job_flow',
key='return_value') }}",
            aws_conn_id='aws_default',
            steps=SPARK_STEPS,
        )

        step_checker = EmrStepSensor(
            task_id='watch_step',
            job_flow_id="{{ task_instance.xcom_pull('create_job_flow',
key='return_value') }}",
            step_id="{{ task_instance.xcom_pull(task_ids='add_steps',
key='return_value')[0] }}",
            aws_conn_id='aws_default',
        )
```

```
        cluster_creator >> step_adder >> step_checker
```

Using Amazon MWAA with Amazon EKS

The following sample demonstrates how to use Amazon Managed Workflows for Apache Airflow with Amazon EKS.

Topics

- Version
- Prerequisites
- Create a public key for Amazon EC2
- Create the cluster
- Create a mwaa namespace
- Create a role for the mwaa namespace
- Create and attach an IAM role for the Amazon EKS cluster
- Create the requirements.txt file
- Create an identity mapping for Amazon EKS
- Create the kubeconfig
- Create a DAG
- Add the DAG and kube_config.yaml to the Amazon S3 bucket
- Enable and trigger the example

Version

- The sample code on this page can be used with **Apache Airflow v1** in Python 3.7.

- You can use the code example on this page with **Apache Airflow v2** in Python 3.10.

Prerequisites

To use the example in this topic, you'll need the following:

- An Amazon MWAA environment.

- eksctl. To learn more, see Install eksctl.
- kubectl. To learn more, see Install and Set Up kubectl. In some case this is installed with eksctl.
- An EC2 key pair in the Region where you create your Amazon MWAA environment. To learn more, see Creating or importing a key pair.

> **Note**
>
> When you use an `eksctl` command, you can include a `--profile` to specify a profile other than the default.

Create a public key for Amazon EC2

Use the following command to create a public key from your private key pair.

```
ssh-keygen -y -f myprivatekey.pem > mypublickey.pub
```

To learn more, see Retrieving the public key for your key pair.

Create the cluster

Use the following command to create the cluster. If you want a custom name for the cluster or to create it in a different Region, replace the name and Region values. You must create the cluster in the same Region where you create the Amazon MWAA environment. Replace the values for the subnets to match the subnets in your Amazon VPC network that you use for Amazon MWAA. Replace the value for the `ssh-public-key` to match the key you use. You can use an existing key from Amazon EC2 that is in the same Region, or create a new key in the same Region where you create your Amazon MWAA environment.

```
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"
```

It takes some time to complete creating the cluster. Once complete, you can verify that the cluster was created successfully and has the IAM OIDC Provider configured by using the following command:

```
eksctl utils associate-iam-oidc-provider \
--region us-west-2 \
--cluster mwaa-eks \
--approve
```

Create a `mwaa` namespace

After confirming that the cluster was successfully created, use the following command to create a namespace for the pods.

```
kubectl create namespace mwaa
```

Create a role for the `mwaa` namespace

After you create the namespace, create a role and role-binding for an Amazon MWAA user on EKS that can run pods in a the MWAA namespace. If you used a different name for the namespace, replace mwaa in `-n mwaa` with the name that you used.

```
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
```

Confirm that the new role can access the Amazon EKS cluster by running the following command. Be sure to use the correct name if you did not use *mwaa*:

```
kubectl get pods -n mwaa --as mwaa-service
```

You should see a message returned that says:

```
No resources found in mwaa namespace.
```

Create and attach an IAM role for the Amazon EKS cluster

You must create an IAM role and then bind it to the Amazon EKS (k8s) cluster so that it can be used for authentication through IAM. The role is used only to log in to the cluster, and does not have any permissions for the console or API calls.

Create a new role for the Amazon MWAA environment using the steps in Amazon MWAA execution role. However, instead of creating and attaching the policies described in that topic, attach the following policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "airflow:PublishMetrics",
            "Resource": "arn:aws:airflow:${MWAA_REGION}:${ACCOUNT_NUMBER}:environment/
${MWAA_ENV_NAME}"
        },
        {
            "Effect": "Deny",
            "Action": "s3:ListAllMyBuckets",
            "Resource": [
                "arn:aws:s3:::{MWAA_S3_BUCKET}",
                "arn:aws:s3:::{MWAA_S3_BUCKET}/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject*",
                "s3:GetBucket*",
                "s3:List*"
            ],
            "Resource": [
                "arn:aws:s3:::{MWAA_S3_BUCKET}",
                "arn:aws:s3:::{MWAA_S3_BUCKET}/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:CreateLogGroup",
                "logs:PutLogEvents",
                "logs:GetLogEvents",
                "logs:GetLogRecord",
                "logs:GetLogGroupFields",
                "logs:GetQueryResults",
```

```
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:${MWAA_REGION}:${ACCOUNT_NUMBER}:log-group:airflow-
${MWAA_ENV_NAME}-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "cloudwatch:PutMetricData",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "sqs:ChangeMessageVisibility",
                "sqs:DeleteMessage",
                "sqs:GetQueueAttributes",
                "sqs:GetQueueUrl",
                "sqs:ReceiveMessage",
                "sqs:SendMessage"
            ],
            "Resource": "arn:aws:sqs:${MWAA_REGION}:*:airflow-celery-*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey",
                "kms:GenerateDataKey*",
                "kms:Encrypt"
            ],
            "NotResource": "arn:aws:kms:*:${ACCOUNT_NUMBER}:key/*",
            "Condition": {
                "StringLike": {
                    "kms:ViaService": [
                        "sqs.${MWAA_REGION}.amazonaws.com"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
```

```
                "eks:DescribeCluster"
            ],
            "Resource": "arn:aws:eks:${MWAA_REGION}:${ACCOUNT_NUMBER}:cluster/
${EKS_CLUSTER_NAME}"
        }
    ]
}
```

After you create role, edit your Amazon MWAA environment to use the role you created as the execution role for the environment. To change the role, edit the environment to use. You select the execution role under **Permissions**.

Known issues:

- There is a known issue with role ARNs with subpaths not being able to authenticate with Amazon EKS. The workaround for this is to create the service role manually rather than using the one created by Amazon MWAA itself. To learn more, see Roles with paths do not work when the path is included in their ARN in the aws-auth configmap
- If Amazon MWAA service listing is not available in IAM you need to choose an alternate service policy, such as Amazon EC2, and then update the role's trust policy to match the following:

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

To learn more, see How to use trust policies with IAM roles.

Create the requirements.txt file

To use the sample code in this section, ensure you've added one of the following database options to your `requirements.txt`. To learn more, see Installing Python dependencies.

Apache Airflow v2

```
kubernetes
apache-airflow[cncf.kubernetes]==3.0.0
```

Apache Airflow v1

```
awscli
kubernetes==12.0.1
```

Create an identity mapping for Amazon EKS

Use the ARN for the role you created in the following command to create an identity mapping for Amazon EKS. Change the Region *your-region* to the Region where you created the environment. Replace the ARN for the role, and finally, replace *mwaa-execution-role* with your environment's execution role.

```
eksctl create iamidentitymapping \
--region your-region \
--cluster mwaa-eks \
--arn arn:aws:iam::111222333444:role/mwaa-execution-role \
--username mwaa-service
```

Create the kubeconfig

Use the following command to create the kubeconfig:

```
aws eks update-kubeconfig \
--region us-west-2 \
--kubeconfig ./kube_config.yaml \
--name mwaa-eks \
--alias aws
```

If you used a specific profile when you ran `update-kubeconfig` you need to remove the `env:` section added to the kube_config.yaml file so that it works correctly with Amazon MWAA. To do so, delete the following from the file and then save it:

```
env:
- name: AWS_PROFILE
  value: profile_name
```

Create a DAG

Use the following code example to create a Python file, such as `mwaa_pod_example.py` for the DAG.

Apache Airflow v2

```
"""
Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the "Software"), to deal in
the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of
the Software, and to permit persons to whom the Software is furnished to do so.
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS
FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER
IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
"""
from airflow import DAG
from datetime import datetime
from airflow.providers.cncf.kubernetes.operators.kubernetes_pod import
 KubernetesPodOperator

default_args = {
   'owner': 'aws',
   'depends_on_past': False,
   'start_date': datetime(2019, 2, 20),
   'provide_context': True
}

dag = DAG(
```

```
   'kubernetes_pod_example', default_args=default_args, schedule_interval=None)

#use a kube_config stored in s3 dags folder for now
kube_config_path = '/usr/local/airflow/dags/kube_config.yaml'

podRun = KubernetesPodOperator(
                        namespace="mwaa",
                        image="ubuntu:18.04",
                        cmds=["bash"],
                        arguments=["-c", "ls"],
                        labels={"foo": "bar"},
                        name="mwaa-pod-test",
                        task_id="pod-task",
                        get_logs=True,
                        dag=dag,
                        is_delete_operator_pod=False,
                        config_file=kube_config_path,
                        in_cluster=False,
                        cluster_context='aws'
                        )
```

Apache Airflow v1

```
"""
Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the "Software"), to deal in
the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of
the Software, and to permit persons to whom the Software is furnished to do so.
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS
FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER
IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
"""
from airflow import DAG
from datetime import datetime
from airflow.contrib.operators.kubernetes_pod_operator import KubernetesPodOperator

default_args = {
   'owner': 'aws',
```

```
    'depends_on_past': False,
    'start_date': datetime(2019, 2, 20),
    'provide_context': True
}

dag = DAG(
    'kubernetes_pod_example', default_args=default_args, schedule_interval=None)

#use a kube_config stored in s3 dags folder for now
kube_config_path = '/usr/local/airflow/dags/kube_config.yaml'

podRun = KubernetesPodOperator(
                        namespace="mwaa",
                        image="ubuntu:18.04",
                        cmds=["bash"],
                        arguments=["-c", "ls"],
                        labels={"foo": "bar"},
                        name="mwaa-pod-test",
                        task_id="pod-task",
                        get_logs=True,
                        dag=dag,
                        is_delete_operator_pod=False,
                        config_file=kube_config_path,
                        in_cluster=False,
                        cluster_context='aws'
                        )
```

Add the DAG and `kube_config.yaml` to the Amazon S3 bucket

Put the DAG you created and the `kube_config.yaml` file into the Amazon S3 bucket for the Amazon MWAA environment. You can put files into your bucket using either the Amazon S3 console or the AWS Command Line Interface.

Enable and trigger the example

In Apache Airflow, enable the example and then trigger it.

After it runs and completes successfully, use the following command to verify the pod:

```
kubectl get pods -n mwaa
```

You should see output similar to the following:

```
NAME READY STATUS RESTARTS AGE
mwaa-pod-test-aa11bb22cc3344445555666677778888 0/1 Completed 0 2m23s
```

You can then verify the output of the pod with the following command. Replace the name value with the value returned from the previous command:

```
kubectl logs -n mwaa mwaa-pod-test-aa11bb22cc3344445555666677778888
```

Connecting to Amazon ECS using the ECSOperator

The topic describes how you can use the ECSOperator to connect to an Amazon Elastic Container Service (Amazon ECS) container from Amazon MWAA. In the following steps, you'll add the required permissions to your environment's execution role, use a AWS CloudFormation template to create an Amazon ECS Fargate cluster, and finally create and upload a DAG that connects to your new cluster.

Topics

- Version
- Prerequisites
- Permissions
- Create an Amazon ECS cluster
- Code sample

Version

- You can use the code example on this page with **Apache Airflow v2** in Python 3.10.

Prerequisites

To use the sample code on this page, you'll need the following:

- An Amazon MWAA environment.

Permissions

- The execution role for your environment needs permission to run tasks in Amazon ECS. You can either attach the AmazonECS_FullAccess AWS-managed policy to your execution role, or create and attach the following policy to your execution role.

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

- In addition to adding the required premissions to run tasks in Amazon ECS, you must also modify the CloudWatch Logs policy statement in your Amazon MWAA execution role to allow access to the Amazon ECS task log group as shown in the following. The Amazon ECS log group is created by the AWS CloudFormation template in the section called “Create an Amazon ECS cluster”.

```
{
    "Effect": "Allow",
    "Action": [
        "logs:CreateLogStream",
```

```
        "logs:CreateLogGroup",
        "logs:PutLogEvents",
        "logs:GetLogEvents",
        "logs:GetLogRecord",
        "logs:GetLogGroupFields",
        "logs:GetQueryResults"
    ],
    "Resource": [
        "arn:aws:logs:region:account-id:log-group:airflow-environment-name-*",
        "arn:aws:logs:*:*:log-group:ecs-mwaa-group:*"
    ]
}
```

For more information about the Amazon MWAA execution role, and how to attach a policy, see Execution role.

Create an Amazon ECS cluster

Using the following AWS CloudFormation template, you will build an Amazon ECS Fargate cluster to use with your Amazon MWAA workflow. For more information, see Creating a task definition in the *Amazon Elastic Container Service Developer Guide*.

1. Create a JSON file with the following code and save it as `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. In your command prompt, use the following AWS CLI command to create a new stack. You must replace the values `SecurityGroups` and `SubnetIds` with values for your Amazon MWAA environment's security groups and subnets.

```
$ 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
```

Alternatively, you can use the following shell script. The script retrieves the required values for your environment's security groups, and subnets using the `get-environment` AWS CLI command, then creates the stack accordingly. To run the script, do the following.

a. Copy, and save the script as `ecs-stack-helper.sh` in the same directory as your AWS CloudFormation template.

```
#!/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. Run the script using the following commands. Replace `environment-name` and `stack-name` with your information.

```
$ chmod +x ecs-stack-helper.sh
$ ./ecs-stack-helper.bash environment-name stack-name
```

If successful, you'll see the following output displaying your new AWS CloudFormation stack ID.

```
{
```

```
    "StackId": "arn:aws:cloudformation:us-west-2:123456789012:stack/my-ecs-
stack/123456e7-8ab9-01cd-b2fb-36cce63786c9"
}
```

After your AWS CloudFormation stack is completed and AWS has provisioned your Amazon ECS resources, you're ready to create and upload your DAG.

Code sample

1. Open a command prompt, and navigate to the directory where your DAG code is stored. For example:

```
cd dags
```

2. Copy the contents of the following code sample and save locally as `mwaa-ecs-operator.py`, then upload your new DAG to 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

In the example DAG, for `awslogs_group`, you might need to modify the log group with the name for your Amazon ECS task log group. The example assumes a log group named `mwaa-ecs-zero`. For `awslogs_stream_prefix`, use the Amazon ECS task log stream prefix. The example assumes a log stream prefix, `ecs`.

3. Run the following AWS CLI command to copy the DAG to your environment's bucket, then trigger the DAG using the Apache Airflow UI.

```
$ aws s3 cp your-dag.py s3://your-environment-bucket/dags/
```

4. If successful, you'll see output similar to the following in the task logs for `ecs_operator_task` in the `ecs_fargate_dag` DAG:

```
[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
```

Using dbt with Amazon MWAA

This topic demonstrates how you can use dbt and Postgres with Amazon MWAA. In the following steps, you'll add the required dependencies to your `requirements.txt`, and upload a sample dbt project to your environment's Amazon S3 bucket. Then, you'll use a sample DAG to verify that Amazon MWAA has installed the dependencies, and finally use the `BashOperator` to run the dbt project.

Topics

- Version
- Prerequisites
- Dependencies
- Upload a dbt project to Amazon S3
- Use a DAG to verify dbt dependency installation
- Use a DAG to run a dbt project

Version

- You can use the code example on this page with **Apache Airflow v2** in Python 3.10.

Prerequisites

Before you can complete the following steps, you'll need the following:

- An Amazon MWAA environment using Apache Airflow v2.2.2. This sample was written, and tested with v2.2.2. You might need to modify the sample to use with other Apache Airflow versions.
- A sample dbt project. To get started using dbt with Amazon MWAA, you can create a fork and clone the dbt starter project from the dbt-labs GitHub repository.

Dependencies

To use Amazon MWAA with dbt, add the following startup script to your environment. To learn more, see Using a startup script with Amazon MWAA.

```
#!/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
```

In the following sections, you'll upload your dbt project directory to Amazon S3 and run a DAG that validates whether Amazon MWAA has successfully installed the required dbt dependencies.

Upload a dbt project to Amazon S3

To be able to use a dbt project with your Amazon MWAA environment, you can upload the entire project directory to your environment's `dags` folder. When the environment updates, Amazon MWAA downloads the dbt directory to the local `usr/local/airflow/dags/` folder.

To upload a dbt project to Amazon S3

1. Navigate to the directory where you cloned the dbt starter project.
2. Run the following Amazon S3 AWS CLI command to recursively copy the content of the project to your environment's `dags` folder using the `--recursive` parameter. The command creates a sub-directory called `dbt` that you can use for all of your dbt projects. If the sub-directory already exists, the project files are copied into the existing directory, and a new directory is not created. The command also creates a sub-directory within the `dbt` directory for this specific starter project.

```
$ aws s3 cp dbt-starter-project s3://mwaa-bucket/dags/dbt/dbt-starter-project --
recursive
```

You can use different names for project sub-directories to organize multiple dbt projects within the parent dbt directory.

Use a DAG to verify dbt dependency installation

The following DAG uses a BashOperator and a bash command to verify whether Amazon MWAA has successfully installed the dbt dependencies specified in requirements.txt.

```
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""
    )
```

Do the following to view task logs and verify that dbt and its dependencies have been installed.

1. Navigate to the Amazon MWAA console, then choose **Open Airflow UI** from the list of available environments.
2. On the Apache Airflow UI, find the dbt-installation-test DAG from the list, then choose the date under the Last Run column to open the last successful task.
3. Using **Graph View**, choose the bash_command task to open the task instance details.
4. Choose **Log** to open the task logs, then verify that the logs successfully list the dbt version we specified in requirements.txt.

Use a DAG to run a dbt project

The following DAG uses a BashOperator to copy the dbt projects you uploaded to Amazon S3 from the local usr/local/airflow/dags/ directory to the write-accessible /tmp directory,

then runs the dbt project. The bash commands assume a starter dbt project titled `dbt-starter-project`. Modify the directory name according to the name of your project directory.

```
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 blogs and tutorials

- Working with Amazon EKS and Amazon MWAA for Apache Airflow v2.x

Best practices for Amazon Managed Workflows for Apache Airflow

This guide describes the best practices we recommend when using Amazon Managed Workflows for Apache Airflow.

Topics

- Performance tuning for Apache Airflow on Amazon MWAA
- Managing Python dependencies in requirements.txt

Performance tuning for Apache Airflow on Amazon MWAA

This topic describes how to tune the performance of an Amazon Managed Workflows for Apache Airflow environment using Using Apache Airflow configuration options on Amazon MWAA.

Contents

- Adding an Apache Airflow configuration option
- Apache Airflow scheduler
  - Parameters
  - Limits
- DAG folders
  - Parameters
- DAG files
  - Parameters
- Tasks
  - Parameters

Adding an Apache Airflow configuration option

The following procedure walks you through the steps of adding an Airflow configuration option to your environment.

1. Open the Environments page on the Amazon MWAA console.

2. Choose an environment.
3. Choose **Edit**.
4. Choose **Next**.
5. Choose **Add custom configuration** in the **Airflow configuration options** pane.
6. Choose a configuration from the dropdown list and enter a value, or type a custom configuration and enter a value.
7. Choose **Add custom configuration** for each configuration you want to add.
8. Choose **Save**.

To learn more, see Using Apache Airflow configuration options on Amazon MWAA.

Apache Airflow scheduler

The Apache Airflow scheduler is a core component of Apache Airflow. An issue with the scheduler can prevent DAGs from being parsed and tasks from being scheduled. For more information about Apache Airflow scheduler tuning, see Fine-tuning your scheduler performance in the Apache Airflow documentation website.

Parameters

This section describes the configuration options available for the Apache Airflow scheduler and their use cases.

Apache Airflow v2

Version	Configuration option	Default	Description	Use case
v2	celery.sync_parallelism	1	The number of processes the Celery Executor uses to sync task state.	You can use this option to prevent queue conflicts by limiting the processes the Celery Executor uses. By default,

Version	Configuration option	Default	Description	Use case
				a value is set to 1 to prevent errors in delivering task logs to CloudWatch Logs. Setting the value to `0` means using the maximum number of processes, but might cause errors when delivering task logs.

Version	Configuration option	Default	Description	Use case
v2	scheduler .idle_sleep_time	1	The number of seconds to wait between consecuti ve DAG file processing in the *Scheduler* "loop."	You can use this option to free up CPU usage on the *Scheduler* by **increasing** the time the *Scheduler* sleeps after it's finished retrieving DAG parsing results, finding and queuing tasks, and executing queued tasks in the *Executor*. Increasing this value consumes the number of *Scheduler* threads run on an environment in `scheduler .parsing_ processes` for Apache Airflow v2 and `scheduler .max_thre ads` for Apache Airflow v1. This may

Version	Configuration option	Default	Description	Use case
				reduce the capacity of the *Schedulers* to parse DAGs, and increase the time it takes for DAGs to appear in the *Web server*.
v2	scheduler.max_dagruns_to_create_per_loop	10	The maximum number of DAGs to create *DagRuns* for per *Scheduler* "loop."	You can use this option to free up resources for scheduling tasks by **decreasing** the maximum number of *DagRuns* for the *Scheduler* "loop."

Version	Configuration option	Default	Description	Use case
v2	scheduler.parsing_processes	Set using the following formula: `(2 * number of vCPUs) - 1` by default.	The number of threads the *Scheduler* can run in parallel to schedule DAGs.	You can use this option to free up resources by **decreasing** the number of processes the *Scheduler* runs in parallel to parse DAGs. We recommend keeping this number low if DAG parsing is impacting task scheduling. You **must** specify a value that's less than the vCPU count on your environment. To learn more, see Limits.

Limits

This section describes the limits you should consider when adjusting the default parameters for the scheduler.

scheduler.parsing_processes, scheduler.max_threads

Two threads are allowed per vCPU for an environment class. At least one thread must be reserved for the scheduler for an environment class. If you notice a delay in tasks being scheduled, you may need to increase your environment class. For example, a large environment has a 4 vCPU Fargate container instance for its scheduler. This means that a maximum of 7 total

threads are available to use for other processes. That is, two threads multiplied four vCPUs, minus one for the scheduler itself. The value you specify in `scheduler.max_threads` and `scheduler.parsing_processes` must not exceed the number of threads available for an environment class (as shown, below:

- **mw1.small** – Must not exceed 1 thread for other processes. The remaining thread is reserved for the *Scheduler*.
- **mw1.medium** – Must not exceed 3 threads for other processes. The remaining thread is reserved for the *Scheduler*.
- **mw1.large** – Must not exceed 7 threads for other processes. The remaining thread is reserved for the *Scheduler*.

DAG folders

The Apache Airflow *Scheduler* continuously scans the DAGs folder on your environment. Any contained `plugins.zip` files, or Python (`.py`) files containing "airflow" import statements. Any resulting Python DAG objects are then placed into a *DagBag* for that file to be processed by the *Scheduler* to determine what, if any, tasks need to be scheduled. Dag file parsing occurs regardless of whether the files contain any viable DAG objects.

Parameters

This section describes the configuration options available for the DAGs folder and their use cases.

Apache Airflow v2

Version	Configuration option	Default	Description	Use case
v2	scheduler .dag_dir_ list_interval	300 seconds	The number of seconds the DAGs folder should be scanned for new files.	You can use this option to free up resources by **increasin g** the number of seconds to parse the DAGs folder. We recommend

Version	Configuration option	Default	Description	Use case
				increasing this value if you're seeing long parsing times in `total_parse_time` `metrics`, which may be due to a large number of files in your DAGs folder.

Version	Configuration option	Default	Description	Use case
v2	scheduler.min_file_process_interval	30 seconds	The number of seconds after which the scheduler parses a DAG and updates to the DAG are reflected.	You can use this option to free up resources by **increasing** the number of seconds that the scheduler waits before parsing a DAG. For example, if you specify a value of `30`, the DAG file is parsed after every 30 seconds. We recommend keeping this number high to decrease the CPU usage on your environment.

DAG files

As part of the Apache Airflow scheduler loop, individual DAG files are parsed to extract DAG Python objects. In Apache Airflow v2 and above, the scheduler parses a maximum of number of parsing processes at the same time. The number of seconds specified in `scheduler.min_file_process_interval` must pass before the same file is parsed again.

Parameters

This section describes the configuration options available for Apache Airflow DAG files and their use cases.

Apache Airflow v2

Version	Configuration option	Default	Description	Use case
v2	core.dag_file_processor_timeout	50 seconds	The number of seconds before the *DagFilePr ocessor* times out processing a DAG file.	You can use this option to free up resources by **increasin g** the time it takes before the *DagFilePr ocessor* times out. We recommend increasing this value if you're seeing timeouts in your DAG processing logs that result in no viable DAGs being loaded.
v2	core.dagbag_import_timeout	30 seconds	The number of seconds before importing a Python file times out.	You can use this option to free up resources by **increasin g** the time it takes before the *Scheduler* times out while importing a

Version	Configuration option	Default	Description	Use case
				Python file to extract the DAG objects. This option is processed as part of the *Scheduler* "loop," and must contain a value lower than the value specified in `core.dag_file_processor_timeout` .

Version	Configuration option	Default	Description	Use case
v2	core.min_serialized_dag_update_interval	30	The minimum number of seconds after which serialized DAGs in the database are updated.	You can use this option to free up resources by **increasing** the number of seconds after which serialized DAGs in the database are updated. We recommend increasing this value if you have a large number of DAGs, or complex DAGs. Increasing this value reduces the load on the *Scheduler* and the database as DAGs are serialized.

Version	Configuration option	Default	Description	Use case
v2	core.min_serialized_dag_fetch_interval	10	The number of seconds a serialized DAG is re-fetched from the database when already loaded in the DagBag.	You can use this option to free up resources by **increasing** the number of seconds a serialized DAG is re-fetched. The value must be higher than the value specified in `core.min_serialized_dag_update_interval` to reduce database "write" rates. Increasing this value reduces the load on the *Web server* and the database as DAGs are serialized.

Tasks

The Apache Airflow scheduler and workers are both involved in queuing and de-queuing tasks. The scheduler takes parsed tasks ready to schedule from a **None** status to a **Scheduled** status. The executor, also running on the scheduler container in Fargate, queues those tasks and sets their status to **Queued**. When the workers have capacity, it takes the task from the queue and sets the

status to **Running**, which subsequently changes its status to **Success** or **Failed** based on whether the task succeeds or fails.

Parameters

This section describes the configuration options available for Apache Airflow tasks and their use cases.

The default configuration options that Amazon MWAA overrides are marked in *`red`*.

Apache Airflow v2

Version	Configuration option	Default	Description	Use case
v2	core.parallelism	Dynamically set based on `(maxWorkers * maxCeleryWorkers) / schedulers * 1.5`.	The maximum number of task instances that can have a status of "Running."	You can use this option to free up resources by **increasing** the number of task instances that can run simultaneously. The value specified should be the number of available *Workers* "times" the *Workers* task density. We recommend changing this value only when you're seeing a large number of tasks stuck in the “Running”

Version	Configuration option	Default	Description	Use case
				or “Queued” state.
v2	core.dag_concurrency	10000	The number of task instances allowed to run concurrently for each DAG.	You can use this option to free up resources by **increasing** the number of task instances allowed to run concurrently. For example, if you have one hundred DAGs with ten parallel tasks, and you want all DAGs to run concurrently, you can calculate the maximum parallelism as the number of available *Workers* "times" the *Workers* task density in `celery.worker_concurrency`, divided by the number of DAGs (e.g. 100).

Version	Configuration option	Default	Description	Use case
v2	core.execute_tasks_new_python_interpreter	`True`	Determines whether Apache Airflow executes tasks by forking the parent process, or by creating a new Python process.	When set to `True`, Apache Airflow recognizes changes you make to your plugins as a new Python process so created to execute tasks.
v2	celery.worker_concurrency	N/A	Amazon MWAA overrides the Airflow base install for this option to scale *Workers* as part of its autoscaling component.	*Any value specified for this option is ignored.*

Version	Configuration option	Default	Description	Use case
v2	celery.worker_autoscale	**mw1.micro** - 3,0 **mw1.small** - 5,0 **mw1.medium** - 10,0 **mw1.large** - 20,0 **mw1.xlarge** - 40,0 **mw1.2xlarge** - 80,0	The task concurrency for *Workers*.	You can use this option to free up resources by **reducing** the `maximum`, `minimum` task concurrency of *Workers*. *Workers* accept up to the `maximum` concurrent tasks configured, regardless of whether there are sufficient resources to do so. If tasks are scheduled without sufficient resources, the tasks immediately fail. We recommend changing this value for resource-intensive tasks by reducing the values to be less than

Version	Configuration option	Default	Description	Use case
				the defaults to allow more capacity per task.

Managing Python dependencies in requirements.txt

This topic describes how to install and manage Python dependencies in a `requirements.txt` file for an Amazon Managed Workflows for Apache Airflow environment.

Contents

- Testing DAGs using the Amazon MWAA CLI utility
- Installing Python dependencies using PyPi.org Requirements File Format
  - Option one: Python dependencies from the Python Package Index
  - Option two: Python wheels (.whl)
    - Using the plugins.zip file on an Amazon S3 bucket
    - Using a WHL file hosted on a URL
    - Creating a WHL files from a DAG
  - Option three: Python dependencies hosted on a private PyPi/PEP-503 Compliant Repo
- Enabling logs on the Amazon MWAA console
- Viewing logs on the CloudWatch Logs console
- Viewing errors in the Apache Airflow UI
  - Logging into Apache Airflow
- Example requirements.txt scenarios

Testing DAGs using the Amazon MWAA CLI utility

- The command line interface (CLI) utility replicates an Amazon Managed Workflows for Apache Airflow environment locally.

- The CLI builds a Docker container image locally that's similar to an Amazon MWAA production image. This allows you to run a local Apache Airflow environment to develop and test DAGs, custom plugins, and dependencies before deploying to Amazon MWAA.
- To run the CLI, see the aws-mwaa-local-runner on GitHub.

Installing Python dependencies using PyPi.org Requirements File Format

The following section describes the different ways to install Python dependencies according to the PyPi.org Requirements File Format.

Option one: Python dependencies from the Python Package Index

The following section describes how to specify Python dependencies from the Python Package Index in a `requirements.txt` file.

Apache Airflow v2

1. **Test locally**. Add additional libraries iteratively to find the right combination of packages and their versions, before creating a `requirements.txt` file. To run the Amazon MWAA CLI utility, see the aws-mwaa-local-runner on GitHub.
2. **Review the Apache Airflow package extras**. To view a list of the packages installed for Apache Airflow v2 on Amazon MWAA, see Amazon MWAA local runner `requirements.txt` on the GitHub website.
3. **Add a constraints statement**. Add the constraints file for your Apache Airflow v2 environment at the top of your `requirements.txt` file. Apache Airflow constraints files specify the provider versions available at the time of a Apache Airflow release.

   Beginning with Apache Airflow v2.7.2, your requirements file must include a `--constraint` statement. If you do not provide a constraint, Amazon MWAA will specify one for you to ensure the packages listed in your requirements are compatible with the version of Apache Airflow you are using.

   In the following example, replace *`{environment-version}`* with your environment's version number, and *`{Python-version}`* with the version of Python that's compatible with your environment.

For information on the version of Python compatible with your Apache Airflow environment, see Apache Airflow Versions.

```
--constraint "https://raw.githubusercontent.com/apache/airflow/
constraints-{Airflow-version}/constraints-{Python-version}.txt"
```

If the constraints file determines that `xyz==1.0` package is not compatible with other packages in your environment, `pip3 install` will fail in order to prevent incompatible libraries from being installed to your environment. If installation fails for any packages, you can view error logs for each Apache Airflow component (the scheduler, worker, and web server) in the corresponding log stream on CloudWatch Logs. For more information on log types, see the section called "Viewing Airflow logs".

4. **Apache Airflow packages**. Add the package extras and the version (==). This helps to prevent packages of the same name, but different version, from being installed on your environment.

```
apache-airflow[package-extra]==2.5.1
```

5. **Python libraries**. Add the package name and the version (==) in your `requirements.txt` file. This helps to prevent a future breaking update from PyPi.org from being automatically applied.

```
library == version
```

Example Boto3 and psycopg2-binary

This example is provided for demonstration purposes. The boto and psycopg2-binary libraries are included with the Apache Airflow v2 base install and don't need to be specified in a `requirements.txt` file.

```
boto3==1.17.54
boto==2.49.0
botocore==1.20.54
psycopg2-binary==2.8.6
```

If a package is specified without a version, Amazon MWAA installs the latest version of the package from PyPi.org. This version may conflict with other packages in your `requirements.txt`.

Option two: Python wheels (.whl)

A Python wheel is a package format designed to ship libraries with compiled artifacts. There are several benefits to wheel packages as a method to install dependencies in Amazon MWAA:

- **Faster installation** – the WHL files are copied to the container as a single ZIP, and then installed locally, without having to download each one.
- **Fewer conflicts** – You can determine version compatibility for your packages in advance. As a result, there is no need for `pip` to recursively work out compatible versions.
- **More resilience** – With externally hosted libraries, downstream requirements can change, resulting in version incompatibility between containers on a Amazon MWAA environment. By not depending on an external source for dependencies, every container on has have the same libraries regardless of when the each container is instantiated.

We recommend the following methods to install Python dependencies from a Python wheel archive (`.whl`) in your `requirements.txt`.

Methods

- Using the plugins.zip file on an Amazon S3 bucket
- Using a WHL file hosted on a URL
- Creating a WHL files from a DAG

Using the `plugins.zip` file on an Amazon S3 bucket

The Apache Airflow scheduler, workers, and web server (for Apache Airflow v2.2.2 and later) look for custom plugins during startup on the AWS-managed Fargate container for your environment at `/usr/local/airflow/plugins/*`. This process begins prior to Amazon MWAA's `pip3 install -r requirements.txt` for Python dependencies and Apache Airflow service startup. A `plugins.zip` file be used for any files that you don't want continuously changed during environment execution, or that you may not want to grant access to users that write DAGs. For example, Python library wheel files, certificate PEM files, and configuration YAML files.

The following section describes how to install a wheel that's in the `plugins.zip` file on your Amazon S3 bucket.

1. **Download the necessary WHL files** You can use `pip download` with your existing `requirements.txt` on the Amazon MWAA local-runner or another Amazon Linux 2 container to resolve and download the necessary Python wheel files.

   ```
   $ pip3 download -r "$AIRFLOW_HOME/dags/requirements.txt" -d "$AIRFLOW_HOME/plugins"
   $ cd "$AIRFLOW_HOME/plugins"
   $ zip "$AIRFLOW_HOME/plugins.zip" *
   ```

2. **Specify the path in your `requirements.txt`**. Specify the plugins directory at the top of your requirements.txt using `--find-links` and instruct `pip` not to install from other sources using `--no-index`, as shown in the following

   ```
   --find-links /usr/local/airflow/plugins
   --no-index
   ```

   Example wheel in requirements.txt

   The following example assumes you've uploaded the wheel in a `plugins.zip` file at the root of your Amazon S3 bucket. For example:

   ```
   --find-links /usr/local/airflow/plugins
   --no-index

   numpy
   ```

   Amazon MWAA fetches the `numpy-1.20.1-cp37-cp37m-manylinux1_x86_64.whl` wheel from the `plugins` folder and installs it on your environment.

Using a WHL file hosted on a URL

The following section describes how to install a wheel that's hosted on a URL. The URL must either be publicly accessible, or accessible from within the custom Amazon VPC you specified for your Amazon MWAA environment.

- **Provide a URL**. Provide the URL to a wheel in your `requirements.txt`.

Example wheel archive on a public URL

The following example downloads a wheel from a public site.

```
--find-links https://files.pythonhosted.org/packages/
--no-index
```

Amazon MWAA fetches the wheel from the URL you specified and installs them on your environment.

> **Note**
>
> URLs are not accessible from private web servers installing requirements in Amazon MWAA v2.2.2 and later.

Creating a WHL files from a DAG

If you have a private web server using Apache Airflow v2.2.2 or later and you're unable to install requirements because your environment does not have access to external repositories, you can use the following DAG to take your existing Amazon MWAA requirements and package them on Amazon S3:

```
from airflow import DAG
from airflow.operators.bash_operator import BashOperator
from airflow.utils.dates import days_ago

S3_BUCKET = 'my-s3-bucket'
S3_KEY = 'backup/plugins_whl.zip'

with DAG(dag_id="create_whl_file", schedule_interval=None, catchup=False,
 start_date=days_ago(1)) as dag:
    cli_command = BashOperator(
        task_id="bash_command",
        bash_command=f"mkdir /tmp/whls;pip3 download -r /usr/local/airflow/
requirements/requirements.txt -d /tmp/whls;zip -j /tmp/plugins.zip /tmp/whls/*;aws s3
 cp /tmp/plugins.zip s3://{S3_BUCKET}/{S3_KEY}"
    )
```

After running the DAG, use this new file as your Amazon MWAA `plugins.zip`, optionally, packaged with other plugins. Then, update your `requirements.txt` preceded by `--find-links /usr/local/airflow/plugins` and `--no-index` without adding `--constraint`.

This method allows you to use the same libraries offline.

Option three: Python dependencies hosted on a private PyPi/PEP-503 Compliant Repo

The following section describes how to install an Apache Airflow extra that's hosted on a private URL with authentication.

1. Add your user name and password as Apache Airflow configuration options. For example:
   - `foo.user` : *YOUR_USER_NAME*
   - `foo.pass` : *YOUR_PASSWORD*
2. Create your `requirements.txt` file. Substitute the placeholders in the following example with your private URL, and the username and password you've added as Apache Airflow configuration options. For example:

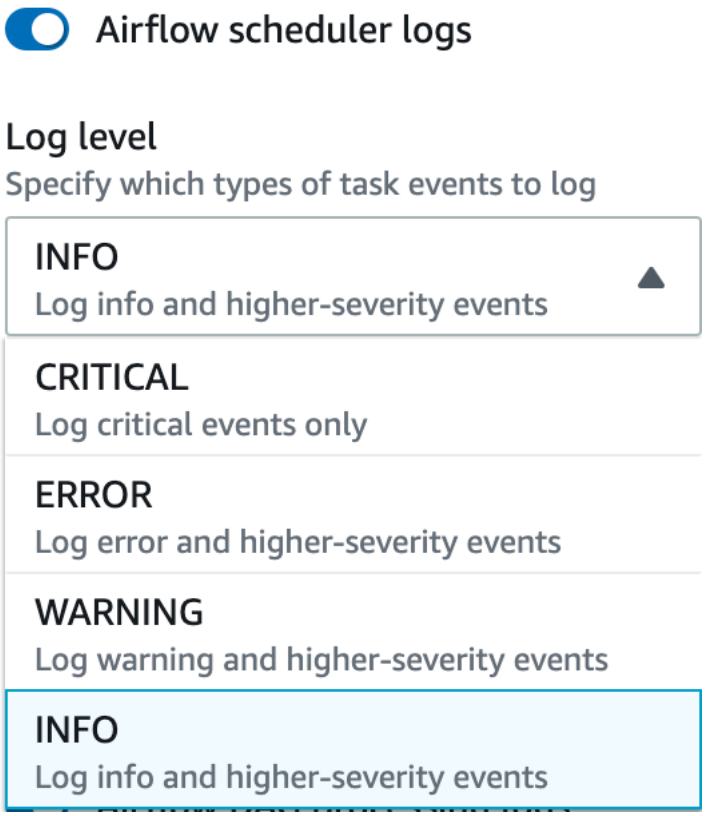
   ```
   --index-url https://${AIRFLOW__FOO__USER}:${AIRFLOW__FOO__PASS}@my.privatepypi.com
   ```
3. Add any additional libraries to your `requirements.txt` file. For example:

   ```
   --index-url https://${AIRFLOW__FOO__USER}:${AIRFLOW__FOO__PASS}@my.privatepypi.com
   my-private-package==1.2.3
   ```

Enabling logs on the Amazon MWAA console

The execution role for your Amazon MWAA environment needs permission to send logs to CloudWatch Logs. To update the permissions of an execution role, see Amazon MWAA execution role.

You can enable Apache Airflow logs at the `INFO`, `WARNING`, `ERROR`, or `CRITICAL` level. When you choose a log level, Amazon MWAA sends logs for that level and all higher levels of severity. For example, if you enable logs at the `INFO` level, Amazon MWAA sends `INFO` logs and `WARNING`, `ERROR`, and `CRITICAL` log levels to CloudWatch Logs. We recommend enabling Apache Airflow logs at the `INFO` level for the *Scheduler* to view logs received for the `requirements.txt`.



Viewing logs on the CloudWatch Logs console

You can view Apache Airflow logs for the *Scheduler* scheduling your workflows and parsing your `dags` folder. The following steps describe how to open the log group for the *Scheduler* on the Amazon MWAA console, and view Apache Airflow logs on the CloudWatch Logs console.

To view logs for a `requirements.txt`

1. Open the Environments page on the Amazon MWAA console.
2. Choose an environment.
3. Choose the **Airflow scheduler log group** on the **Monitoring** pane.
4. Choose the `requirements_install_ip` log in **Log streams**.
5. You should see the list of packages that were installed on the environment at `/usr/local/airflow/.local/bin`. For example:

```
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. Review the list of packages and whether any of these encountered an error during installation. If something went wrong, you may see an error similar to the following:

```
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))
```

Viewing errors in the Apache Airflow UI

You may also want to check your Apache Airflow UI to identify whether an error may be related to another issue. The most common error you may encounter with Apache Airflow on Amazon MWAA is:

```
Broken DAG: No module named x
```

If you see this error in your Apache Airflow UI, you're likely missing a required dependency in your `requirements.txt` file.

Logging into Apache Airflow

You need Apache Airflow UI access policy: AmazonMWAAWebServerAccess permissions for your AWS account in AWS Identity and Access Management (IAM) to view your Apache Airflow UI.

To access your Apache Airflow UI

1. Open the Environments page on the Amazon MWAA console.
2. Choose an environment.
3. Choose **Open Airflow UI**.

Example `requirements.txt` scenarios

You can mix and match different formats in your `requirements.txt`. The following example uses a combination of the different ways to install extras.

Example Extras on PyPi.org and a public URL

You need to use the `--index-url` option when specifying packages from PyPi.org, in addition to packages on a public URL, such as custom PEP 503 compliant repo URLs.

```
aws-batch == 0.6
phoenix-letter >= 0.3

--index-url http://dist.repoze.org/zope2/2.10/simple
    zopelib
```

Monitoring and metrics for Amazon Managed Workflows for Apache Airflow

Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon Managed Workflows for Apache Airflow and your AWS solution. We recommend collecting monitoring data from all parts of your AWS solution so that you can more easily debug a multi-point failure if one occurs. This topic describes what resources AWS provides for monitoring your Amazon MWAA environment and responding to potential events.

> **Note**
>
> Apache Airflow metrics and logging are subject to standard Amazon CloudWatch pricing.

For more information about monitoring Apache Airflow, see Logging & Monitoring in the Apache Airflow documentation website.

Sections

- Monitoring overview on Amazon MWAA
- Viewing audit logs in AWS CloudTrail
- Viewing Airflow logs in Amazon CloudWatch
- Monitoring dashboards and alarms on Amazon MWAA
- Apache Airflow v2 environment metrics in CloudWatch
- Container, queue, and database metrics for Amazon MWAA

Monitoring overview on Amazon MWAA

This page describes the AWS services used to monitor an Amazon Managed Workflows for Apache Airflow environment.

Contents

- Amazon CloudWatch overview
- AWS CloudTrail overview

Amazon CloudWatch overview

CloudWatch is a metrics repository for AWS services that allows you to retrieve statistics based on the metrics and dimensions published by a service. You can use these metrics to configure alarms, calculate statistics and then present the data in a dashboard that helps you assess the health of your environment in the Amazon CloudWatch console.

Apache Airflow is already set-up to send StatsD metrics for an Amazon Managed Workflows for Apache Airflow environment to Amazon CloudWatch.

To learn more, see What is Amazon CloudWatch?.

AWS CloudTrail overview

CloudTrail is an auditing service that provides a record of actions taken by a user, role, or an AWS service in Amazon MWAA. Using the information collected by CloudTrail, you can determine the request that was made to Amazon MWAA, the IP address from which the request was made, who made the request, when it was made, and additional details available in audit logs.

To learn more, see What is AWS CloudTrail?.

Viewing audit logs in AWS CloudTrail

AWS CloudTrail is enabled on your AWS account when you create it. CloudTrail logs the activity taken by an IAM entity or an AWS service, such as Amazon Managed Workflows for Apache Airflow, which is recorded as a CloudTrail event. You can view, search, and download the past 90 days of event history in the CloudTrail console. CloudTrail captures all events on the Amazon MWAA console and all calls to Amazon MWAA APIs. It doesn't capture read-only actions, such as `GetEnvironment`, or the `PublishMetrics` action. This page describes how to use CloudTrail to monitor events for Amazon MWAA.

Contents

- Creating a trail in CloudTrail
- Viewing events with CloudTrail Event History
- Example trail for CreateEnvironment
- What's next?

Creating a trail in CloudTrail

You need to create a trail to view an ongoing record of events in your AWS account, including events for Amazon MWAA. A trail enables CloudTrail to deliver log files to an Amazon S3 bucket. If you do not create a trail, you can still view available event history in the CloudTrail console. For example, using the information collected by CloudTrail, you can determine the request that was made to Amazon MWAA, the IP address from which the request was made, who made the request, when it was made, and additional details. To learn more, see the Creating a trail for your AWS account.

Viewing events with CloudTrail Event History

You can troubleshoot operational and security incidents over the past 90 days in the CloudTrail console by viewing event history. For example, you can view events related to the creation, modification, or deletion of resources (such as IAM users or other AWS resources) in your AWS account on a per-region basis. To learn more, see the Viewing Events with CloudTrail Event History.

1. Open the CloudTrail console.
2. Choose **Event history**.
3. Select the events you want to view, and then choose **Compare event details**.

Example trail for `CreateEnvironment`

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify.

CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, such as the date and time of the action, or request parameters. CloudTrail log files are not an ordered stack trace of the public API calls, and don't appear in any specific order. The following example is a log entry for the `CreateEnvironment` action that is denied due to lacking permissions. The values in `AirflowConfigurationOptions` have been redacted for privacy.

```
{
    "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"
}
```

What's next?

- Learn how to configure other AWS services for the event data collected in CloudTrail logs in CloudTrail Supported Services and Integrations.
- Learn how to be notified when CloudTrail publishes new log files to an Amazon S3 bucket in Configuring Amazon SNS Notifications for CloudTrail.

Viewing Airflow logs in Amazon CloudWatch

Amazon MWAA can send Apache Airflow logs to Amazon CloudWatch. You can view logs for multiple environments from a single location to easily identify Apache Airflow task delays or workflow errors without the need for additional third-party tools. Apache Airflow logs need to be enabled on the Amazon Managed Workflows for Apache Airflow console to view Apache Airflow DAG processing, tasks, *Web server*, *Worker* logs in CloudWatch.

Contents

- Pricing
- Before you begin
- Log types
- Enabling Apache Airflow logs
- Viewing Apache Airflow logs
- Example scheduler logs
- What's next?

Pricing

- Standard CloudWatch Logs charges apply. For more information, see CloudWatch pricing.

Before you begin

- You must have a role that can view logs in CloudWatch. For more information, see Accessing an Amazon MWAA environment.

Log types

Amazon MWAA creates a log group for each Airflow logging option you enable, and pushes the logs to the CloudWatch Logs groups associated with an environment. Log groups are named in the following format: `YourEnvironmentName-`**`LogType`**. For example, if your environment's named `Airflow-v202-Public`, Apache Airflow task logs are sent to `Airflow-v202-Public-`**`Task`**.

Log type	Description
`YourEnvironmentName-` **`DAGProcessing`**	The logs of the DAG processor manager (the part of the scheduler that processes DAG files).
`YourEnvironmentName-` **`Scheduler`**	The logs the Airflow scheduler generates.
`YourEnvironmentName-` **`Task`**	The task logs a DAG generates.
`YourEnvironmentName-` **`WebServer`**	The logs the Airflow web interface generates.
`YourEnvironmentName-` **`Worker`**	The logs generated as part of workflow and DAG execution.

Enabling Apache Airflow logs

You can enable Apache Airflow logs at the `INFO`, `WARNING`, `ERROR`, or `CRITICAL` level. When you choose a log level, Amazon MWAA sends logs for that level and all higher levels of severity. For example, if you enable logs at the `INFO` level, Amazon MWAA sends `INFO` logs and `WARNING`, `ERROR`, and `CRITICAL` log levels to CloudWatch Logs.

1. Open the Environments page on the Amazon MWAA console.
2. Choose an environment.
3. Choose **Edit**.
4. Choose **Next**.

5. Choose one or more of the following logging options:

    a. Choose the **Airflow scheduler log group** on the **Monitoring** pane.

    b. Choose the **Airflow web server log group** on the **Monitoring** pane.

    c. Choose the **Airflow worker log group** on the **Monitoring** pane.

    d. Choose the **Airflow DAG processing log group** on the **Monitoring** pane.

    e. Choose the **Airflow task log group** on the **Monitoring** pane.

    f. Choose the logging level in **Log level**.

6. Choose **Next**.
7. Choose **Save**.

Viewing Apache Airflow logs

The following section describes how to view Apache Airflow logs in the CloudWatch console.

1. Open the Environments page on the Amazon MWAA console.
2. Choose an environment.
3. Choose a log group in the **Monitoring** pane.
4. Choose a log in **Log stream**.

Example scheduler logs

You can view Apache Airflow logs for the *Scheduler* scheduling your workflows and parsing your `dags` folder. The following steps describe how to open the log group for the *Scheduler* on the Amazon MWAA console, and view Apache Airflow logs on the CloudWatch Logs console.

To view logs for a `requirements.txt`

1. Open the Environments page on the Amazon MWAA console.
2. Choose an environment.
3. Choose the **Airflow scheduler log group** on the **Monitoring** pane.
4. Choose the `requirements_install_ip` log in **Log streams**.
5. You should see the list of packages that were installed on the environment at `/usr/local/airflow/.local/bin`. For example:

```
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. Review the list of packages and whether any of these encountered an error during installation. If something went wrong, you may see an error similar to the following:

```
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))
```

What's next?

- Learn how to configure a CloudWatch alarm in Using Amazon CloudWatch alarms.
- Learn how to create a CloudWatch dashboard in Using CloudWatch dashboards.

Monitoring dashboards and alarms on Amazon MWAA

You can create a custom dashboard in Amazon CloudWatch and add alarms for a particular metric to monitor the health status of an Amazon Managed Workflows for Apache Airflow environment. When an alarm is on a dashboard, it turns red when it is in the `ALARM` state, making it easier for you to monitor the health of an Amazon MWAA environment proactively.

Apache Airflow exposes metrics for a number of processes, including the number of DAG processes, DAG bag size, currently running tasks, task failures, and successes. When you create an environment, Airflow is configured to automatically send metrics for an Amazon MWAA environment to CloudWatch. This page describes how to create a health status dashboard for the Airflow metrics in CloudWatch for an Amazon MWAA environment.

Contents

- Metrics
- Alarm states overview

- Example custom dashboards and alarms
  - About these metrics
  - About the dashboard
  - Using AWS tutorials
  - Using AWS CloudFormation
- Deleting metrics and dashboards
- What's next?

Metrics

You can create a custom dashboard and alarm for any of the metrics available for your Apache Airflow version. Each metric corresponds to an Apache Airflow key performance indicator (KPI). To view a list of metrics, see:

- Apache Airflow v2 environment metrics in CloudWatch

Alarm states overview

A metric alarm has the following possible states:

- `OK` – The metric or expression is within the defined threshold.
- `ALARM` – The metric or expression is outside of the defined threshold.
- `INSUFFICIENT_DATA` – The alarm has just started, the metric is not available, or not enough data is available for the metric to determine the alarm state.

Example custom dashboards and alarms

You can build a custom monitoring dashboard that displays charts of selected metrics for your Amazon MWAA environment.

About these metrics

The following list describes each of the metrics created in the custom dashboard by the tutorial and template definitions in this section.

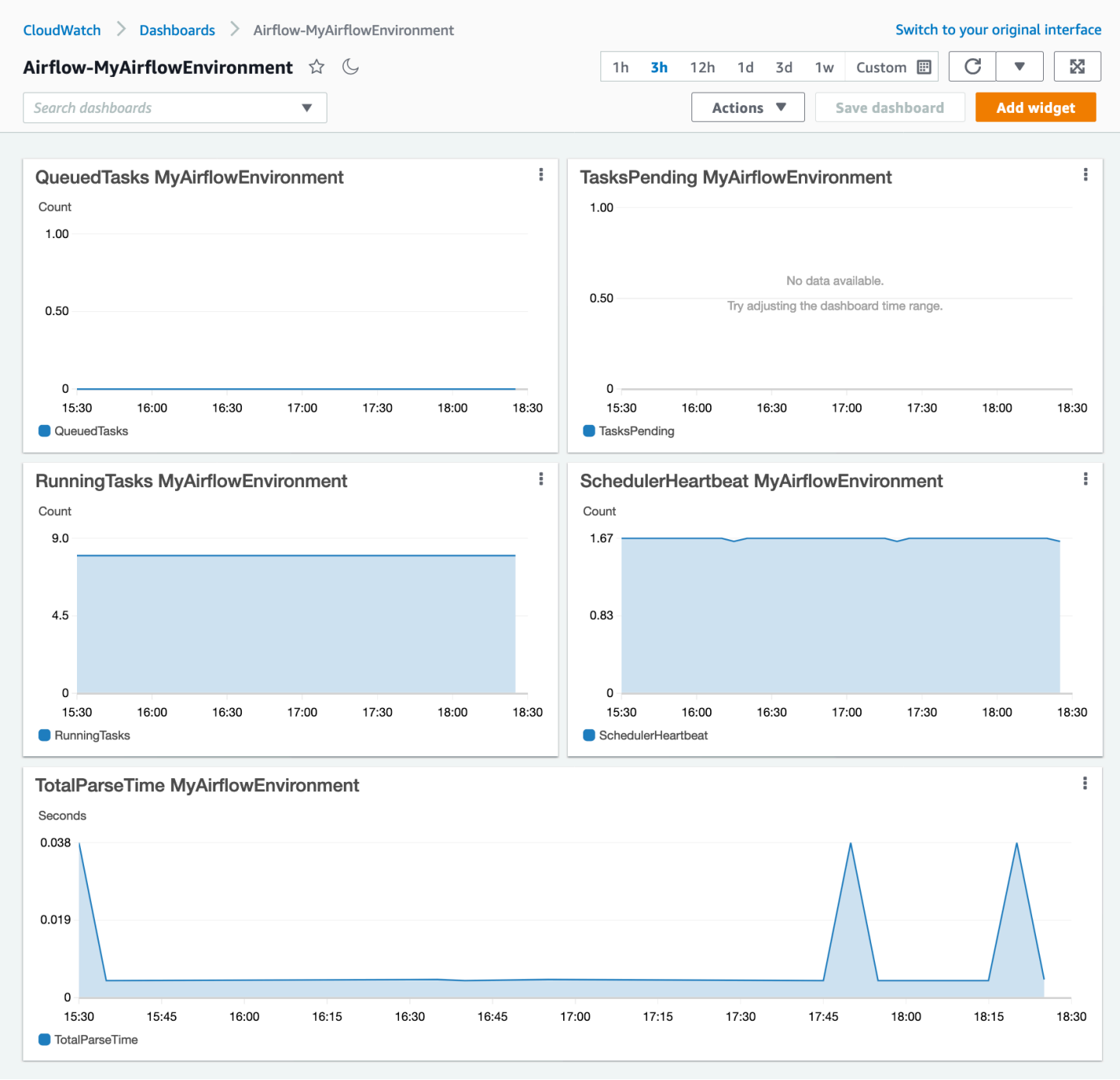
- *QueuedTasks* - The number of tasks with queued state. Corresponds to the `executor.queued_tasks` Apache Airflow metric.
- *TasksPending* - The number of tasks pending in executor. Corresponds to the `scheduler.tasks.pending` Apache Airflow metric.

> **Note**
>
> Does not apply to Apache Airflow v2.2 and above.

- *RunningTasks* - The number of tasks running in executor. Corresponds to the `executor.running_tasks` Apache Airflow metric.
- *SchedulerHeartbeat* - The number of check-ins Apache Airflow performs on the scheduler job. Corresponds to the `scheduler_heartbeat` Apache Airflow metrics.
- *TotalParseTime* - The number of seconds taken to scan and import all DAG files once. Corresponds to the `dag_processing.total_parse_time` Apache Airflow metric.

About the dashboard

The following image shows the monitoring dashboard created by the tutorial and template definition in this section.



Using AWS tutorials

You can use the following AWS tutorial to automatically create a health status dashboard for any Amazon MWAA environments that are currently deployed. It also creates CloudWatch alarms for unhealthy workers and scheduler heartbeat failures across all Amazon MWAA environments.

- CloudWatch Dashboard Automation for Amazon MWAA

Using AWS CloudFormation

You can use the AWS CloudFormation template definition in this section to create a monitoring dashboard in CloudWatch, then add alarms on the CloudWatch console to receive notifications when a metric surpasses a particular threshold. To create the stack using this template definition, see Creating a stack on the AWS CloudFormation console. To add an alarm to the dashboard, see Using alarms.

```
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
                }
            }
        ]
    }'
```

Deleting metrics and dashboards

If you delete an Amazon MWAA environment, the corresponding dashboard is also deleted. CloudWatch metrics are stored for fifteen (15) months and can not be deleted. The CloudWatch console limits the search of metrics to two (2) weeks after a metric is last ingested to ensure that the most up to date instances are shown for your Amazon MWAA environment. To learn more, see Amazon CloudWatch FAQs.

What's next?

- Learn how to create a DAG that queries the Amazon Aurora PostgreSQL metadata database for your environment and publishes custom metrics to CloudWatch in Using a DAG to write custom metrics in CloudWatch.

Apache Airflow v2 environment metrics in CloudWatch

Apache Airflow v2 is already set-up to collect and send StatsD metrics for an Amazon Managed Workflows for Apache Airflow environment to Amazon CloudWatch. The complete list of metrics Apache Airflow sends is available on the Metrics page in the *Apache Airflow reference guide*. This page describes the Apache Airflow metrics available in CloudWatch, and how to access metrics in the CloudWatch console.

Contents

- Terms
- Dimensions
- Accessing metrics in the CloudWatch console
- Apache Airflow metrics available in CloudWatch
  - Apache Airflow Counters
  - Apache Airflow Gauges
  - Apache Airflow Timers
- Choosing which metrics are reported
- What's next?

Terms

Namespace

A namespace is a container for the CloudWatch metrics of an AWS service. For Amazon MWAA, the namespace is *AmazonMWAA*.

CloudWatch metrics

A CloudWatch metric represents a time-ordered set of data points that are specific to CloudWatch.

Apache Airflow metrics

The Metrics specific to Apache Airflow.

Dimension

A dimension is a name/value pair that is part of the identity of a metric.

Unit

A statistic has a unit of measure. For Amazon MWAA, units include *Count*, *Seconds*, and *Milliseconds*. For Amazon MWAA, units are set based on the units in the original Airflow metrics.

Dimensions

This section describes the CloudWatch *Dimensions* grouping for Apache Airflow metrics in CloudWatch.

Dimension	Description
DAG	Indicates a specific Apache Airflow DAG name.
DAG Filename	Indicates a specific Apache Airflow DAG file name.
Function	This dimension is used to improve the grouping of metrics in CloudWatch.
Job	Indicates an Apache Airflow *Job* run by the *Scheduler*. Always has a value of *Job*.
Operator	Indicates a specific Apache Airflow operator.
Pool	Indicates a specific Apache Airflow *worker pool*.
Task	Indicates a specific Apache Airflow task.
HostName	Indicates the hostname for a specific running Apache Airflow process.

Accessing metrics in the CloudWatch console

This section describes how to access performance metrics in CloudWatch for a specific DAG.

To view performance metrics for a dimension

1. Open the Metrics page on the CloudWatch console.
2. Use the AWS Region selector to select your region.
3. Choose the **AmazonMWAA** namespace.
4. In the **All metrics** tab, select a dimension. For example, *DAG, Environment*.
5. Choose a CloudWatch metric for a dimension. For example, *TaskInstanceSuccesses* or *TaskInstanceDuration*. Choose **Graph all search results**.
6. Choose the **Graphed metrics** tab to view performance statistics for Apache Airflow metrics, such as *DAG, Environment, Task*.

Apache Airflow metrics available in CloudWatch

This section describes the Apache Airflow metrics and dimensions sent to CloudWatch.

Apache Airflow Counters

The Apache Airflow metrics in this section contain data about Apache Airflow *Counters*.

CloudWatch metric	Apache Airflow metric	Unit	Dimension	
SLAMissed **Note** Available for Apache Airflow v2.4.3 and above.	sla_missed	Count	Function, Scheduler	
FailedSLACallback	sla_callback_notification_failure	Count	Function, Scheduler	

CloudWatch metric	Apache Airflow metric	Unit	Dimension
Note Available for Apache Airflow v2.4.3 and above.			
Updates **Note** Available for Apache Airflow v2.6.3 and above.	dataset.updates	Count	Function, Scheduler
Orphaned **Note** Available for Apache Airflow v2.6.3 and above.	dataset.orphaned	Count	Function, Scheduler
FailedCeleryTaskExecution **Note** Available for Apache Airflow v2.4.3 and above.	celery.execute_command.failure	Count	Function, Celery

CloudWatch metric	Apache Airflow metric	Unit	Dimension	
FilePathQueueUpdateCount **Note** Available for Apache Airflow v2.6.3 and above.	dag_processing.file_path_queue_update_count	Count	Function, Scheduler	
CriticalSectionBusy	scheduler.critical_section_busy	Count	Function, Scheduler	
DagBagSize	dagbag_size	Count	Function, DAG Processing	
DagCallbackExceptions	dag.callback_exceptions	Count	DAG, All	
FailedSLAEmailAttempts	sla_email_notification_failure	Count	Function, Scheduler	
TaskInstanceFinished	ti.finish.{dag_id}.{task_id}.{state}	Count	DAG, {dag_id} Task, {task_id} State, {state}	

CloudWatch metric	Apache Airflow metric	Unit	Dimension
JobEnd	{job_name}_end	Count	Job, {job_name}
JobHeartbeatFailure	{job_name}_heartbeat_failure	Count	Job, {job_name}
JobStart	{job_name}_start	Count	Job, {job_name}
ManagerStalls	dag_processing.manager_stalls	Count	Function, DAG Processing
OperatorFailures	operator_failures_{operator_name}	Count	Operator, {operator_name}
OperatorSuccesses	operator_successes_{operator_name}	Count	Operator, {operator_name}
OtherCallbackCount **Note** Available in Apache Airflow v2.6.3 and above.	dag_processing.other_callback_count	Count	Function, Scheduler

CloudWatch metric	Apache Airflow metric	Unit	Dimension
Processes	dag_processing.processes	Count	Function, DAG Processing
SchedulerHeartbeat	scheduler_heartbeat	Count	Function, Scheduler
StartedTaskInstances	ti.start.{dag_id}.{task_id}	Count	DAG, All Task, All
SlaCallbackCount	dag_processing.sla_callback_count **Note** Available for Apache Airflow v2.6.3 and above.	Count	Function, Scheduler
TasksKilledExternally	scheduler.tasks.killed_externally	Count	Function, Scheduler

CloudWatch metric	Apache Airflow metric	Unit	Dimension
TaskTimeoutError	celery.task_timeout_error	Count	Function, Celery
TaskInstanceCreatedUsingOperator	task_instance_created-{operator_name}	Count	Operator, {operator_name}
TaskInstancePreviouslySucceeded	previously_succeeded	Count	DAG, All Task, All
TaskInstanceFailures	ti_failures	Count	DAG, All Task, All
TaskInstanceSuccesses	ti_successes	Count	DAG, All Task, All
TaskRemovedFromDAG	task_removed_from_dag.{dag_id}	Count	DAG, {dag_id}
TaskRestoredToDAG	task_restored_to_dag.{dag_id}	Count	DAG, {dag_id}

CloudWatch metric	Apache Airflow metric	Unit	Dimension	
TriggersSucceeded **Note** Available for Apache Airflow v2.7.2 and above.	triggers.succeeded	Count	Function, Trigger	
TriggersFailed **Note** Available for Apache Airflow v2.7.2 and above.	triggers.failed	Count	Function, Trigger	
TriggersBlockedMainThread **Note** Available for Apache Airflow v2.7.2 and above.	triggers.blocked_main_thread	Count	Function, Trigger	
TriggerHeartbeat **Note** Available for Apache Airflow v2.8.1 and above.	triggerer_heartbeat	Count	Function, Triggerer	

CloudWatch metric	Apache Airflow metric	Unit	Dimension
TaskInstanceCreatedUsingOperator	airflow.task_instance_created_{operator_name} **Note** Available for Apache Airflow v2.7.2 and above.	Count	Operator, {operator_name}
ZombiesKilled	zombies_killed	Count	DAG, All Task, All

Apache Airflow Gauges

The Apache Airflow metrics in this section contain data about Apache Airflow *Gauges*.

CloudWatch metric	Apache Airflow metric	Unit	Dimension
DAGFileRefreshError	dag_file_refresh_error	Count	Function, DAG Processing
ImportErrors	dag_processing.import_errors	Count	Function, DAG Processing

CloudWatch metric	Apache Airflow metric	Unit	Dimension
Exception Failures	smart_sensor_operator.exception_failures	Count	Function, Smart Sensor Operator
ExecutedTasks	smart_sensor_operator.executed_tasks	Count	Function, Smart Sensor Operator
InfraFailures	smart_sensor_operator.infra_failures	Count	Function, Smart Sensor Operator
LoadedTasks	smart_sensor_operator.loaded_tasks	Count	Function, Smart Sensor Operator
TotalParseTime	dag_processing.total_parse_time	Seconds	Function, DAG Processing
TriggeredDagRuns **Note** Available in Apache Airflow v2.6.3 and above.	dataset.triggered_dagruns	Count	Function, Scheduler

CloudWatch metric	Apache Airflow metric	Unit	Dimension
TriggersRunning **Note** Available in Apache Airflow v2.7.2 and above.	triggers.running.*{hostname}*	Count	Function, Trigger HostName, *{hostname}*
PoolDeferredSlots **Note** Available in Apache Airflow v2.7.2 and above.	pool.deferred_slots.{pool_name}	Count	Pool, {pool_name}
DAGFileProcessingLastRunSecondsAgo	dag_processing.last_run.seconds_ago.{dag_filename}	Seconds	DAG Filename, {dag_filename}
OpenSlots	executor.open_slots	Count	Function, Executor

CloudWatch metric	Apache Airflow metric	Unit	Dimension
OrphanedTasksAdopted	scheduler.orphaned_tasks.adopted	Count	Function, Scheduler
OrphanedTasksCleared	scheduler.orphaned_tasks.cleared	Count	Function, Scheduler
PokedExceptions	smart_sensor_operator.poked_exception	Count	Function, Smart Sensor Operator
PokedSuccess	smart_sensor_operator.poked_success	Count	Function, Smart Sensor Operator
PokedTasks	smart_sensor_operator.poked_tasks	Count	Function, Smart Sensor Operator
PoolFailures	pool.open_slots.{pool_name}	Count	Pool, {pool_name}
PoolStarvingTasks	pool.starving_tasks.{pool_name}	Count	Pool, {pool_name}
PoolOpenSlots	pool.open_slots.{pool_name}	Count	Pool, {pool_name}
PoolQueuedSlots	pool.queued_slots.{pool_name}	Count	Pool, {pool_name}
PoolRunningSlots	pool.running_slots.{pool_name}	Count	Pool, {pool_name}
ProcessorTimeouts	dag_processing.processor_timeouts	Count	Function, DAG Processing

CloudWatch metric	Apache Airflow metric	Unit	Dimension
QueuedTasks	executor.queued_tasks	Count	Function, Executor
RunningTasks	executor.running_tasks	Count	Function, Executor
TasksExecutable	scheduler.tasks.executable	Count	Function, Scheduler
TasksPending **Note** Does not apply to Apache Airflow v2.2 and above.	scheduler.tasks.pending	Count	Function, Scheduler
TasksRunning	scheduler.tasks.running	Count	Function, Scheduler
TasksStarving	scheduler.tasks.starving	Count	Function, Scheduler
TasksWithoutDagRun	scheduler.tasks.without_dagrun	Count	Function, Scheduler

CloudWatch metric	Apache Airflow metric	Unit	Dimension	
DAGFilePr ocessingL astNumOfD bQueries **Note** Available in Apache Airflow v2.10.1 and above.	dag_processing.las t_num_of_db_queries. {dag_filename}	Count	DAG Filename, {dag_file name}	
PoolSched uledSlots **Note** Available in Apache Airflow v2.10.1 and above.	pool.scheduled_slots. {pool_name}	Count	Pool, {pool_name}	

CloudWatch metric	Apache Airflow metric	Unit	Dimension
TaskCpuUsage **Note** Available in Apache Airflow v2.10.1 and above.	cpu.usage.{dag_id}.{task_id}	Percent	DAG, {dag_id} Task, {task_id}
TaskMemoryUsage **Note** Available in Apache Airflow v2.10.1 and above.	mem.usage.{dag_id}.{task_id}	Percent	DAG, {dag_id} Task, {task_id}

Apache Airflow Timers

The Apache Airflow metrics in this section contain data about Apache Airflow *Timers*.

CloudWatch metric	Apache Airflow metric	Unit	Dimension
CollectDBDags	collect_db_dags	Milliseconds	Function, DAG Processing
CriticalSectionDuration	scheduler.critical_section_duration	Milliseconds	Function, Scheduler
CriticalSectionQueryDuration **Note** Available for Apache Airflow v2.5.1 and above.	scheduler.critical_section_query_duration	Milliseconds	Function, Scheduler
DAGDependencyCheck	dagrun.dependency-check.{dag_id}	Milliseconds	DAG, {dag_id}
DAGDurationFailed	dagrun.duration.failed.{dag_id}	Milliseconds	DAG, {dag_id}
DAGDurationSuccess	dagrun.duration.success.{dag_id}	Milliseconds	DAG, {dag_id}

CloudWatch metric	Apache Airflow metric	Unit	Dimension
DAGFilePr ocessingL astDuration	dag_proce ssing.las t_duration. {dag_filename}	Seconds	DAG Filename, {dag_filename}
DAGSchedu leDelay	dagrun.sc hedule_delay. {dag_id}	Milliseconds	DAG, {dag_id}
FirstTask SchedulingDelay	dagrun.{d ag_id}.fi rst_task_ schedulin g_delay	Milliseconds	DAG, {dag_id}
Scheduler LoopDuration **Note** Available for Apache Airflow v2.5.1 and above.	scheduler .schedule r_loop_duration	Milliseconds	Function, Scheduler
TaskInsta nceDuration	dag.{dag_id}. {task_id}.dura tion	Milliseconds	DAG, {dag_id} Task, {task_id}

CloudWatch metric	Apache Airflow metric	Unit	Dimension
TaskInstanceQueuedDuration	dag.{dag_id}.{ta .queued_duration **Note** Available for Apache Airflow v2.7.2 and above.	Milliseconds	DAG, {dag_id} Task, {task_id}
TaskInstanceScheduledDuration **Note** Available for Apache Airflow v2.7.2 and above.	dag.{dag_id}.{ta .scheduled_duration	Milliseconds	DAG, {dag_id} Task, {task_id}

Choosing which metrics are reported

You can choose which Apache Airflow metrics are emitted to CloudWatch, or blocked by Apache Airflow, using the following Amazon MWAA configuration options:

- **`metrics.metrics_allow_list`** — A list of comma-separated prefixes you can use to select which metrics are emitted to CloudWatch by your environment. Use this option if you want Apache Airflow to not send all available metrics and instead select a subset of elements. For example, `scheduler,executor,dagrun`.
- **`metrics.metrics_block_list`** — A list of comma-separated prefixes to filter out metrics that start with the elements of the list. For example, `scheduler,executor,dagrun`.

If you configure both `metrics.metrics_allow_list` and `metrics.metrics_block_list`, Apache Airflow ignores `metrics.metrics_block_list`. If you configure `metrics.metrics_block_list` but not `metrics.metrics_allow_list`, Apache Airflow filters out the elements you specify in `metrics.metrics_block_list`.

> **Note**
>
> The `metrics.metrics_allow_list` and `metrics.metrics_block_list` configuration options only apply to Apache Airflow v2.6.3 and above. For previous version of Apache Airflow use `metrics.statsd_allow_list` and `metrics.statsd_block_list` instead.

What's next?

- Explore the Amazon MWAA API operation used to publish environment health metrics at PublishMetrics.

Container, queue, and database metrics for Amazon MWAA

In addition to Apache Airflow metrics, you can monitor the underlying components of your Amazon Managed Workflows for Apache Airflow environments using CloudWatch, which collects raw data and processes data into readable, near real-time metrics. With these environment metrics, you will have greater visibility into key performance indicators to help you appropriately size your environments and debug issues with your workflows. These metrics apply to all supported Apache Airflow versions on Amazon MWAA.

Amazon MWAA will provide CPU and memory utilization for each Amazon Elastic Container Service (Amazon ECS) container and Amazon Aurora PostgreSQL instance, and Amazon Simple Queue

Service (Amazon SQS) metrics for the number of messages and the age of the oldest message, Amazon Relational Database Service (Amazon RDS) metrics for database connections, disk queue depth, write operations, latency, and throughput, and Amazon RDS Proxy metrics. These metrics also include the number of base workers, additional workers, schedulers, and web servers.

These statistics are kept for 15 months, so that you can access historical information and gain a better perspective on why a schedule is failing, and troubleshoot underlying issues. You can also set alarms that watch for certain thresholds, and send notifications or take actions when those thresholds are met. For more information, see the Amazon CloudWatch User Guide.

Topics

- Terms
- Dimensions
- Accessing metrics in the CloudWatch console
- List of metrics

Terms

Namespace

A namespace is a container for the CloudWatch metrics of an AWS service. For Amazon MWAA, the namespace is `AWS/MWAA`.

CloudWatch metrics

A CloudWatch metric represents a time-ordered set of data points that are specific to CloudWatch.

Dimension

A dimension is a name/value pair that is part of the identity of a metric.

Unit

A statistic has a unit of measure. For Amazon MWAA, units include *Count*.

Dimensions

This section describes the CloudWatch dimensions grouping for Amazon MWAA metrics in CloudWatch.

Dimension	Description
Cluster	Metrics for the minimum three Amazon ECS container that an Amazon MWAA environme nt uses to run Apache Airflow components: scheduler, worker, and web server.
Queue	Metrics for the Amazon SQS queues that decouple the scheduler from workers. When workers read the messages, they are considere d in-flight and not available for other workers. Messages become available for other workers to read if they are not deleted before the 12 hours visibility timeout.
Database	Metrics the Aurora clusters used by Amazon MWAA. This includes metrics for the primary database instance and a read replica to support the read operations. Amazon MWAA publishes database metrics for both READER and WRITER instances.

Accessing metrics in the CloudWatch console

This section describes how to access your Amazon MWAA metrics in CloudWatch.

To view performance metrics for a dimension

1. Open the Metrics page on the CloudWatch console.
2. Use the AWS Region selector to select your region.
3. Choose the **AWS/MWAA** namespace.
4. In the **All metrics** tab, choose a dimension. For example, **Cluster**.
5. Choose a CloudWatch metric for a dimension. For example, *NumSchedulers* or *CPUUtilization*. Then, choose **Graph all search results**.
6. Choose the **Graphed metrics** tab to view performance metrics.

List of metrics

The following tables list the cluster, queue, and database service metrics for Amazon MWAA. To view descriptions for metrics directly emitted from Amazon ECS, Amazon SQS, or Amazon RDS, choose the respective documentation link.

Topics

- Cluster metrics
- Database metrics
- Queue metrics
- Application Load Balancer metrics

Cluster metrics

The following metrics apply to each scheduler, base worker, additional worker, and web server. For more information and descriptions of each cluster metric, see Available metrics and dimensions in the *Amazon ECS Developer Guide*.

Namespace	Metric	Unit
AWS/MWAA	CPUUtilization	Percent
AWS/MWAA	MemoryUtilization	Percent

Evaluating the number of additional worker and web server containers

You can use the component metrics provided under the **Cluster** dimension, as described in the following procedure, to assess how many additional workers, or web servers, an environment is using at a given point in time. You can do this by graphing either the **CPUUtilization** or the **MemoryUtilization** metric and setting the statistic type to **Sample Count**. The resulting value is the total number of `RUNNING` tasks for the `AdditionalWorker` component. Understanding the number of additional worker instances utilized by your environment can help you gauge how your environment scales and allow you to optimize the number of additional workers.

Workers

To evaluate the number of additional workers using the AWS Management Console

1. Choose the **AWS/MWAA** namespace.
2. In the **All metrics** tab, choose the **Cluster** dimension.
3. Under the **Cluster** dimension, for the **AdditionalWorker**, choose either the **CPUUtilization** or the **MemoryUtilization** metric.
4. On the **Graphed metrics** tab, set **Period** to **1 Minute** and **Statistic** to **Sample Count**.

Web servers

To evaluate the number of additional web servers using the AWS Management Console

1. Choose the **AWS/MWAA** namespace.
2. In the **All metrics** tab, choose the **Cluster** dimension.
3. Under the **Cluster** dimension, for the **AdditionalWebservers**, choose either the **CPUUtilization** or the **MemoryUtilization** metric.
4. On the **Graphed metrics** tab, set **Period** to **1 Minute** and **Statistic** to **Sample Count**.

For more information, see Service RUNNING task count in the *Amazon Elastic Container Service Developer Guide*.

Database metrics

The following metrics apply to each database instance associated with the Amazon MWAA environment.

Namespace	Metric	Unit
AWS/MWAA	CPUUtilization	Percent
AWS/MWAA	DatabaseConnections	Count
AWS/MWAA	DiskQueueDepth	Count
AWS/MWAA	FreeableMemory	Bytes

Namespace	Metric	Unit
AWS/MWAA	VolumeWriteIOPS	Count per five minutes
AWS/MWAA	WriteIOPS	Count per second
AWS/MWAA	WriteLatency	Seconds
AWS/MWAA	WriteThroughput	Bytes per second

Queue metrics

For more information on units and descriptions for the following queue metrics, see Available CloudWatch metrics for Amazon SQS in the *Amazon Simple Queue Service Developer Guide*.

Namespace	Metric	Unit
AWS/MWAA	ApproximateAgeOfOldestTask	Seconds
AWS/MWAA	RunningTasks	Count
AWS/MWAA	QueuedTasks	Count

Application Load Balancer metrics

Application Load Balancer metrics apply to the web servers running in your environment. Amazon MWAA uses these metrics to for scaling your web servers based on the amount of traffic. For more information on units and descriptions for the following load balancer metrics, see CloudWatch metrics for your Application Load Balancer in the *Application Load Balancers User Guide*.

Namespace	Metric	Unit
AWS/MWAA	ActiveConnectionCount	Count

Security in Amazon Managed Workflows for Apache Airflow

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you (the customer). The shared responsibility model describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS Compliance Programs. To learn about the compliance programs that apply to Amazon MWAA, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using Amazon Managed Workflows for Apache Airflow. It shows you how to configure Amazon MWAA to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your Amazon MWAA resources.

In this section:

- Data Protection in Amazon Managed Workflows for Apache Airflow
- AWS Identity and Access Management
- Compliance Validation for Amazon Managed Workflows for Apache Airflow
- Resilience in Amazon Managed Workflows for Apache Airflow
- Infrastructure Security in Amazon MWAA
- Configuration and Vulnerability Analysis in Amazon MWAA
- Security best practices on Amazon MWAA

Data Protection in Amazon Managed Workflows for Apache Airflow

The AWS shared responsibility model applies to data protection in Amazon Managed Workflows for Apache Airflow. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. This content includes the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual user accounts with AWS Identity and Access Management (IAM). That way each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We recommend TLS 1.2 or later.
- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing personal data that is stored in Amazon S3.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form fields such as a **Name** field. This includes when you work with Amazon MWAA or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Encryption on Amazon MWAA

The following topics describe how Amazon MWAA protects your data at rest, and in transit. Use this information to learn how Amazon MWAA integrates with AWS KMS to encrypt data at rest, and how data is encrypted using Transport Layer Security (TLS) protocol in transit.

Topics

- Encryption at rest
- Encryption in transit

Encryption at rest

On Amazon MWAA, data *at rest* is data that the service saves to persistent media.

You can use an AWS owned key for data at rest encryption, or optionally provide a Customer managed key for additional encryption when you create an environment. If you choose to use a customer managed KMS key, it must be in the same account as the other AWS resources and services you are using with your environment.

To use a customer managed KMS key, you must attach the required policy statement for CloudWatch access to your key policy. When you use a customer managed KMS key for your environment, Amazon MWAA attaches four grants on your behalf. For more information on the grants Amazon MWAA attaches to a customer managed KMS key, see Customer managed keys for data encryption.

If you do not specify a customer managed KMS key, by default, Amazon MWAA uses an AWS owned KMS key for to encrypt and decrypt your data. We recommend using an AWS owned KMS key to manage data encryption on Amazon MWAA.

> **Note**
>
> You pay for the storage and use of AWS owned, or customer managed KMS keys on Amazon MWAA. For more information, see AWS KMS Pricing.

Encryption artifacts

You specify the encryption artifacts used for at rest encryption by specifying an AWS owned key or Customer managed key when you create your Amazon MWAA environment. Amazon MWAA adds the grants needed to your specified key.

Amazon S3 – Amazon S3 data is encrypted at the object-level using Server-Side Encryption (SSE). Amazon S3 encryption and decryption takes place on the Amazon S3 bucket where your DAG code and supporting files are stored. Objects are encrypted when they are uploaded to Amazon S3 and decrypted when they are downloaded to your Amazon MWAA environment. By default, if you are

using a customer managed KMS key, Amazon MWAA uses it to read and decrypt the data on your Amazon S3 bucket.

CloudWatch Logs – If you are using an AWS owned KMS key, Apache Airflow logs sent to CloudWatch Logs are encrypted using Server-Side Encryption (SSE) with CloudWatch Logs's AWS owned KMS key. If you are using a customer managed KMS key, you must add a key policy to your KMS key to allow CloudWatch Logs to use your key.

Amazon SQS – Amazon MWAA creates one Amazon SQS queue for your environment. Amazon MWAA handles encrypting data passed to and from the queue using Server-Side Encryption (SSE) with either an AWS owned KMS key, or a customer managed KMS key that you specify. You must add Amazon SQS permissions to your execution role regardless of whether you are using an AWS owned or customer managed KMS key.

Aurora PostgreSQL – Amazon MWAA creates one PostgreSQL cluster for your environment. Aurora PostgreSQL encrypts the content with either an AWS owned or customer managed KMS key using Server-Side Encryption (SSE). If you are using a customer managed KMS key, Amazon RDS adds at least two grants to the key: one for the cluster and one for the database instance. Amazon RDS might create additional grants if you choose to use your customer managed KMS key on multiple environments. For more information, see Data protection in Amazon RDS.

Encryption in transit

Data in transit is referred to as data that may be intercepted as it travels the network.

Transport Layer Security (TLS) encrypts the Amazon MWAA objects in transit between your environment's Apache Airflow components and other AWS services that integrate with Amazon MWAA. such as Amazon S3. For more information about Amazon S3 encryption, see Protecting data using encryption.

Using customer managed keys for encryption

You can optionally provide a Customer managed key for data encryption on your environment. You must create the customer managed KMS key in the same Region as your Amazon MWAA environment instance and your Amazon S3 bucket where you store resources for your workflows. If the customer managed KMS key that you specify is in a different account from the one you use to configure an environment, you must specify the key using its ARN for cross-account access. For more information about creating keys, see Creating Keys in the *AWS Key Management Service Developer Guide*.

What's supported

AWS KMS feature	Supported
An AWS KMS key ID or ARN.	Yes
An AWS KMS key alias.	No
An AWS KMS multi-region key.	No

Using Grants for Encryption

This topic describes the grants Amazon MWAA attaches to a customer managed KMS key on your behalf to encrypt and decrypt your data.

How it works

There are two resource-based access control mechanisms supported by AWS KMS for customer managed KMS key: a key policy and grant.

A key policy is used when the permission is mostly static and used in synchronous service mode. A grant is used when more dynamic and granular permissions are required, such as when a service needs to define different access permissions for itself or other accounts.

Amazon MWAA uses and attaches four grant policies to your customer managed KMS key. This is due to the granular permissions required for an environment to encrypt data at rest from CloudWatch Logs, Amazon SQS queue, Aurora PostgreSQL database database, Secrets Manager secrets, Amazon S3 bucket and DynamoDB tables.

When you create an Amazon MWAA environment and specify a customer managed KMS key, Amazon MWAA attaches the grant policies to your customer managed KMS key. These policies allow Amazon MWAA in `airflow.`*`region`*`}.amazonaws.com` to use your customer managed KMS key to encrypt resources on your behalf that are owned by Amazon MWAA.

Amazon MWAA creates, and attaches, additional grants to a specified KMS key on your behalf. This includes policies to retire a grant if you delete your environment, to use your customer managed KMS key for Client-Side Encryption (CSE), and for the AWS Fargate execution role that needs to access secrets protected by your customer managed key in Secrets Manager.

Grant policies

Amazon MWAA adds the following resource based policy grants on your behalf to a customer managed KMS key. These policies allow the grantee and the principal (Amazon MWAA) to perform actions defined in the policy.

Grant 1: used to create data plane resources

```
{
            "Name": "mwaa-grant-for-env-mgmt-role-environment name",
            "GranteePrincipal": "airflow.region.amazonaws.com",
            "RetiringPrincipal": "airflow.region.amazonaws.com",
            "Operations": [
              "kms:Encrypt",
              "kms:Decrypt",
              "kms:ReEncrypt*",
              "kms:GenerateDataKey*",
              "kms:CreateGrant",
              "kms:DescribeKey",
              "kms:RetireGrant"
            ]
          }
```

Grant 2: used for `ControllerLambdaExecutionRole` access

```
{
            "Name": "mwaa-grant-for-lambda-exec-environment name",
            "GranteePrincipal": "airflow.region.amazonaws.com",
            "RetiringPrincipal": "airflow.region.amazonaws.com",
            "Operations": [
              "kms:Encrypt",
              "kms:Decrypt",
              "kms:ReEncrypt*",
              "kms:GenerateDataKey*",
              "kms:DescribeKey",
              "kms:RetireGrant"
            ]
          }
```

Grant 3: used for `CfnManagementLambdaExecutionRole` access

```
{
              "Name": " mwaa-grant-for-cfn-mgmt-environment name",
              "GranteePrincipal": "airflow.region.amazonaws.com",
              "RetiringPrincipal": "airflow.region.amazonaws.com",
              "Operations": [
                "kms:Encrypt",
                "kms:Decrypt",
                "kms:ReEncrypt*",
                "kms:GenerateDataKey*",
                "kms:DescribeKey"
              ]
            }
```

Grant 4: used for Fargate execution role to access backend secrets

```
{
                "Name": "mwaa-fargate-access-for-environment name",
                "GranteePrincipal": "airflow.region.amazonaws.com",
                "RetiringPrincipal": "airflow.region.amazonaws.com",
                "Operations": [
                  "kms:Encrypt",
                  "kms:Decrypt",
                  "kms:ReEncrypt*",
                  "kms:GenerateDataKey*",
                  "kms:DescribeKey",
                  "kms:RetireGrant"
                ]
              }
```

Attaching key policies to a customer managed key

If you choose to use your own customer managed KMS key with Amazon MWAA, you must attach the following policy to the key to allow Amazon MWAA to use it to encrypt your data.

If the customer managed KMS key you used for your Amazon MWAA environment is not already configured to work with CloudWatch, you must update the key policy to allow for encrypted CloudWatch Logs. For more information, see the Encrypt log data in CloudWatch using AWS Key Management Service service.

The following example represents a key policy for CloudWatch Logs. Substitute the sample values provided for the region.

```
{
          "Effect": "Allow",
          "Principal": {
          "Service": "logs.us-west-2.amazonaws.com"
        },
        "Action": [
          "kms:Encrypt*",
          "kms:Decrypt*",
          "kms:ReEncrypt*",
          "kms:GenerateDataKey*",
          "kms:Describe*"
        ],
        "Resource": "*",
        "Condition": {
          "ArnLike": {
            "kms:EncryptionContext:aws:logs:arn": "arn:aws:logs:us-west-2:*:*"
            }
          }
        }
```

AWS Identity and Access Management

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be authenticated (signed in) and authorized (have permissions) to use Amazon Managed Workflows for Apache Airflow resources. IAM is an AWS service that you can use with no additional charge.

This topic provides a basic overview of how Amazon MWAA uses AWS Identity and Access Management (IAM). To learn about managing access to Amazon MWAA, see Managing access to an Amazon MWAA environment.

Contents

- Audience
- Authenticating With Identities
- Managing Access Using Policies
- Allowing users to view their own permissions

- Troubleshooting Amazon Managed Workflows for Apache Airflow identity and access
- How Amazon MWAA works with IAM

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in Amazon MWAA.

Service user – If you use the Amazon MWAA service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more Amazon MWAA features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in Amazon MWAA, see Troubleshooting Amazon Managed Workflows for Apache Airflow identity and access.

Service administrator – If you're in charge of Amazon MWAA resources at your company, you probably have full access to Amazon MWAA. It's your job to determine which Amazon MWAA features and resources your service users should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with Amazon MWAA, see How Amazon MWAA works with IAM.

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to Amazon MWAA. To view example Amazon MWAA identity-based policies that you can use in IAM, see Amazon MWAA identity-based policy examples.

Authenticating With Identities

Authentication is how you sign in to AWS using your identity credentials. You must be *authenticated* (signed in to AWS) as the AWS account root user, as an IAM user, or by assuming an IAM role.

You can sign in to AWS as a federated identity by using credentials provided through an identity source. AWS IAM Identity Center (IAM Identity Center) users, your company's single sign-on authentication, and your Google or Facebook credentials are examples of federated identities. When you sign in as a federated identity, your administrator previously set up identity federation using IAM roles. When you access AWS by using federation, you are indirectly assuming a role.

Depending on the type of user you are, you can sign in to the AWS Management Console or the AWS access portal. For more information about signing in to AWS, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

If you access AWS programmatically, AWS provides a software development kit (SDK) and a command line interface (CLI) to cryptographically sign your requests by using your credentials. If you don't use AWS tools, you must sign requests yourself. For more information about using the recommended method to sign requests yourself, see AWS Signature Version 4 for API requests in the *IAM User Guide*.

Regardless of the authentication method that you use, you might be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Multi-factor authentication in the *AWS IAM Identity Center User Guide* and AWS Multi-factor authentication in IAM in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see Tasks that require root user credentials in the *IAM User Guide*.

IAM Users and Groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see Rotate access keys regularly for use cases that require long-term credentials in the *IAM User Guide*.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see Use cases for IAM users in the *IAM User Guide*.

IAM Roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. To temporarily assume an IAM role in the AWS Management Console, you can switch from a user to an IAM role (console). You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Methods to assume a role in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see Create a role for a third-party identity provider (federation) in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see Permission sets in the *AWS IAM Identity Center User Guide*.
- **Temporary IAM user permissions** – An IAM user or role can assume an IAM role to temporarily take on different permissions for a specific task.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
  - **Forward access sessions (FAS)** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the

principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.

- **Service role** – A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Create a role to delegate permissions to an AWS service in the *IAM User Guide*.
- **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Use an IAM role to grant permissions to applications running on Amazon EC2 instances in the *IAM User Guide*.

Managing Access Using Policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when a principal (user, root user, or role session) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-Based Policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choose between managed policies and inline policies in the *IAM User Guide*.

Resource-Based Policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Access Control Lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the *Amazon Simple Storage Service Developer Guide*.

Other Policy Types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of an entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see Service control policies in the *AWS Organizations User Guide*.
- **Resource control policies (RCPs)** – RCPs are JSON policies that you can use to set the maximum available permissions for resources in your accounts without updating the IAM policies attached to each resource that you own. The RCP limits permissions for resources in member accounts and can impact the effective permissions for identities, including the AWS account root user, regardless of whether they belong to your organization. For more information about Organizations and RCPs, including a list of AWS services that support RCPs, see Resource control policies (RCPs) in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session policies in the *IAM User Guide*.

Multiple Policy Types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

Allowing users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
```

```
    ]
}
```

Troubleshooting Amazon Managed Workflows for Apache Airflow identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with Amazon MWAA and IAM.

I am not authorized to perform an action in Amazon MWAA

If the AWS Management Console tells you that you're not authorized to perform an action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to Amazon MWAA.

Some AWS services allow you to pass an existing role to that service instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in Amazon MWAA. However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to allow people outside of my AWS account to access my Amazon MWAA resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether Amazon MWAA supports these features, see How Amazon MWAA works with IAM.
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.

How Amazon MWAA works with IAM

Amazon MWAA uses IAM identity-based policies to grant permissions to Amazon MWAA actions and resources. For recommended examples of custom IAM policies you can use to control access to your Amazon MWAA resources, see the section called “Accessing an Amazon MWAA environment”.

To get a high-level view of how Amazon MWAA and other AWS services work with IAM, see AWS Services That Work with IAM in the *IAM User Guide*.

Amazon MWAA identity-based policies

With IAM identity-based policies, you can specify allowed or denied actions and resources, as well as the conditions under which actions are allowed or denied. Amazon MWAA supports specific actions, resources, and condition keys.

The following steps show how you can create a new JSON policy using the IAM console. This policy provides read-only access to your Amazon MWAA resources.

To use the JSON policy editor to create a policy

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane on the left, choose **Policies**.

   If this is your first time choosing **Policies**, the **Welcome to Managed Policies** page appears. Choose **Get Started**.
3. At the top of the page, choose **Create policy**.
4. In the **Policy editor** section, choose the **JSON** option.
5. Enter the following JSON policy document:

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

6. Choose **Next**.

> **Note**
>
> You can switch between the **Visual** and **JSON** editor options anytime. However, if you make changes or choose **Next** in the **Visual** editor, IAM might restructure your policy to optimize it for the visual editor. For more information, see Policy restructuring in the *IAM User Guide*.

7. On the **Review and create** page, enter a **Policy name** and a **Description** (optional) for the policy that you are creating. Review **Permissions defined in this policy** to see the permissions that are granted by your policy.

8. Choose **Create policy** to save your new policy.

To learn about all of the elements that you use in a JSON policy, see IAM JSON Policy Elements Reference in the *IAM User Guide*.

Actions

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

Policy statements must include either an `Action` element or a `NotAction` element. The `Action` element lists the actions allowed by the policy. The `NotAction` element lists the actions that are not allowed.

The actions defined for Amazon MWAA reflect tasks that you can perform using Amazon MWAA. Policy actions in Detective have the following prefix: `airflow:`.

You can also use wildcards (*) to specify multiple actions. Instead of listing these actions separately, you can grant access to all actions that end with the word, for example, `environment`.

To see a list of Amazon MWAA actions, see Actions Defined by Amazon Managed Workflows for Apache Airflow in the *IAM User Guide*.

Amazon MWAA identity-based policy examples

To view the Amazon MWAA policies, see Managing access to an Amazon MWAA environment.

By default, IAM users and roles don't have permission to create or modify Amazon MWAA resources. They also can't perform tasks using the AWS Management Console, AWS CLI, or AWS API.

An IAM administrator must create IAM policies that grant users and roles permission to perform specific API operations on the specified resources they need. The administrator then attaches those policies to the IAM users or groups that require those permissions.

> **Important**
>
> We recommend using IAM roles and temporary credentials to provide access to your Amazon MWAA resources. Avoiding attaching permission poicies directly to your IAM users.

To learn how to create an IAM identity-based policy using these example JSON policy documents, see Creating Policies on the JSON Tab in the *IAM User Guide*.

Topics

- Policy best practices
- Using the Amazon MWAA console
- Allowing users to view their own permissions

Policy best practices

Identity-based policies determine whether someone can create, access, or delete Amazon MWAA resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as AWS CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.

- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see Validate policies with IAM Access Analyzer in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see Secure API access with MFA in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM in the *IAM User Guide*.

Using the Amazon MWAA console

To use the Amazon MWAA console, the user or role must have access to the relevant actions, which match corresponding actions in the API.

To view the Amazon MWAA policies, see Managing access to an Amazon MWAA environment.

Allowing users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
```

```
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Compliance Validation for Amazon Managed Workflows for Apache Airflow

To learn whether an AWS service is within the scope of specific compliance programs, see AWS services in Scope by Compliance Program and choose the compliance program that you are interested in. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security Compliance & Governance – These solution implementation guides discuss architectural considerations and provide steps for deploying security and compliance features.
- HIPAA Eligible Services Reference – Lists HIPAA eligible services. Not all AWS services are HIPAA eligible.
- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.

- AWS Customer Compliance Guides – Understand the shared responsibility model through the lens of compliance. The guides summarize the best practices for securing AWS services and map the guidance to security controls across multiple frameworks (including National Institute of Standards and Technology (NIST), Payment Card Industry Security Standards Council (PCI), and International Organization for Standardization (ISO)).
- Evaluating Resources with Rules in the *AWS Config Developer Guide* – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS. Security Hub uses security controls to evaluate your AWS resources and to check your compliance against security industry standards and best practices. For a list of supported services and controls, see Security Hub controls reference.
- Amazon GuardDuty – This AWS service detects potential threats to your AWS accounts, workloads, containers, and data by monitoring your environment for suspicious and malicious activities. GuardDuty can help you address various compliance requirements, like PCI DSS, by meeting intrusion detection requirements mandated by certain compliance frameworks.
- AWS Audit Manager – This AWS service helps you continuously audit your AWS usage to simplify how you manage risk and compliance with regulations and industry standards.

Resilience in Amazon Managed Workflows for Apache Airflow

The AWS global infrastructure is built around AWS Regions and Availability Zones. Regions provide multiple physically separated and isolated Availability Zones, which are connected through low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

Infrastructure Security in Amazon MWAA

As a managed service, Amazon Managed Workflows for Apache Airflow is protected by AWS global network security. For information about AWS security services and how AWS protects infrastructure, see AWS Cloud Security. To design your AWS environment using the best practices

for infrastructure security, see Infrastructure Protection in *Security Pillar AWS Well-Architected Framework*.

You use AWS published API calls to access Amazon MWAA through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

Configuration and Vulnerability Analysis in Amazon MWAA

Configuration and IT controls are a shared responsibility between AWS and you, our customer.

Amazon Managed Workflows for Apache Airflow periodically patches and upgrades Apache Airflow on your environments. You should ensure that the appropriate access policies are used for your VPCs.

For more details, see the following resources:

- Compliance Validation for Amazon Managed Workflows for Apache Airflow
- Shared Responsibility Model
- Amazon Web Services: Overview of Security Processes
- Infrastructure Security in Amazon MWAA
- Security best practices on Amazon MWAA

Security best practices on Amazon MWAA

Amazon MWAA provides a number of security features to consider as you develop and implement your own security policies. The following best practices are general guidelines and don't represent a complete security solution. Because these best practices might not be appropriate or sufficient for your environment, treat them as helpful considerations rather than prescriptions.

- Use least-permissive permission policies. Grant permissions to only the resources or actions that users need to perform tasks.
- Use AWS CloudTrail to monitor user activity in your account.
- Ensure that the Amazon S3 bucket policy and object ACLs grant permissions to the users from the associated Amazon MWAA environment to put objects into the bucket. This ensures that users with permissions to add workflows to the bucket also have permissions to run the workflows in Airflow.
- Use the Amazon S3 buckets associated with Amazon MWAA environments. Your Amazon S3 bucket can be any name. Do not store other objects in the bucket, or use the bucket with another service.

Security best practices in Apache Airflow

Apache Airflow is not multi-tenant. While there are access control measures to limit some features to specific users, which Amazon MWAA implements, DAG creators do have the ability to write DAGs that can change Apache Airflow user privileges and interact with the underlying metadatabase.

We recommend the following steps when working with Apache Airflow on Amazon MWAA to ensure your environment's metadatabase and DAGs are secure.

- Use separate environments for separate teams with DAG writing access, or the ability to add files to your Amazon S3 `/dags` folder, assuming anything accessible by the Amazon MWAA Execution Role or Apache Airflow connections will also be accessible to users who can write to the environment.
- Do not provide direct Amazon S3 DAGs folder access. Instead, use CI/CD tools to write DAGs to Amazon S3, with a validation step ensuring that the DAG code meets your team's security guidelines.
- Prevent user access to your environment's Amazon S3 bucket. Instead, use a DAG factory that generates DAGs based on a YAML, JSON, or other definition file stored in a separate location from your Amazon MWAA Amazon S3 bucket where you store DAGs.
- Store secrets in Secrets Manager. While this will not prevent users who can write DAGs from reading secrets, it will prevent them from modifying the secrets that your environment uses.

Detecting changes to Apache Airflow user privileges

You can use CloudWatch Logs Insights to detect occurences of DAGs changing Apache Airflow user privileges. To do so, you can use an EventBridge scheduled rule, a Lambda function, and CloudWatch Logs Insights to deliver notifications to CloudWatch metrics whenever one of your DAGs changes Apache Airflow user privileges.

Prerequisites

To complete the following steps, you will need the following:

- An Amazon MWAA environment with all Apache Airflow log types enabled at the `INFO` log level. For more information, see the section called “Viewing Airflow logs”.

To configure notifications for changes to Apache Airflow user privileges

1. Create a Lambda function that runs the following CloudWatch Logs Insights query string against the five Amazon MWAA environment log groups (`DAGProcessing`, `Scheduler`, `Task`, `WebServer`, and `Worker`).

   ```
   fields @log, @timestamp, @message | filter @message like "add-role" | stats count()
    by @log
   ```

2. Create an EventBridge rule that runs on a schedule, with the Lambda function you created in the previous step as the rule's target. Configure your schedule using a cron or rate expression to run at regular intervals.

Apache Airflow versions on Amazon Managed Workflows for Apache Airflow

This topic describes the Apache Airflow versions Amazon Managed Workflows for Apache Airflow supports, and best-practices for upgrading to the latest version.

Topics

- About Amazon MWAA versions
- Latest version
- Apache Airflow versions
- Apache Airflow components
- Upgrading the Apache Airflow version
- Apache Airflow deprecated versions
- Apache Airflow version support and FAQ

About Amazon MWAA versions

Amazon MWAA builds container images that bundle Apache Airflow releases with other common binaries and Python libraries. The image uses the Apache Airflow base install for the version you specify. When you create an environment, you specify an image version to use. Once an environment is created, it keeps using the specified image version until you upgrade it to a later version.

Latest version

Amazon MWAA supports more than one Apache Airflow version. If you do not specify an image version when you create an environment, Amazon MWAA creates an environment using the latest supported version of Apache Airflow.

Apache Airflow versions

The following Apache Airflow versions are supported on Amazon Managed Workflows for Apache Airflow.

Note

- Beginning with Apache Airflow v2.2.2, Amazon MWAA supports installing Python requirements, provider packages, and custom plugins directly on the Apache Airflow web server.
- Beginning with Apache Airflow v2.7.2, your requirements file must include a `--constraint` statement. If you do not provide a constraint, Amazon MWAA will specify one for you to ensure the packages listed in your requirements are compatible with the version of Apache Airflow you are using.

  For more information on setting up constraints in your requirements file, see Installing Python dependencies.

Apache Airflow version	Apache Airflow guide	Apache Airflow constraints	Python version
2.10.3;	Apache Airflow v2.10.3 reference guide	Apache Airflow v2.10.3 constraints file	Python 3.11
2.10.1;	Apache Airflow v2.10.1 reference guide	Apache Airflow v2.10.1 constraints file	Python 3.11
v2.9.2	Apache Airflow v2.9.2 reference guide	Apache Airflow v2.9.2 constraints file	Python 3.11
v2.8.1	Apache Airflow v2.8.1 reference guide	Apache Airflow v2.8.1 constraints file	Python 3.11
v2.7.2	Apache Airflow v2.7.2 reference guide	Apache Airflow v2.7.2 constraints file	Python 3.11
v2.6.3	Apache Airflow v2.6.3 reference guide	Apache Airflow v2.6.3 constraints file	Python 3.10

Apache Airflow version	Apache Airflow guide	Apache Airflow constraints	Python version
v2.5.1	Apache Airflow v2.5.1 reference guide	Apache Airflow v2.5.1 constraints file	Python 3.10
v2.4.3	Apache Airflow v2.4.3 reference guide	Apache Airflow v2.4.3 constraints file	Python 3.10
v2.2.2	Apache Airflow v2.2.2 reference guide	Apache Airflow v2.2.2 constraints file	Python 3.7
v2.0.2	Apache Airflow v2.0.2 reference guide	Apache Airflow v2.0.2 constraints file	Python 3.7

For more information about migrating your self-managed Apache Airflow deployments, or migrating an existing Amazon MWAA environment, including instructions for backing up your metadata database, see the Amazon MWAA Migration Guide.

Apache Airflow components

This section describes the number of Apache Airflow schedulers and workers available for each Apache Airflow version on Amazon MWAA, and provides a list of key Apache Airflow features, indicating the version that supports each feature.

Schedulers

Apache Airflow version	Scheduler (default)	Scheduler (min)	Scheduler (max)	
Apache Airflow v2 and above	2	2	5	

Workers

Airflow version	Workers (min)	Workers (max)	Workers (default)	
Apache Airflow v2	1	25	10	

Upgrading the Apache Airflow version

Amazon MWAA supports minor version upgrades. This means you can upgrade your environment from version x**.1.**z to x**.2.**z, but no to a new major version, for example, from **1.**y**.**z to **2.**y**.**z.

> **Note**
>
> You cannot downgrade the Apache Airflow version for your environment.

For more information, and detailed instructions on updating your workflow resources, and upgrading the environment to a new version, see the section called “Upgrading the version”.

Apache Airflow deprecated versions

The following table lists the deprecated versions of Apache Airflow in Amazon MWAA, along with initial release and end of support dates for each version. For more information about migrating to a newer version, see the Amazon MWAA Migration Guide.

Apache Airflow version	Apache Airflow release date	Amazon MWAA availability date	Amazon MWAA limited support date	Amazon MWAA end of support date
v1.10.12	August 25, 2020	November 24, 2020	August 21, 2023	February 21, 2024
v2.0.2	April 19, 2021	May 25, 2021	November 23, 2023	April 29, 2024

Apache Airflow version	Apache Airflow release date	Amazon MWAA availability date	Amazon MWAA limited support date	Amazon MWAA end of support date
v2.2.2	November 15, 2021	January 27,2022	January 25, 2024	June 27, 2024

Apache Airflow version support and FAQ

In accordance with the Apache Airflow community release process and version policy, Amazon MWAA is committed to supporting at least three minor versions of Apache Airflow at any given time. We will announce the end of support date of a given Apache Airflow minor version at least 90 days before the end of support date.

Frequently asked questions

Q: How long does Amazon MWAA support an Apache Airflow version?

A: Amazon MWAA supports an Apache Airflow minor version for a minimum of 12 months after first being available.

Q: Am I notified when support is ending for an Apache Airflow version on Amazon MWAA?

A: Yes. If any Amazon MWAA environments in your account run the version nearing the end of support, Amazon MWAA sends out a notice through the AWS Health Dashboard with the end of support date.

Q: What happens on the limited support date?

A: On the limited support date, you can no longer create new Amazon MWAA environments with the associated version. Your existing environments will continue to be available until the end of support date.

Q: What happens on the end of support date?

A: On the end of support date, you will continue to be able to access your existing Amazon MWAA environments that run the associated, deprecated version of Apache Airflow at your own risk. For instructions on upgrading to a newer version of Apache Airflow on Amazon MWAA, see the Amazon MWAA Migration Guide.

Important

You are responsible for keeping your Amazon MWAA versions current. AWS urges all customers to upgrade their Amazon MWAA environments to the latest version in order to benefit from the most current security, privacy, and availability safeguards. If you operate your environment on an unsupported version or software past the deprecation date, referred to as the *legacy version*, you face a greater likelihood of security, privacy, and operational risks, including downtime events. By operating your Amazon MWAA environment on a legacy version, you confirm that you understand and knowingly assume these risks, and you agree to complete your upgrade to the latest version as soon as possible. Continued operation of your environment on a legacy version is subject to the agreement governing your use of the AWS services.

Legacy versions are not considered generally available, and AWS no longer provides support for the legacy version. As a result, AWS may place limits on the access to or use of any legacy version at any time, if AWS determines that the legacy version poses a security or liability risk, or a risk of harm, to the services, AWS, its Affiliates, or any other third party. Your decision to continue running Your workloads on a legacy version might result in Your content becoming unavailable, corrupted, or unrecoverable. Environments running on a legacy version are subject to Service Level Agreement (SLA) exceptions.

Environments, and related software, running on a legacy version might contain bugs, errors, defects, and harmful components. Accordingly, and notwithstanding any information to the contrary in the agreement, or the terms of service, AWS provides the legacy version *as is*.

For more information about AWS's shared responsibility model, see Shared responsibility in the *AWS Well-Architected Framework*.

Amazon Managed Workflows for Apache Airflow service endpoints and quotas

Amazon Managed Workflows for Apache Airflow has the following service quotas and endpoints. Service quotas, also referred to as limits, are the maximum number of service resources or operations for your AWS account.

Contents

- Service endpoints
- Service quotas
- Increasing quotas

Service endpoints

To view a list of endpoints for Amazon MWAA, see Amazon Managed Workflows for Apache Airflow endpoints and quotas.

Service quotas

Quota name	Description	Default quota	Adjustable
Environments	The maximum number of Amazon MWAA environments per account per Region.	10	Yes
Workers per environment	The maximum number of workers per Amazon MWAA environment.	25	Yes
Web servers per environment	The maximum number of web	5	Yes

Quota name	Description	Default quota	Adjustable
	servers per Amazon MWAA environment.		

Increasing quotas

You can request an increase to an adjustable quota by submitting a quota increase request.

Amazon MWAA frequently asked questions

This page describes common questions you may encounter when using Amazon Managed Workflows for Apache Airflow.

Contents

- Supported versions
  - What does Amazon MWAA support for Apache Airflow v2?
  - Why are older versions of Apache Airflow not supported?
  - What Python version should I use?
  - What version of pip does Amazon MWAA use?
- Use cases
  - When should I use AWS Step Functions vs. Amazon MWAA?
- Environment specifications
  - How much task storage is available to each environment?
  - What is the default operating system used for Amazon MWAA environments?
  - Can I use a custom image for my Amazon MWAA environment?
  - Is Amazon MWAA HIPAA compliant?
  - Does Amazon MWAA support Spot Instances?
  - Does Amazon MWAA support a custom domain?
  - Can I SSH into my environment?
  - Why is a self-referencing rule required on the VPC security group?
  - Can I hide environments from different groups in IAM?
  - Can I store temporary data on the Apache Airflow Worker?
  - Can I specify more than 25 Apache Airflow Workers?
  - Does Amazon MWAA support shared Amazon VPCs or shared subnets?
  - Can I create or integrate custom Amazon SQS queues to manage task execution and workflow orchestration in Apache Airflow?
- Metrics
  - What metrics are used to determine whether to scale Workers?
  - Can I create custom metrics in CloudWatch?

- DAGs, Operators, Connections, and other questions
  - Can I use the PythonVirtualenvOperator?
  - How long does it take Amazon MWAA to recognize a new DAG file?
  - Why is my DAG file not picked up by Apache Airflow?
  - Can I remove a plugins.zip or requirements.txt from an environment?
  - Why don't I see my plugins in the Apache Airflow v2.0.2 Admin Plugins menu?
  - Can I use AWS Database Migration Service (DMS) Operators?
  - When I access the Airflow REST API using the AWS credentials, can I increase the throttling limit to more than 10 transactions per second (TPS)?

Supported versions

What does Amazon MWAA support for Apache Airflow v2?

To learn what Amazon MWAA supports, see Apache Airflow versions on Amazon Managed Workflows for Apache Airflow.

Why are older versions of Apache Airflow not supported?

We are only supporting the latest (as of launch) Apache Airflow version Apache Airflow v1.10.12 due to security concerns with older versions.

What Python version should I use?

The following Apache Airflow versions are supported on Amazon Managed Workflows for Apache Airflow.

Note

- Beginning with Apache Airflow v2.2.2, Amazon MWAA supports installing Python requirements, provider packages, and custom plugins directly on the Apache Airflow web server.
- Beginning with Apache Airflow v2.7.2, your requirements file must include a `--constraint` statement. If you do not provide a constraint, Amazon MWAA will specify one for you to ensure the packages listed in your requirements are compatible with the version of Apache Airflow you are using.

For more information on setting up constraints in your requirements file, see Installing Python dependencies.

Apache Airflow version	Apache Airflow guide	Apache Airflow constraints	Python version
2.10.3;	Apache Airflow v2.10.3 reference guide	Apache Airflow v2.10.3 constraints file	Python 3.11
2.10.1;	Apache Airflow v2.10.1 reference guide	Apache Airflow v2.10.1 constraints file	Python 3.11
v2.9.2	Apache Airflow v2.9.2 reference guide	Apache Airflow v2.9.2 constraints file	Python 3.11
v2.8.1	Apache Airflow v2.8.1 reference guide	Apache Airflow v2.8.1 constraints file	Python 3.11
v2.7.2	Apache Airflow v2.7.2 reference guide	Apache Airflow v2.7.2 constraints file	Python 3.11
v2.6.3	Apache Airflow v2.6.3 reference guide	Apache Airflow v2.6.3 constraints file	Python 3.10
v2.5.1	Apache Airflow v2.5.1 reference guide	Apache Airflow v2.5.1 constraints file	Python 3.10
v2.4.3	Apache Airflow v2.4.3 reference guide	Apache Airflow v2.4.3 constraints file	Python 3.10
v2.2.2	Apache Airflow v2.2.2 reference guide	Apache Airflow v2.2.2 constraints file	Python 3.7
v2.0.2	Apache Airflow v2.0.2 reference guide	Apache Airflow v2.0.2 constraints file	Python 3.7

For more information about migrating your self-managed Apache Airflow deployments, or migrating an existing Amazon MWAA environment, including instructions for backing up your metadata database, see the Amazon MWAA Migration Guide.

What version of `pip` does Amazon MWAA use?

For environments running Apache Airflow v1.10.12, Amazon MWAA installs `pip` version 21.1.2.

> **Note**
>
> Amazon MWAA will not upgrade `pip` for Apache Airflow v1.10.12 environments.

For environments running Apache Airflow v2 and above, Amazon MWAA installs `pip` version 21.3.1.

Use cases

When should I use AWS Step Functions vs. Amazon MWAA?

1. You can use Step Functions to process individual customer orders, since Step Functions can scale to meet demand for one order or one million orders.
2. If you're running an overnight workflow that processes the previous day's orders, you can use Step Functions or Amazon MWAA. Amazon MWAA allows you an open source option to abstract the workflow from the AWS resources you're using.

Environment specifications

How much task storage is available to each environment?

The task storage is limited to 20 GB, and is specified by Amazon ECS Fargate 1.4. The amount of RAM is determined by the environment class you specify. For more information about environment classes, see Configuring the Amazon MWAA environment class.

What is the default operating system used for Amazon MWAA environments?

Amazon MWAA environments are created on instances running Amazon Linux 2 for versions 2.6 and older, and on instances running Amazon Linux 2023 for versions 2.7 and newer.

Can I use a custom image for my Amazon MWAA environment?

Custom images are not supported. Amazon MWAA uses images that are built on Amazon Linux AMI. Amazon MWAA installs the additional requirements by running `pip3 -r install` for the requirements specified in the requirements.txt file you add to the Amazon S3 bucket for the environment.

Is Amazon MWAA HIPAA compliant?

Amazon MWAA is Health Insurance Portability and Accountability Act (HIPAA) eligible. If you have a HIPAA Business Associate Addendum (BAA) in place with AWS, you can use Amazon MWAA for workflows handling Protected Health Information (PHI) on environments created on, or after, November 14th, 2022.

Does Amazon MWAA support Spot Instances?

Amazon MWAA does not currently support on-demand Amazon EC2 Spot Instance types for Apache Airflow. However, an Amazon MWAA environment can trigger Spot Instances on, for example, Amazon EMR and Amazon EC2.

Does Amazon MWAA support a custom domain?

To be able to use a custom domain for your Amazon MWAA hostname, do one of the following:

- For Amazon MWAA deployments with public web server access, you can use Amazon CloudFront with Lambda@Edge to direct traffic to your environment, and map a custom domain name to CloudFront. For more information and an example of setting up a custom domain for a public environment, see the Amazon MWAA custom domain for public web server sample in the Amazon MWAA examples GitHub repository.
- For Amazon MWAA deployments with private web server access, see the section called “Setting up a custom domain”.

Can I SSH into my environment?

While SSH is not supported on a Amazon MWAA environment, it's possible to use a DAG to run bash commands using the `BashOperator`. For example:

```
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'] }}"
      )
```

To trigger the DAG in the Apache Airflow UI, use:

```
{ "command" : "your bash command"}
```

Why is a self-referencing rule required on the VPC security group?

By creating a self-referencing rule, you're restricting the source to the same security group in the VPC, and it's not open to all networks. To learn more, see the section called “Security in your VPC”.

Can I hide environments from different groups in IAM?

You can limit access by specifying an environment name in AWS Identity and Access Management, however, visibility filtering isn't available in the AWS console—if a user can see one environment, they can see all environments.

Can I store temporary data on the Apache Airflow Worker?

Your Apache Airflow Operators can store temporary data on the *Workers*. Apache Airflow *Workers* can access temporary files in the `/tmp` on the Fargate containers for your environment.

> **Note**
>
> Total task storage is limited to 20 GB, according to Amazon ECS Fargate 1.4. There's no guarantee that subsequent tasks will run on the same Fargate container instance, which might use a different `/tmp` folder.

Can I specify more than 25 Apache Airflow Workers?

Yes. Although you can specify up to 25 Apache Airflow workers on the Amazon MWAA console, you can configure up to 50 on an environment by requesting a quota increase. For more information, see Requesting a quota increase.

Does Amazon MWAA support shared Amazon VPCs or shared subnets?

Amazon MWAA does not support shared Amazon VPCs or shared subnets. The Amazon VPC you select when you create an environment should be owned by the account that is attempting to create the environment. However, you can route traffic from an Amazon VPC in the Amazon MWAA account to a shared VPC. For more information, and to see an example of routing traffic to a shared Amazon VPC, see Centralized outbound routing to the internet in the *Amazon VPC Transit Gateways Guide*.

Can I create or integrate custom Amazon SQS queues to manage task execution and workflow orchestration in Apache Airflow?

No, you cannot create, modify, or use custom Amazon SQS queues within Amazon MWAA. This is because Amazon MWAA automatically provisions and manages its own Amazon SQS queue for each Amazon MWAA environment.

Metrics

What metrics are used to determine whether to scale Workers?

Amazon MWAA monitors the **QueuedTasks** and **RunningTasks** in CloudWatch to determine whether to scale Apache Airflow *Workers* on your environment. To learn more, see *Monitoring and metrics*.

Can I create custom metrics in CloudWatch?

Not on the CloudWatch console. However, you can create a DAG that writes custom metrics in CloudWatch. For more information, see the section called “Using a DAG to write custom metrics”.

DAGs, Operators, Connections, and other questions

Can I use the `PythonVirtualenvOperator`?

The `PythonVirtualenvOperator` is not explicitly supported on Amazon MWAA, but you can create a custom plugin that uses the `PythonVirtualenvOperator`. For sample code, see the section called "Custom plugin to patch PythonVirtualenvOperator ".

How long does it take Amazon MWAA to recognize a new DAG file?

DAGs are periodically synchronized from the Amazon S3 bucket to your environment. If you add a new DAG file, it takes about 300 seconds for Amazon MWAA to start *using* the new file. If you update an existing DAG, it takes Amazon MWAA about 30 seconds to recognize your updates.

These values, 300 seconds for new DAGs, and 30 seconds for updates to existing DAGs, correspond to Apache Airflow configuration options `dag_dir_list_interval`, and `min_file_process_interval` respectively.

Why is my DAG file not picked up by Apache Airflow?

The following are possible solutions for this issue:

1. Check that your execution role has sufficient permissions to your Amazon S3 bucket. To learn more, see Amazon MWAA execution role.
2. Check that the Amazon S3 bucket has *Block Public Access* configured, and *Versioning* enabled. To learn more, see Create an Amazon S3 bucket for Amazon MWAA.
3. Verify the DAG file itself. For example, be sure that each DAG has a unique DAG ID.

Can I remove a `plugins.zip` or `requirements.txt` from an environment?

Currently, there is no way to remove a plugins.zip or requirements.txt from an environment once they've been added, but we're working on the issue. In the interim, a workaround is to point to an empty text or zip file, respectively. To learn more, see Deleting files on Amazon S3.

Why don't I see my plugins in the Apache Airflow v2.0.2 Admin Plugins menu?

For security reasons, the Apache Airflow Web server on Amazon MWAA has limited network egress, and does not install plugins nor Python dependencies directly on the Apache Airflow *web server* for version 2.0.2 environments. The plugin that's shown allows Amazon MWAA to authenticate your Apache Airflow users in AWS Identity and Access Management (IAM).

To be able to install plugins and Python dependencies directly on the web server, we recommend creating a new environemnt with Apache Airflow v2.2 and above. Amazon MWAA installs Python dependencies and and custom plugins directly on the web server for Apache Airflow v2.2 and above.

Can I use AWS Database Migration Service (DMS) Operators?

Amazon MWAA supports DMS Operators. However, this operator cannot be used to perform actions on the Amazon Aurora PostgreSQL metadata database associated with an Amazon MWAA environment.

When I access the Airflow REST API using the AWS credentials, can I increase the throttling limit to more than 10 transactions per second (TPS)?

Yes, you can. To increase the throttling limit, please contact AWS Customer Support.

Troubleshooting Amazon Managed Workflows for Apache Airflow

This chapter describes common issues and errors you may encounter when using Apache Airflow on Amazon Managed Workflows for Apache Airflow and recommended steps to resolve these errors.

Contents

- Troubleshooting: DAGs, Operators, Connections, and other issues in Apache Airflow v2
  - Connections
    - I can't connect to Secrets Manager
    - How do I configure secretsmanager:ResourceTag/<tag-key> secrets manager conditions or a resource restriction in my execution role policy?
    - I can't connect to Snowflake
    - I can't see my connection in the Airflow UI
  - Web server
    - I see a 5xx error accessing the web server
    - I see a 'The scheduler does not appear to be running' error
  - Tasks
    - I see my tasks stuck or not completing
  - CLI
    - I see a '503' error when triggering a DAG in the CLI
    - Why does the dags backfill Apache Airflow CLI command fail? Is there a workaround?
  - Operators
    - I received a PermissionError: [Errno 13] Permission denied error using the S3Transform operator
- Troubleshooting: Creating and updating an Amazon MWAA environment
  - Updating requirements.txt
    - I specified a new version of my requirements.txt and it's taking more than 20 minutes to update my environment
  - Plugins

    - Does Amazon MWAA support implementing custom UI?
    - I am able to implement custom UI changes on the Amazon MWAA local runner via plugins, yet when I try to do the same on Amazon MWAA, I do not see my changes nor any errors. Why is this happening?
- Create bucket
    - I can't select the option for S3 Block Public Access settings
- Create environment
    - I tried to create an environment and it's stuck in the "Creating" state
    - I tried to create an environment but it shows the status as "Create failed"
    - I tried to select a VPC and received a "Network Failure" error
    - I tried to create an environment and received a service, partition, or resource "must be passed" error
    - I tried to create an environment and it shows the status as "Available" but when I try to access the Airflow UI an "Empty Reply from Server" or "502 Bad Gateway" error is shown
    - I tried to create an environment and my user name is a bunch of random character names
- Update environment
    - I tried changing the environment class but the update failed
- Access environment
    - I can't access the Apache Airflow UI
- Troubleshooting: CloudWatch Logs and CloudTrail errors
    - Logs
        - I can't see my task logs, or I received a 'Reading remote log from Cloudwatch log_group' error
        - Tasks are failing without any logs
        - I see a 'ResourceAlreadyExistsException' error in CloudTrail
        - I see an 'Invalid request' error in CloudTrail
        - I see a 'Cannot locate a 64-bit Oracle Client library: "libclntsh.so: cannot open shared object file: No such file or directory' in Apache Airflow logs
        - I see psycopg2 'server closed the connection unexpectedly' in my Scheduler logs
        - I see 'Executor reports task instance %s finished (%s) although the task says its %s' in my DAG processing logs

- I see 'Could not read remote logs from log_group: airflow-*{*environmentName}-Task log_stream:* {*DAG_ID}/*{*TASK_ID}/*{*time}/*{*n}.log.' in my task logs

Troubleshooting: DAGs, Operators, Connections, and other issues in Apache Airflow v2

The topics on this page describe resolutions to Apache Airflow v2 Python dependencies, custom plugins, DAGs, Operators, Connections, tasks, and *Web server* issues you may encounter on an Amazon Managed Workflows for Apache Airflow environment.

Contents

- Connections
  - I can't connect to Secrets Manager
  - How do I configure secretsmanager:ResourceTag/<tag-key> secrets manager conditions or a resource restriction in my execution role policy?
  - I can't connect to Snowflake
  - I can't see my connection in the Airflow UI
- Web server
  - I see a 5xx error accessing the web server
  - I see a 'The scheduler does not appear to be running' error
- Tasks
  - I see my tasks stuck or not completing
- CLI
  - I see a '503' error when triggering a DAG in the CLI
  - Why does the dags backfill Apache Airflow CLI command fail? Is there a workaround?
- Operators
  - I received a PermissionError: [Errno 13] Permission denied error using the S3Transform operator

Connections

The following topic describes the errors you may receive when using an Apache Airflow connection, or using another AWS database.

I can't connect to Secrets Manager

We recommend the following steps:

1. Learn how to create secret keys for your Apache Airflow connection and variables in the section called "Configuring Secrets Manager".
2. Learn how to use the secret key for an Apache Airflow variable (`test-variable`) in Using a secret key in AWS Secrets Manager for an Apache Airflow variable.
3. Learn how to use the secret key for an Apache Airflow connection (`myconn`) in Using a secret key in AWS Secrets Manager for an Apache Airflow connection.

How do I configure `secretsmanager:ResourceTag/<tag-key>` secrets manager conditions or a resource restriction in my execution role policy?

> **Note**
>
> Applies to Apache Airflow version 2.0 and earlier.

Currently, you cannot limit access to Secrets Manager secrets by using condition keys or other resource restrictions in your environment's execution role, due to a known issue in Apache Airflow.

I can't connect to Snowflake

We recommend the following steps:

1. Test your DAGs, custom plugins, and Python dependencies locally using the aws-mwaa-local-runner on GitHub.
2. Add the following entries to the requirements.txt for your environment.

```
apache-airflow-providers-snowflake==1.3.0
```

3. Add the following imports to your DAG:

```
from airflow.providers.snowflake.operators.snowflake import SnowflakeOperator
```

Ensure the Apache Airflow connection object includes the following key-value pairs:

1. **Conn Id:** snowflake_conn
2. **Conn Type:** Snowflake
3. **Host:** <my account>.<my region if not us-west-2>.snowflakecomputing.com
4. **Schema:** <my schema>
5. **Login:** <my user name>
6. **Password:** ********
7. **Port:** <port, if any>
8. **Extra:**

```
{
    "account": "<my account>",
    "warehouse": "<my warehouse>",
    "database": "<my database>",
    "region": "<my region if not using us-west-2 otherwise omit this line>"
}
```

For example:

```
>>> import json
>>> from airflow.models.connection import Connection
>>> myconn = Connection(
...    conn_id='snowflake_conn',
...    conn_type='Snowflake',
...    host='YOUR_ACCOUNT.YOUR_REGION.snowflakecomputing.com',
...    schema='YOUR_SCHEMA'
...    login='YOUR_USERNAME',
...    password='YOUR_PASSWORD',
...    port='YOUR_PORT'
...    extra=json.dumps(dict(account='YOUR_ACCOUNT', warehouse='YOUR_WAREHOUSE',
 database='YOUR_DB_OPTION', region='YOUR_REGION')),
... )
```

I can't see my connection in the Airflow UI

Apache Airflow provides connection templates in the Apache Airflow UI. It uses this to generate the connection URI string, regardless of the connection type. If a connection template is not available in the Apache Airflow UI, an alternate connection template can be used to generate a connection URI string, such as using the HTTP connection template.

We recommend the following steps:

1. View the connection types Amazon MWAA's providing in the Apache Airflow UI at Apache Airflow provider packages installed on Amazon MWAA environments.
2. View the commands to create an Apache Airflow connection in the CLI at Apache Airflow CLI command reference.
3. Learn how to use connection templates in the Apache Airflow UI interchangeably for connection types that aren't available in the Apache Airflow UI on Amazon MWAA at Overview of connection types.

Web server

The following topic describes the errors you may receive for your Apache Airflow *Web server* on Amazon MWAA.

I see a 5xx error accessing the web server

We recommend the following steps:

1. Check Apache Airflow configuration options. Verify that the key-value pairs you specified as an Apache Airflow configuration option, such as AWS Secrets Manager, were configured correctly. To learn more, see I can't connect to Secrets Manager.
2. Check the `requirements.txt`. Verify the Airflow "extras" package and other libraries listed in your `requirements.txt` are compatible with your Apache Airflow version.
3. Explore ways to specify Python dependencies in a `requirements.txt` file, see Managing Python dependencies in requirements.txt.

I see a 'The scheduler does not appear to be running' error

If the scheduler doesn't appear to be running, or the last "heart beat" was received several hours ago, your DAGs may not appear in Apache Airflow, and new tasks will not be scheduled.

We recommend the following steps:

1. Confirm that your VPC security group allows inbound access to port 5432. This port is needed to connect to the Amazon Aurora PostgreSQL metadata database for your environment. After this rule is added, give Amazon MWAA a few minutes, and the error should disappear. To learn more, see the section called “Security in your VPC”.

> **Note**
>
> - The Aurora PostgreSQL metadatabase is part of the Amazon MWAA service architecture and is not visible in your AWS account.
> - Database-related errors are usually a symptom of scheduler failure and not the root cause.

2. If the scheduler is not running, it might be due to a number of factors such as dependency installation failures, or an overloaded scheduler. Confirm that your DAGs, plugins, and requirements are working correctly by viewing the corresponding log groups in CloudWatch Logs. To learn more, see *Monitoring and metrics*.

Tasks

The following topic describes the errors you may receive for Apache Airflow tasks in an environment.

I see my tasks stuck or not completing

If your Apache Airflow tasks are "stuck" or not completing, we recommend the following steps:

1. There may be a large number of DAGs defined. Reduce the number of DAGs and perform an update of the environment (such as changing a log level) to force a reset.

   a. Airflow parses DAGs whether they are enabled or not. If you're using greater than 50% of your environment's capacity you may start overwhelming the Apache Airflow *Scheduler*. This leads to large *Total Parse Time* in CloudWatch Metrics or long DAG processing times in CloudWatch Logs. There are other ways to optimize Apache Airflow configurations which are outside the scope of this guide.

   b. To learn more about the best practices we recommend to tune the performance of your environment, see the section called “Performance tuning for Apache Airflow”.

2. There may be a large number of tasks in the queue. This often appears as a large—and growing—number of tasks in the "None" state, or as a large number in *Queued Tasks* and/or *Tasks Pending* in CloudWatch. This can occur for the following reasons:

a. If there are more tasks to run than the environment has the capacity to run, and/or a large number of tasks that were queued before autoscaling has time to detect the tasks and deploy additional *Workers*.

b. If there are more tasks to run than an environment has the capacity to run, we recommend **reducing** the number of tasks that your DAGs run concurrently, and/or increasing the minimum Apache Airflow *Workers*.

c. If there are a large number of tasks that were queued before autoscaling has had time to detect and deploy additional workers, we recommend **staggering** task deployment and/or increasing the minimum Apache Airflow *Workers*.

d. You can use the update-environment command in the AWS Command Line Interface (AWS CLI) to change the minimum or maximum number of *Workers* that run on your environment.

```
aws mwaa update-environment --name MyEnvironmentName --min-workers 2 --max-
workers 10
```

e. To learn more about the best practices we recommend to tune the performance of your environment, see the section called “Performance tuning for Apache Airflow”.

3. There may be tasks being deleted mid-execution that appear as task logs which stop with no further indication in Apache Airflow. This can occur for the following reasons:

a. If there is a brief moment where 1) the current tasks exceed current environment capacity, followed by 2) a few minutes of no tasks executing or being queued, then 3) new tasks being queued.

b. Amazon MWAA autoscaling reacts to the first scenario by adding additional workers. In the second scenario, it removes the additional workers. Some of the tasks being queued may result with the workers in the process of being removed, and will end when the container is deleted.

c. We recommend increasing the minimum number of workers on your environment. Another option is to adjust the timing of your DAGs and tasks to ensure that that these scenarios don't occur.

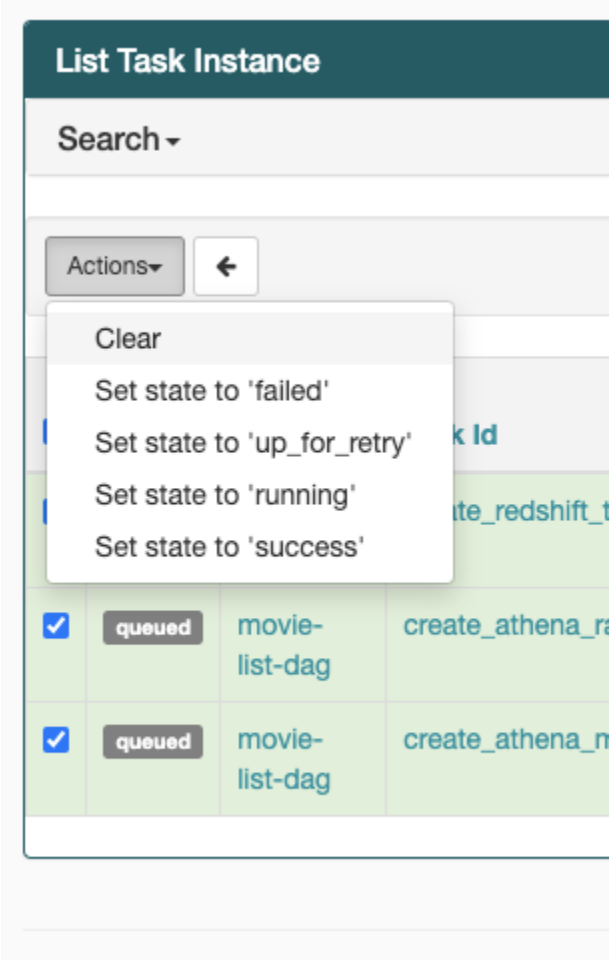
d. You can also set the minimum workers equal to the maximum workers on your environment, effectively disabling autoscaling. Use the update-environment command in the AWS Command Line Interface (AWS CLI) to **disable autoscaling** by setting the minimum and maximum number of workers to be the same.

```
aws mwaa update-environment --name MyEnvironmentName --min-workers 5 --max-
workers 5
```

e. To learn more about the best practices we recommend to tune the performance of your environment, see the section called “Performance tuning for Apache Airflow”.

4. If your tasks are stuck in the "running" state, you can also clear the tasks or mark them as succeeded or failed. This allows the autoscaling component for your environment to scale down the number of workers running on your environment. The following image shows an example of a stranded task.

- Choose the circle for the stranded task, and then select **Clear** (as shown). This allows Amazon MWAA to scale down workers; otherwise, Amazon MWAA can't determine which DAGs are enabled or disabled, and can't scale down, if there are still queued tasks.



5. Learn more about the Apache Airflow task lifecycle at Concepts in the *Apache Airflow reference guide*.

CLI

The following topic describes the errors you may receive when running Airflow CLI commands in the AWS Command Line Interface.

I see a '503' error when triggering a DAG in the CLI

The Airflow CLI runs on the Apache Airflow *Web server*, which has limited concurrency. Typically a maximum of 4 CLI commands can run simultaneously.

Why does the `dags backfill` Apache Airflow CLI command fail? Is there a workaround?

> **Note**
>
> The following applies only to Apache Airflow v2.0.2 environments.

The `backfill` command, like other Apache Airflow CLI commands, parses all DAGs locally before any DAGs are processed, regardless of which DAG the CLI operation applies to. In Amazon MWAA environments using Apache Airflow v2.0.2, because plugins and requirements are not yet installed on the web server by the time the CLI command runs, the parsing operation fails, and the `backfill` operation is not invoked. If you did not have any requirements nor plugins in your environment, the `backfill` operation would succeed.

In order to be able to run the `backfill` CLI command, we recommend invoking it in a bash operator. In a bash operator, `backfill` is initiated from the worker, allowing the DAGs to parse successfully as all necessary requirements and plguins are available and installed. The following example shows how you can create a DAG with a `BashOperator` to run `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"
    )
```

Operators

The following topic describes the errors you may receive when using Operators.

I received a `PermissionError: [Errno 13] Permission denied` error using the S3Transform operator

We recommend the following steps if you're trying to run a shell script with the *S3Transform* operator and you're receiving a `PermissionError: [Errno 13] Permission denied` error. The following steps assume you have an existing plugins.zip file. If you're creating a *new* plugins.zip, see Installing custom plugins.

1. Test your DAGs, custom plugins, and Python dependencies locally using the aws-mwaa-local-runner on GitHub.
2. Create your "transform" script.

   ```
   #!/bin/bash
   cp $1 $2
   ```

3. (optional) macOS and Linux users may need to run the following command to ensure the script is executable.

   ```
   chmod 777 transform_test.sh
   ```

4. Add the script to your plugins.zip.

   ```
   zip plugins.zip transform_test.sh
   ```

5. Follow the steps in Upload the plugins.zip to Amazon S3.
6. Follow the steps in Specifying the plugins.zip version on the Amazon MWAA console.
7. Create the following DAG.

   ```
   from airflow import DAG
   from airflow.providers.amazon.aws.operators.s3_file_transform import
    S3FileTransformOperator
   ```

```
from airflow.utils.dates import days_ago
import os

DAG_ID = os.path.basename(__file__).replace(".py", "")

with DAG (dag_id=DAG_ID, schedule_interval=None, catchup=False,
 start_date=days_ago(1)) as dag:
    file_transform = S3FileTransformOperator(
        task_id='file_transform',
        transform_script='/usr/local/airflow/plugins/transform_test.sh',
        source_s3_key='s3://YOUR_S3_BUCKET/files/input.txt',
        dest_s3_key='s3://YOUR_S3_BUCKET/files/output.txt'
    )
```

8. Follow the steps in Uploading DAG code to Amazon S3.

Troubleshooting: Creating and updating an Amazon MWAA environment

The topics on this page contains errors you may encounter when creating and updating an Amazon Managed Workflows for Apache Airflow environment and how to resolve these errors.

Contents

- Updating requirements.txt
  - I specified a new version of my requirements.txt and it's taking more than 20 minutes to update my environment
- Plugins
  - Does Amazon MWAA support implementing custom UI?
  - I am able to implement custom UI changes on the Amazon MWAA local runner via plugins, yet when I try to do the same on Amazon MWAA, I do not see my changes nor any errors. Why is this happening?
- Create bucket
  - I can't select the option for S3 Block Public Access settings
- Create environment
  - I tried to create an environment and it's stuck in the "Creating" state
  - I tried to create an environment but it shows the status as "Create failed"

- I tried to select a VPC and received a "Network Failure" error
- I tried to create an environment and received a service, partition, or resource "must be passed" error
- I tried to create an environment and it shows the status as "Available" but when I try to access the Airflow UI an "Empty Reply from Server" or "502 Bad Gateway" error is shown
- I tried to create an environment and my user name is a bunch of random character names
- Update environment
  - I tried changing the environment class but the update failed
- Access environment
  - I can't access the Apache Airflow UI

Updating `requirements.txt`

The following topic describes the errors you may receive when updating your `requirements.txt`.

I specified a new version of my `requirements.txt` and it's taking more than 20 minutes to update my environment

If it takes more than twenty minutes for your environment to install a new version of a `requirements.txt` file, the environment update failed and Amazon MWAA is rolling back to the last stable version of the container image.

1. Check package versions. We recommend always specifying either a specific version (==) or a maximum version (<=) for the Python dependencies in your `requirements.txt`.
2. Check Apache Airflow logs. If you enabled Apache Airflow logs, verify your log groups were created successfully on the Logs groups page on the CloudWatch console. If you see blank logs, the most common reason is due to missing permissions in your execution role for CloudWatch or Amazon S3 where logs are written. To learn more, see Execution role.
3. Check Apache Airflow configuration options. If you're using Secrets Manager, verify that the key-value pairs you specified as an Apache Airflow configuration option were configured correctly. To learn more, see the section called “Configuring Secrets Manager”.
4. Check VPC network configuration. To learn more, see the section called “Environment stuck”.
5. Check execution role permissions. An execution role is an AWS Identity and Access Management (IAM) role with a permissions policy that grants Amazon MWAA permission to invoke the resources of other AWS services (such as Amazon S3, CloudWatch, Amazon SQS,

Amazon ECR) on your behalf. Your Customer managed key or AWS owned key also needs to be permitted access. To learn more, see Execution role.

6. To run a troubleshooting script that checks the Amazon VPC network setup and configuration for your Amazon MWAA environment, see the Verify Environment script in AWS Support Tools on GitHub.

Plugins

The following topic describes issues you may encounter when configuring or updating Apache Airflow plugins.

Does Amazon MWAA support implementing custom UI?

Starting with Apache Airflow v2.2.2, Amazon MWAA supports installing plugins on the Apache Airflow web server, and implementing custom UI. If your Amazon MWAA environment is running Apache Airflow v2.0.2 or older, you will not be able to implement custom UI.

For more information about version management, and upgrading your existing environments, see *Versions*.

I am able to implement custom UI changes on the Amazon MWAA local runner via plugins, yet when I try to do the same on Amazon MWAA, I do not see my changes nor any errors. Why is this happening?

the Amazon MWAA local runner has all the Apache Airflow components bundled into one image, allowing you to apply custom UI plugin changes.

Create bucket

The following topic describes the errors you may receive when creating an Amazon S3 bucket.

I can't select the option for S3 Block Public Access settings

The execution role for your Amazon MWAA environment needs permission to the `GetBucketPublicAccessBlock` action on the Amazon S3 bucket to verify the bucket blocked public access. We recommend the following steps:

1. Follow the steps to Attach a JSON policy to your execution role.

2. Attach the following JSON policy:

```
{
   "Effect":"Allow",
   "Action":[
      "s3:GetObject*",
      "s3:GetBucket*",
      "s3:List*"
   ],
   "Resource":[
      "arn:aws:s3:::YOUR_S3_BUCKET_NAME",
      "arn:aws:s3:::YOUR_S3_BUCKET_NAME/*"
   ]
}
```

   Substitute the sample placeholders in *YOUR_S3_BUCKET_NAME* with your Amazon S3 bucket name, such as *my-mwaa-unique-s3-bucket-name*.

3. To run a troubleshooting script that checks the Amazon VPC network setup and configuration for your Amazon MWAA environment, see the Verify Environment script in AWS Support Tools on GitHub.

Create environment

The following topic describes the errors you may receive when creating an environment.

I tried to create an environment and it's stuck in the "Creating" state

We recommend the following steps:

1. Check VPC network with *public routing*. If you're using an Amazon VPC *with Internet access*, verify the following:

   - That your Amazon VPC is configured to allow network traffic between the different AWS resources used by your Amazon MWAA environment, as defined in the section called "About networking". For example, your VPC security group must either allow all traffic in a self-referencing rule, or optionally specify the port range for HTTPS port range 443 and a TCP port range 5432.

2. Check VPC network with *private routing*. If you're using an Amazon VPC *without Internet access*, verify the following:

- That your Amazon VPC is configured to allow network traffic between the different AWS resources for your Amazon MWAA environment, as defined in the section called "About networking". For example, your two private subnets must **not** have a route table to a NAT gateway (or NAT instance), **nor** an Internet gateway.

3. To run a troubleshooting script that checks the Amazon VPC network setup and configuration for your Amazon MWAA environment, see the Verify Environment script in AWS Support Tools on GitHub.

I tried to create an environment but it shows the status as "Create failed"

We recommend the following steps:

1. Check VPC network configuration. To learn more, see the section called "Environment stuck".
2. Check user permissions. Amazon MWAA performs a dry run against a user's credentials before creating an environment. Your AWS account may not have permission in AWS Identity and Access Management (IAM) to create some of the resources for an environment. For example, if you chose the **Private network** Apache Airflow access mode, your AWS account must have been granted access by your administrator to the AmazonMWAAFullConsoleAccess access control policy for your environment, which allows your account to create VPC endpoints.
3. Check execution role permissions. An execution role is an AWS Identity and Access Management (IAM) role with a permissions policy that grants Amazon MWAA permission to invoke the resources of other AWS services (such as Amazon S3, CloudWatch, Amazon SQS, Amazon ECR) on your behalf. Your Customer managed key or AWS owned key also needs to be permitted access. To learn more, see Execution role.
4. Check Apache Airflow logs. If you enabled Apache Airflow logs, verify your log groups were created successfully on the Logs groups page on the CloudWatch console. If you see blank logs, the most common reason is due to missing permissions in your execution role for CloudWatch or Amazon S3 where logs are written. To learn more, see Execution role.
5. To run a troubleshooting script that checks the Amazon VPC network setup and configuration for your Amazon MWAA environment, see the Verify Environment script in AWS Support Tools on GitHub.
6. If you are using an Amazon VPC *without* internet access, ensure that you've created an Amazon S3 gateway endpoint, and granted the minimum required permisions to Amazon ECR to access Amazon S3. To learn more about creating an Amazon S3 gateway endpoint, see the following:

- Creating an Amazon VPC network without internet access
- Create the Amazon S3 gateway endpoint in the *Amazon Elastic Container Registry User Guide*

I tried to select a VPC and received a "Network Failure" error

We recommend the following steps:

- If you see a "Network Failure" error when you try to select an Amazon VPC when creating your environment, turn off any in-browser proxies that are running, and then try again.

I tried to create an environment and received a service, partition, or resource "must be passed" error

We recommend the following steps:

- You may be receiving this error because the URI you specified for your Amazon S3 bucket includes a '/' at the end of the URI. We recommend removing the '/' in the path. The value should be in the following format:

```
s3://your-bucket-name
```

I tried to create an environment and it shows the status as "Available" but when I try to access the Airflow UI an "Empty Reply from Server" or "502 Bad Gateway" error is shown

We recommend the following steps:

1. Check VPC security group configuration. To learn more, see the section called “Environment stuck”.
2. Confirm that any Apache Airflow packages you listed in the `requirements.txt` correspond to the Apache Airflow version you're running on Amazon MWAA. To learn more, see Installing Python dependencies.
3. To run a troubleshooting script that checks the Amazon VPC network setup and configuration for your Amazon MWAA environment, see the Verify Environment script in AWS Support Tools on GitHub.

I tried to create an environment and my user name is a bunch of random character names

- Apache Airflow has a maximum of 64 characters for user names. If your AWS Identity and Access Management (IAM) role exceeds this length, a hash algorithm is used to reduce it, while remaining unique.

Update environment

The following topic describes the errors you may receive when updating an environment.

I tried changing the environment class but the update failed

If you update your environment to a different environment class (such as changing an `mw1.medium` to an `mw1.small`), and the request to update your environment failed, the environment status goes into an `UPDATE_FAILED` state and the environment is rolled back to, and is billed according to, the previous stable version of an environment.

We recommend the following steps:

1. Test your DAGs, custom plugins, and Python dependencies locally using the aws-mwaa-local-runner on GitHub.
2. To run a troubleshooting script that checks the Amazon VPC network setup and configuration for your Amazon MWAA environment, see the Verify Environment script in AWS Support Tools on GitHub.

Access environment

The following topic describes the errors you may receive when accessing an environment.

I can't access the Apache Airflow UI

We recommend the following steps:

1. Check user permissions. You may not have been granted access to a permissions policy that allows you to view the Apache Airflow UI. To learn more, see the section called “Accessing an Amazon MWAA environment”.

2. Check network access. This may be because you selected the **Private network** access mode. If the URL of your Apache Airflow UI is in the following format `387fbcn-8dh4-9hfj-0dnd-834jhdfb-vpce.c10.us-west-2.airflow.amazonaws.com`, it means that you're using *private routing* for your Apache Airflow *Web server*. You can either update the Apache Airflow access mode to the **Public network** access mode, or create a mechanism to access the VPC endpoint for your Apache Airflow *Web server*. To learn more, see the section called “Managing access to VPC endpoints”.

Troubleshooting: CloudWatch Logs and CloudTrail errors

The topics on this page contains resolutions to Amazon CloudWatch Logs and AWS CloudTrail errors you may encounter on an Amazon Managed Workflows for Apache Airflow environment.

Contents

- Logs
  - I can't see my task logs, or I received a 'Reading remote log from Cloudwatch log_group' error
  - Tasks are failing without any logs
  - I see a 'ResourceAlreadyExistsException' error in CloudTrail
  - I see an 'Invalid request' error in CloudTrail
  - I see a 'Cannot locate a 64-bit Oracle Client library: "libclntsh.so: cannot open shared object file: No such file or directory' in Apache Airflow logs
  - I see psycopg2 'server closed the connection unexpectedly' in my Scheduler logs
  - I see 'Executor reports task instance %s finished (%s) although the task says its %s' in my DAG processing logs
  - I see 'Could not read remote logs from log_group: airflow-*{*environmentName}-Task log_stream:* {*DAG_ID}/*{*TASK_ID}/*{*time}/*{*n}.log.' in my task logs

Logs

The following topic describes the errors you may receive when viewing Apache Airflow logs.

I can't see my task logs, or I received a 'Reading remote log from Cloudwatch log_group' error

Amazon MWAA has configured Apache Airflow to read and write logs directly from and to Amazon CloudWatch Logs. If a worker fails to start a task, or fails to write any logs, you will see the error:

```
*** 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.
```

- We recommend the following steps:

  a. Verify that you have enabled task logs at the `INFO` level for your environment. For more information, see Viewing Airflow logs in Amazon CloudWatch.

  b. Verify that the environment execution role has the correct permission policies.

  c. Verify that your operator or task is working correctly, has sufficient resources to parse the DAG, and has the appropriate Python libraries to load. To verify your whether you have the correct dependencies, try eliminating imports until you find the one that is causing the issue. We recommend testing your Python dependencies using the Amazon MWAA local-runner tool.

Tasks are failing without any logs

If tasks are failing in a workflow and you can't locate any logs for the failed tasks, check if you are setting the `queue` parameter in your default arguments, as shown in the following.

```
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'] }}"
    )
```

To resovle the issue, remove `queue` from your code, and invoke the DAG again.

I see a 'ResourceAlreadyExistsException' error in CloudTrail

```
"errorCode": "ResourceAlreadyExistsException",
    "errorMessage": "The specified log stream already exists",
    "requestParameters": {
        "logGroupName": "airflow-MyAirflowEnvironment-DAGProcessing",
        "logStreamName": "scheduler_cross-account-eks.py.log"
    }
```

Certain Python requirements such as `apache-airflow-backport-providers-amazon` roll back the `watchtower` library that Amazon MWAA uses to communicate with CloudWatch to an older version. We recommend the following steps:

- Add the following library to your `requirements.txt`

  ```
  watchtower==1.0.6
  ```

I see an 'Invalid request' error in CloudTrail

```
Invalid request provided: Provided role does not have sufficient permissions for s3
 location airflow-xxx-xxx/dags
```

If you're creating an Amazon MWAA environment and an Amazon S3 bucket using the same AWS CloudFormation template, you need to add a `DependsOn` section within your AWS CloudFormation template. The two resources (*MWAA Environment* and *MWAA Execution Policy*) have a dependency in AWS CloudFormation. We recommend the following steps:

- Add the following **`DependsOn`** statement to your AWS CloudFormation template.

  ```
  ...
        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:
...
```

For an example, see Quick start tutorial for Amazon Managed Workflows for Apache Airflow.

I see a 'Cannot locate a 64-bit Oracle Client library: "libclntsh.so: cannot open shared object file: No such file or directory' in Apache Airflow logs

- We recommend the following steps:
  - If you're using Apache Airflow v2, add `core.lazy_load_plugins : False` as an Apache Airflow configuration option. To learn more, see Using configuration options to load plugins in 2.

I see psycopg2 'server closed the connection unexpectedly' in my Scheduler logs

If you see an error similar to the following, your Apache Airflow *Scheduler* may have run out of resources.

```
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.
```

We recommend the following steps:

- Consider upgrading to Apache Airflow v2.0.2, which allows you to specify up to 5 *Schedulers*.

I see 'Executor reports task instance %s finished (%s) although the task says its %s' in my DAG processing logs

If you see an error similar to the following, your long-running tasks may have reached the task time limit on Amazon MWAA. Amazon MWAA has a limit of 12 hours for any one Airflow task, to prevent tasks from getting stuck in the queue and blocking activities like autoscaling.

```
Executor reports task instance %s finished (%s) although the task says its %s. (Info:
 %s) Was the task killed externally
```

We recommend the following steps:

- Consider breaking up the task into multiple, shorter running tasks. Airflow typically has a model whereby operators are asynchronous. It invokes activities on external systems, and Apache Airflow Sensors poll to see when its complete. If a Sensor fails, it can be safely retried without impacting the Operator's functionality.

I see 'Could not read remote logs from log_group: airflow-*{*environmentName}-Task log_stream:* {*DAG_ID}/*{*TASK_ID}/*{*time}/*{*n}.log.' in my task logs

If you see an error similar to the following, the execution role for your environment may not contain a permissions policy to create log streams for task logs.

```
Could not read remote logs from log_group: airflow-*{*environmentName}-Task
 log_stream:* {*DAG_ID}/*{*TASK_ID}/*{*time}/*{*n}.log.
```

We recommend the following steps:

- Modify the execution role for your environment using one of the sample policies at the section called “Execution role”.

You may have also specified a provider package in your `requirements.txt` file that is incompatible with your Apache Airflow version. For example, if you're using Apache Airflow v2.0.2, you may have specified a package, such as the apache-airflow-providers-databricks package, which is only compatible with Airflow 2.1+.

We recommend the following steps:

1. If you're using Apache Airflow v2.0.2, modify the `requirements.txt` file and add `apache-airflow[databricks]`. This installs the correct version of the Databricks package that is compatible with Apache Airflow v2.0.2.
2. Test your DAGs, custom plugins, and Python dependencies locally using the aws-mwaa-local-runner on GitHub.

Amazon MWAA Document History

The following table describes important additions to the Amazon MWAA service documentation, beginning in November 2020. To receive notifications about updates to this documentation, subscribe to the RSS feed.

Change	Description	Date
New Apache Airflow version	Amazon MWAA now supports Apache Airflow v2.10.3. This update includes informati on on updated provider packages, and details about using Apache Airflow v2.10.3 on Amazon MWAA. • *Versions* • the section called “Provider packages for Apache Airflow v2.10.3 connectio ns”	December 18, 2024
Added a new environment class: mw1.micro	Amazon MWAA now provides a new environment class: mw1.micro. • the section called “Configur ing the environment class” • the section called “Performance tuning for Apache Airflow”	November 19, 2024
Support for simpler method to access Apache Airflow REST API	Amazon MWAA now provides a simplified approach for interacting with the Apache Airflow REST API using AWS credentials.	October 23, 2024

	• the section called “Using the Apache Airflow REST API” • the section called “Apache Airflow Rest API access”	
New Apache Airflow version	Amazon MWAA now supports Apache Airflow v2.10.1. This update includes informati on on updated provider packages, and details about using Apache Airflow v2.10.1 on Amazon MWAA. • *Versions* • the section called “Provider packages for Apache Airflow v2.10.1 connectio ns”	September 26, 2024
New Apache Airflow version	Amazon MWAA now supports Apache Airflow v2.9.2. This update includes informati on on updated provider packages, and details about using Apache Airflow v2.9.2 on Amazon MWAA. • *Versions* • the section called “Provider packages for Apache Airflow v2.9.2 connections”	July 9, 2024

Amazon MWAA supports configuring a custom web server domain names	Amazon MWAA supports configuring a custom web server domain names for private environments with no internet access. This update includes the following new topic that describes setting up a new custom domain. • the section called “Setting up a custom domain”	June 18, 2024
Amazon MWAA supports web server automatic scaling and the Apache Airflow REST API	Amazon MWAA now supports automatic scaling of web servers as well as the ability to access and use the Apache Airflow REST API. • the section called “Configuring web server auto scaling” • the section called “Using the Apache Airflow REST API”	May 16, 2024
Improved description of automatic scaling behavior	Updated the following topic to reflect the new Amazon MWAA automatic scaling behavior when workers pick up new tasks as Fargate workers downscale. • the section called “Configuring worker auto scaling”	May 10, 2024

Support for larger instance sizes	Amazon MWAA now supports two larger instance size options for larger workloads : `mw1.xlarge` , and `mw1.2xlarge` • the section called "Environment capabilities"	April 16, 2024
New Apache Airflow version	Amazon MWAA now supports Apache Airflow v2.8.1. This update includes informati on on updated provider packages, and details about using Apache Airflow v2.8.1 on Amazon MWAA. • *Versions* • the section called "Provider packages for Apache Airflow v2.8.1 connections"	February 22, 2024

Support for shared Amazon VPC	Amazon MWAA supports cross-account environment creation for organizations using Amazon OpenSearch Service to manage Amazon MWAA resources using a central shared Amazon VPC in an *owner* account. As part of this launch, Amazon MWAA lets you choose to create, and manage, your own Amazon VPC endpoints. • the section called "Managing your own Amazon VPC endpoints"	November 15, 2023
New Apache Airflow version	Amazon MWAA now supports Apache Airflow v2.7.2. This update includes informati on on updated provider packages, and details about using Apache Airflow v2.7.2 on Amazon MWAA. • *Versions* • the section called "Provider packages for Apache Airflow v2.7.2 connections"	November 6, 2023

New Apache Airflow version	Amazon MWAA now supports Apache Airflow v2.6.3. This update includes informati on on updated provider packages, and details about using Apache Airflow v2.6.3 on Amazon MWAA, • *Versions* • the section called “Provider packages for Apache Airflow v2.6.3 connections”	August 9, 2023
Version deprecation informati on	Updated topic on version deprecation to include deprecation notices and timelines for Apache Airflow v2.0.2 and Apache Airflow v2.2.2. • the section called “Apache Airflow deprecated versions”	July 31, 2023
New topics and use cases	Amazon MWAA supports minor version upgrades. This updates includes the following new topic that describes how to upgrade the environment and make sure your workflow resources are compatible with the version of Apache Airflow you are upgrading to: • the section called “Upgrading the version”	June 5, 2023

Updated topic	Updated customer managed IAM policies that grant a user full console and API access to Amazon MWAA. The update describes why you must provide permission for `iam:PassRole` in order to allow a user to pass roles to Amazon MWAA. Amazon MWAA uses these permissions to perform actions on a user's behalf. • the section called “Accessing an Amazon MWAA environment”	April 12, 2023
New guidance	Updated topic on configuring AWS Secrets Manager as a backend for Amazon MWAA to provide guidance on using lookup patterns. Using lookup patterns narrow the secrets that Apache Airflow searches for and reduce the number of API calls Amazon MWAA makes to Secrets Manager to retrieve connections and variables. This reduces the costs associated with using Secrets Manager as a backend. • Create the Secrets Manager backend as an Apache Airflow configuration option	April 12, 2023

New Apache Airflow version	Amazon MWAA now supports Apache Airflow v2.5.1. This update includes informati on on updated provider packages, and details about using Apache Airflow v2.5.1 on Amazon MWAA, • *Versions* • the section called "Provider packages for Apache Airflow v2.5.1 connections"	April 11, 2023
New topics and use cases	Added a new topic on using a startup script with an Amazon MWAA environment. This topic descibes configuring a startup script for an existing environment, using it to install Linux runtimes, and setting environment variables . • the section called "Using a startup script"	April 3, 2023
Updated section on private web server access	Updated the following topic on private web server access. The update clarifies that, in environments with private web server access, you must use a Python wheel archive (`.whl`) to package, and install, dependencies. • Private web server access mode	February 24, 2023

Added information on deprecated Apache Airflow versions	Updated the Versions topic with new information on how Amazon MWAA managed deprecating Apache Airflow versions. Removed a section about upgrading to newer version of Apache Airflow, and a section that described changes between Apache Airflow v1 and Apache Airflow v2. For more informati on about migrating to a newversion of Apache Airflow, see the Amazon MWAA Migration Guide. • the section called “Apache Airflow deprecated versions” • the section called “Apache Airflow version support and FAQ”	February 17, 2023

Fixes in Amazon MWAA container metrics	Updated the container metrics topic, and removed a set of erroneous metrics that did not exist under the `Cluster` dimension. Added an additional section that describes how you can evaluate the number of additional workers that an environment is utilizing at a given time by graphing the `CPUUtilization` or the `MemoryUtilization` metric for the `AdditionalWorker` component, and setting the statistics type to `Sample Count`. • the section called "Evaluating the number of additional worker and web server containers"	January 20, 2023

New Apache Airflow version	Amazon MWAA now supports Apache Airflow v2.4.3. This update includes informati on on updated provider packages, details about using Apache Airflow v2.4.3 on Amazon MWAA, and consolidated information about which features are supported in each Apache Airflow version on Amazon MWAA. • *Versions* • the section called "Provider packages for Apache Airflow v2.4.3 connections"	January 5, 2023
Updated topic on service-l inked role	Updated information about the service-linked role that Amazon MWAA uses to create and manage AWS resources on your behalf, including information about how you can delete the service-linked role when you no longer need it. This includes an updated service-linked role permissio n policy that allows Amazon MWAA to publishe additiona l CloudWatch metrics under the `AWS/MWAA` namespace. • the section called "Service-linked role"	November 18, 2022

New topic on service metrics	Added new topic that describes service metrics emitted by Amazon MWAA under the `AWS/MWAA` namespace. These include Amazon ECS cluster metrics schedulers, workers, and web servers, Amazon SQS metrics for the queues that allow Amazon MWAA to decouple schedulers and workers, as well as Amazon RDS metrics for the metadata database. • the section called “Container, queue, and database metrics”	November 18, 2022
New topic	Added new guidance on modifying a constraints file to specify new versions of provider packages to use with your Amazon MWAA environment. • the section called “Specifying newer provider packages”	November 18, 2022
Updated FAQ entry	Updated information related to Amazon MWAA's HIPAA eligibility. • the section called “HIPAA compliance”	November 15, 2022

New topic	Added new topic on using `aws:SourceArn` and `aws:SourceAccount` global condition context keys in an Amazon MWAA execution role trust policy, in order to prevent cross-service confused deputy. • the section called “Cross-service confused deputy prevention”	October 21, 2022
New sample code	Added updated instructions and DAG code example that writes custom OS-level metrics to CloudWatch. • the section called “Using a DAG to write custom metrics”	September 13, 2022
New sample code	Added updated instructions and a new AWS Lambda Python code example that retrieves an Apache Airflow CLI token, then invokes a DAG in a specified Amazon MWAA environment. • the section called “Invoking DAGs with Lambda”	September 12, 2022

New architectural diagrams	Added new architectural diagrams that demonstrate an Amazon MWAA environment with a public and private web server. • the section called “Apache Airflow access modes”	September 12, 2022
New sample code	Added updated instructions and a new DAG code example that retrieves an Apache Airflow CLI token, then invokes another DAG in a different Amazon MWAA environment. • the section called “Invoking DAGs in different environments”	August 16, 2022
New sample code	Added updated instructions and new DAG that queries an environment's Aurora PostgreSQL for metadata information, writes the result to CSV files and stores the files in Amazon S3. • the section called “Exporting environment metadata to Amazon S3”	August 12, 2022

New sample code	Added updated instructions and new DAG that refreshes an AWS CodeArtifact token at runtime and stores the result in Amazon S3. • the section called “Refreshing an AWS CodeArtifact token at runtime”	August 3, 2022
New sample code	Added updated instructions and DAG code sample for using the `ECSOperator` in Amazon MWAA. • the section called “Using the `ECSOperator`”	July 26, 2022
New sample code	Added updated instructions and DAG code sample for using the `SSHOperator` in Amazon MWAA. • the section called “Using the `SSHOperator`”	July 15, 2022
New sample code	Added new instructions and DAG code sample for using dbt Postgres with Amazon MWAA. • the section called “Using dbt with Amazon MWAA”	June 17, 2022

New topics and use cases	Added new instructions and DAG code sample for installing dependencies using Python wheel files for Amazon MWAA environments with public and private access. • Managing dependencies using Python wheels	May 13, 2022
New topics and use cases	Added new guidance on choosing which Apache Airflow metrics Amazon MWAA sends to CloudWatch. • Choosing which Apache Airflow metrics are reported	April 19, 2022
New guides	Amazon MWAA offers a migration guide for migrating Apache Airflow workflows from self-managed deployments, as well as existing Amazon MWAA environments. • Amazon MWAA Migration Guide	March 7, 2022

New topics and use cases	Added new security best practice for working with Apache Airflow, including a solution for detecting changes to the Apache Airflow user privileges. • the section called “Security best practices in Apache Airflow”	February 18, 2022
New sample code	Added new code sample for creating timezone-aware DAGs using Pendulum, and clarified how to use a custom plugin to change the timezone in which Apache Airflow logs are created. • the section called “Changing a DAG's timezone”	February 11, 2022

Apache Airflow v2.2.2 launch	Amazon Managed Workflows for Apache Airflow now supports Apache Airflow v2.2.2. Beginning with v2.2, Amazon MWAA will install Python packages and custom plugins directly on the Apache Airflow web server allowing you greater flexibility to manage your environments. For more information, see the following. • Apache Airflow versions on Amazon Managed Workflows for Apache Airflow. • the section called “Provider packages for Apache Airflow v2.2.2 connections”. • Apache Airflow v2.2.2 changelog on the Apache Airflow documentation website.	January 27, 2022

New tutorials	Added a new tutorial that demonstrates creating a new custom Apache Airflow role, and assigning the role to an Apache Airflow user mapped from IAM in order to limit the user's access to a subset of specified DAGs. • the section called “Tutorial: Restricting users to a subset of DAGs”	December 8, 2021
Fixes	Fixed a best practices recommendation for setting the value of `scheduler.min_file_process_interval` in order to optimize CPU usage. Added an IAM policy example granting access to Secrets Manager resources in the execution role. Added troubleshooting topic on using Secrets Manager condition keys. • Performance tuning how the scheduler parses DAGs • Provide Amazon MWAA with permission to access Secrets Manager secret keys • Configuring condition keys in the Amazon MWAA execution role for Secrets Manager	November 22, 2021

New sample code	Added the following new code sample for modifying the time zone in which DAGs are processed using a custom plugin, and new troublesh ooting topic for invoking the `dags backfill` Apache Airflow CLI command from within a bash operator. • the section called "Changing a DAG's timezone" • Backfill CLI command using a bash operator	November 1, 2021
Fixes	Fixed issues in the Amazon ECS operator code sample, and clarified the additional permissions required in the Amazon MWAA execution role to allow the environment to access Amazon ECS task log group in CloudWatch Logs. • Amazon ECS operator permissions.	October 26, 2021

New sample code	Added new code sample that queries the Aurora PostgreSQL database for information relevant to DAG runs and writes the results to CSV file stored on Amazon S3. • the section called “Exporting environment metadata to Amazon S3”.	October 1, 2021
Fixes	Corrected information about how Amazon MWAA automatically syncs new and changed objects from your target Amazon S3 bucket to your schedulers and workers. • How the DAG folder works.	October 1, 2021
Now supported	Amazon MWAA now supports additional provider packages for Apache Airflow 2.0+. To learn more about supported packages, see the following: • the section called “Provider packages for Apache Airflow v2.0.2 connections”.	September 24, 2021

New commands and procedures	Added additional guidance and AWS CLI command examples for creating an Amazon S3 gateway endpoint when using an Amazon VPC without internet access: • Creating an Amazon VPC network without Internet access.	September 24, 2021
New topics and use cases	Added the following changes: • Added a new code sample that uses an Amazon Elastic Container Service operator in the section called "Using the `ECSOperator`". • Added new troublesh ooting topics for issues in configuring Apache Airflow plugins in the section called "Plugins".	September 19, 2021

New supported region	Amazon MWAA is now available in the following regions: • Asia Pacific (Mumbai) - ap-south-1 • Asia Pacific (Seoul) - ap-northeast-2 • Europe (London) - eu-west-2 • Europe (Paris) - eu-west-3 • Canada (Central) - ca-central-1 • South America (São Paulo) - sa-east-1 For more information about region availability and service endpoints, see the following: • Amazon MWAA endpoints and quotas in the *AWS General Reference*.	August 31, 2021
New topics and use cases	Added the following changes: • Updated the sample policies to allow Amazon MWAA to fetch account-level Amazon S3 settings (`s3:GetAccountPublicAccessBlock`) in Amazon MWAA execution role.	August 27, 2021

Fixes	Added the following changes: • Fixed the AWS CloudFormation template to use a self-referencing inbound rule for the security group in Create the VPC network. • Fixed the AWS CloudFormation template to use a self-referencing inbound rule for the security group in Quick start tutorial for Amazon Managed Workflows for Apache Airflow.	August 27, 2021
New topics and use cases	Added the following changes: • Added DAG decorator to the list of what's supported for Apache Airflow v2.0.2 Apache Airflow versions on Amazon Managed Workflows for Apache Airflow.	August 20, 2021

New topics and use cases	Added the following changes: • Added `celery.sync_parallelism` use case to Performance tuning for Apache Airflow on Amazon MWAA. • Added service endpoints to quotas page and changed name to Amazon Managed Workflows for Apache Airflow service endpoints and quotas. • Clarified networking prerequisites based on user feedback at Get started with Amazon Managed Workflows for Apache Airflow. • Moved `dags list-runs` and `dags next-execution` to unsupported Airflow CLI commands in Apache Airflow CLI command reference.	August 13, 2021

New sample code	Added the following changes: • Added bash example to set, get or delete an Apache Airflow v2.0.2 variable in Apache Airflow CLI command reference. • Added Apache Airflow v2.0.2 dependencies and Airflow connection example to Using Amazon MWAA with Amazon RDS for Microsoft SQL Server.	August 13, 2021
Fixes	Added the following changes: • Fixed the Python code sample based on user feedback at Creating an SSH connection using the `SSHOperator` .	August 13, 2021

New topics and use cases	Added the following changes: • Moved `variables set` to supported Airflow CLI commands in Apache Airflow CLI command reference. • Added the summary of **What's changed** in v2.0.2 from the Airflow versions page to Installing Python dependencies based on user feedback. • Added the summary of **What's changed** in v2.0.2 from the Airflow versions page to Apache Airflow CLI command reference based on user feedback. • Added the summary of **What's changed** in v2.0.2 from the Airflow versions page to Overview of connection types based on user feedback. • Added the summary of **What's changed** in v2.0.2 from the Airflow versions page to Installing custom plugins based on user feedback. • Added the summary of **What's changed** in v2.0.2 from the Airflow versions page to Adding or	August 6, 2021

	updating DAGs based on user feedback.	
New sample code	Added the following changes: • Added Apache Airflow v2.0.2 sample code to Using a DAG to import variables in the CLI. • Added Apache Airflow v2.0.2 sample code to Invoking DAGs with a Lambda function.	August 6, 2021
New topics and use cases	Added the following changes: • Added troubleshooting topic for 'I can't see my connection in the Airflow UI' at Troubleshooting Amazon Managed Workflows for Apache Airflow. • Added a list of Amazon VPCs Amazon MWAA supports to About networking on Amazon MWAA.	July 29, 2021

Fixes	Added the following changes: • Fixed the Python code sample based on user feedback to print the web login token at Create a Apache Airflow web server access token. • Fixed the Snowflake connection topic based on user feedback to use a single quote for the warehouse parameter at Troubleshooting Amazon Managed Workflows for Apache Airflow.	July 29, 2021
Removed or moved topics	Added the following changes: • Restructed the existing page to include all monitoring and metrics documentation pages in Monitoring and metrics for Amazon Managed Workflows for Apache Airflow. • Moved Apache Airflow v2 environment metrics in CloudWatch to the monitoring and metrics navigation menu.	July 23, 2021

New guides	Added the following changes: • Created Apache Airflow provider packages installed on Amazon MWAA environments. • Created Monitoring overview on Amazon MWAA. • Created Viewing audit logs in AWS CloudTrail. • Created Viewing Airflow logs in Amazon CloudWatch.	July 23, 2021
Fixes	Added the following changes: • Fixed the Python code sample based on user feedback to generate an Airflow connection string in the correct sequence and added the port parameter in Configuring an Apache Airflow connection using a AWS Secrets Manager secret. • Added a step to install an unzip package locally based on user feedback in Creating a custom plugin with Oracle.	July 23, 2021

New topics and use cases	Added the following changes: • Added topic for AWS DMS Operators at Amazon MWAA frequently asked questions. • Added troubleshooting topic for a remote logs error to Troubleshooting Amazon Managed Workflows for Apache Airflow. • Moved `variables set` to unsupported Airflow CLI commands in Apache Airflow CLI command reference.	July 16, 2021

New topics and use cases	Added the following changes: • Added sequential steps to create a requirements.txt file based on user feedback at Installing Python dependencies. • Added sequential steps to create a plugins.zip file based on user feedback at Installing custom plugins. • Added cross-reference links throughout the user guide to the API reference guide at Amazon Managed Workflows for Apache Airflow API Reference guide. • Added topic for why plugins aren't shown in the Airflow 2.0 Admin > Plugins menu at Amazon MWAA frequently asked questions.	July 9, 2021
New guides	Added the following changes: • Created Deleting files on Amazon S3.	July 9, 2021

New topics and use cases	Added the following changes: • Added a list of supported values at Using customer managed keys for encryption. • Updated and clarified the example for a private repo URL based on user feedback in Managing Python dependencies in requirements.txt.	July 2, 2021
New sample code	Added the following changes: • Added Apache Airflow v1.10.12 sample code to use a private key in AWS Secrets Manager for an SSH connection at Creating an SSH connection using the `SSHOperator` .	July 2, 2021
New topics and use cases	Added the following changes: • Added StartedTaskInstances and FinishedTaskInstances metrics to Apache Airflow v2 environment metrics in CloudWatch.	June 25, 2021
New sample code	Added the following changes: • Added Apache Airflow v2.0.2 sample code at Using Amazon MWAA with Amazon EKS.	June 25, 2021

New guides	Added the following changes: • Created Performance tuning for Apache Airflow on Amazon MWAA.	June 25, 2021

New topics and use cases	Added the following changes: • Added `connections add` and `connections delete` to the **supported** Apache Airflow v2.0.2 CLI commands at Apache Airflow CLI command reference. • Added that the latest version available in AWS CloudFormation is Apache Airflow v2.0.2 at Quick start tutorial for Amazon Managed Workflows for Apache Airflow. • Added question for storing temporary data on Apache Airflow Workers to Amazon MWAA frequently asked questions. • Added topic for the 'Executor reports task instance %s finished' error to Troubleshooting Amazon Managed Workflows for Apache Airflow. • Added topic for the 'server closed the connectio n unexpectedly' log to Troubleshooting Amazon Managed Workflows for Apache Airflow. • Added example to run CLI commands on an SSH	June 18, 2021

tunnel to a bastion host to Creating an Apache Airflow CLI token.

- Added topic for randomly-generated user names to Troubleshooting Amazon Managed Workflows for Apache Airflow.
- Added topic for a 503 error when running a DAG in the CLI to Troubleshooting Amazon Managed Workflows for Apache Airflow.
- Added topic for custom plugins in Apache Airflow v2.0.2 which need an Airflow configuration option of `core.lazy_load_plugins : False` to load plugins at the start of each Airflow process to override the version's default setting to Using Apache Airflow configuration options on Amazon MWAA.
- Added Airflow configuration options step for Apache Airflow v2.0.2 plugins sample code at Creating a custom plugin with Apache Hive and Hadoop.

	• Added Airflow configuration options step for Apache Airflow v2.0.2 plugins sample code at Creating a custom plugin that generates runtime environment variables. • Added Airflow configuration options step for Apache Airflow v2.0.2 plugins sample code at Creating a custom plugin for Apache Airflow PythonVirtualenvOperator. • Added Airflow configuration options step for Apache Airflow v2.0.2 plugins sample code at Creating a custom plugin with Oracle.	
New sample code	Added the following changes: • Added sample code for an Apache Airflow Snowflake connection at Using a secret key in AWS Secrets Manager for an Apache Airflow Snowflake connection.	June 18, 2021

New topics and use cases	Added the following changes: • Added server-side encryption guidance to Create an Amazon S3 bucket for Amazon MWAA. • Added the secrets backend for Apache Airflow v2.0.2 to Configuring an Apache Airflow connection using a AWS Secrets Manager secret. • Added question for Apache Airflow Workers quota increase requests to Amazon MWAA frequently asked questions. • Added question for which metrics are used to determine whether to scale Apache Airflow Workers to Amazon MWAA frequently asked questions. • Added question for creating custom metrics in CloudWatch to Amazon MWAA frequently asked questions. • Added steps to enable private IP addresses for an Amazon S3 VPC interface endpoint for a VPC with private routing in Creating the required VPC service	June 2, 2021

	endpoints in an Amazon VPC with private routing. • Added an option to setup an SSH Tunnel using local port forwarding in Tutorial: Configuring private network access using a Linux Bastion Host.	
New sample code	Added the following changes: • Added sample code for a DAG that queries the Amazon Aurora PostgreSQL metadata database and publishes custom metrics to Amazon CloudWatch at Using a DAG to write custom metrics in CloudWatch.	June 2, 2021
New guides	Added the following changes: • Created a guide on how to use connection templates interchangeably in the Apache Airflow UI in Overview of connection types.	June 2, 2021

Fixes	Added the following changes: • Added Apache Airflow VPC endpoints to the AWS CloudFormation template in *Option three: Creating a VPC network without Internet access* to Create the VPC network.	June 2, 2021

Apache Airflow v2.0.2 launch	General availability launch of Apache Airflow v2.0.2. • Created Apache Airflow versions on Amazon Managed Workflows for Apache Airflow. • Created Apache Airflow v2 environment metrics in CloudWatch. • Added version-specific links for Apache Airflow v2.0.2 to Using Apache Airflow configuration options on Amazon MWAA. • Added Apache Airflow v2.0.2 version-specific guidance to Installing Python dependencies. • Added Apache Airflow v2.0.2 version-specific guidance to Managing Python dependencies in requirements.txt. • Added Apache Airflow v2.0.2 sample plugins to Installing custom plugins. • Added Apache Airflow v2.0.2 sample code to Aurora PostgreSQL database cleanup on an Amazon MWAA environment. • Added Apache Airflow v2.0.2 sample code to Using	May 26, 2021

	a secret key in AWS Secrets Manager for an Apache Airflow connection. • Added Apache Airflow v2.0.2 sample code to Creating a custom plugin for Apache Airflow PythonVirtualenvOperator. • Added Apache Airflow v2.0.2 commands to Apache Airflow CLI command reference. • Added Apache Airflow v2.0.2 scripts to Creating an Apache Airflow CLI token. • Added a note that Amazon MWAA uses the latest Apache Airflow version by default to Create an Amazon MWAA environment.	
New topics and use cases	Added the following changes: • Added guidance to troubleshooting Airflow tasks that are stuck or not running to Troubleshooting Amazon Managed Workflows for Apache Airflow.	May 14, 2021

Fixes	Added the following changes: • We've updated the sample plugins code to use the latest Java version in Creating a custom plugin with Apache Hive and Hadoop. Previously, it was `os.environ["JAVA_HOME"]="/usr/lib/jvm/jre-1.8.0-openjdk-1.8.0.272.b10-1.amzn2.0.1.x86_64"` .	May 12, 2021
Removed or moved topics	Added the following changes: • Moved topics in Troubleshooting Amazon Managed Workflows for Apache Airflow to new pages by category.	May 10, 2021
New topics and use cases	Added the following changes: • Added Amazon S3 bucket overview to Working with DAGs on Amazon MWAA.	May 10, 2021

Removed or moved topics	Added the following changes: • Moved Accessing Apache Airflow to the top-level navigation, and added pages for Create a Apache Airflow web server access token, Creating an Apache Airflow CLI token, and Apache Airflow CLI command reference.	May 7, 2021
New topics and use cases	Added the following changes: • Added version-specific links to the *Apache Airflow reference guide* for all supported and unsupport ed Airflow CLI commands in Apache Airflow CLI command reference. • Added version-specific links to the *Apache Airflow reference guide* for all configuration options in Using Apache Airflow configuration options on Amazon MWAA. • Added the Amazon MWAA CLI utility to Managing Python dependencies in requirements.txt.	May 7, 2021

New topics and use cases	Added the following changes: • Added flat and nested examples for how to structure a plugins.zip in Installing custom plugins. • Added the Amazon MWAA CLI utility to the Adding or updating DAGs, Installing custom plugins, and Installing Python dependencies pages. • Restructured content into an overview, upload to Amazon S3, and installing on Amazon MWAA sections based on user feedback in Installing custom plugins, and Installing Python dependencies pages. • Added an example use case to create and attach required VPC endpoints to an existing Amazon VPC *without Internet access* in About networking on Amazon MWAA.	April 30, 2021

New sample code	Added the following changes: • Added sample code that uses a secret key in Secrets Manager for an Apache Airflow variable in Using a secret key in AWS Secrets Manager for an Apache Airflow variable.	April 30, 2021
New guides	Added the following changes: • Created Creating the required VPC service endpoints in an Amazon VPC with private routing.	April 30, 2021
Fixes	Added the following changes: • Oops! We've updated `core.default_ui_timezone` to `webserver.default_ui_timezone` in Using Apache Airflow configuration options on Amazon MWAA.	April 30, 2021

New topics and use cases	Added the following changes: • Added Windows (PuTTY) steps for SSH tunnel to Tutorial: Configuring private network access using a Linux Bastion Host. • Added topic for `apache-airflow-providers-amazon` , which is only compatible with Apache Airflow 2.0 to Troublesh ooting Amazon Managed Workflows for Apache Airflow.	April 23, 2021
New sample code	Added the following changes: • Added sample code that uses a secret key in Secrets Manager for an Apache Airflow connection in Using a secret key in AWS Secrets Manager for an Apache Airflow connection.	April 23, 2021
New guides	Added the following changes: • Created About networking on Amazon MWAA. • Created Security in your VPC on Amazon MWAA. • Created Managing access to service-specific Amazon VPC endpoints on Amazon MWAA.	April 23, 2021

New topics and use cases	Added the following changes: • Added a new AWS CloudFormation template to create an Amazon VPC network without Internet access in Create the VPC network. • Added a new tutorial to create an AWS Client VPN in Tutorial: Configuring private network access using an AWS Client VPN. • Changed the name of the **Networking access** page to **Apache Airflow access modes** based on user feedback, and streamlined docs in Apache Airflow access modes. • Streamlined docs to include only Amazon VPC getting started information and templates based on user feedback in Create the VPC network. • Added BigQuery operator workaround to Troubleshooting Amazon Managed Workflows for Apache Airflow. • Added an Apache Airflow v1.10.12 constraints file best practice to Installing Python dependencies.	April 16, 2021

New sample code	Added the following changes: • Added sample code to create a custom plugin using Oracle in Creating a custom plugin with Oracle. • Added sample code to create a custom plugin that generates runtime environment variables in Creating a custom plugin that generates runtime environment variables. •	April 16, 2021

New topics and use cases	Added the following changes: • Added topic for the self-referencing rule requirement on a VPC security group to Amazon MWAA frequently asked questions. • Added custom plugins directory and size limits to Installing custom plugins. • Added requirements directory and size limits to Installing Python dependencies. • Clarified the Apache Airflow configuration options for `foo.user` and `foo.pass` in Managing Python dependencies in requirements.txt. • Added configuration options overview to Using Apache Airflow configuration options on Amazon MWAA.	April 9, 2021

New sample code	Added the following changes: • Added sample code to create a custom plugin using PythonVirtualenvOperator in Creating a custom plugin for Apache Airflow PythonVirtualenvOperator. • Added sample code to create a custom plugin with Apache Hive and Hadoop in Creating a custom plugin with Apache Hive and Hadoop.	April 9, 2021
Fixes	Added the following changes: • Oops! We've updated the format for a requirements.txt, and added an example that's compatible with Apache Airflow v1.10.12 in Installing Python dependencies.	March 31, 2021

New topics and use cases	Added the following changes: • Added workaround to removing a requirements.txt or plugins.zip to Amazon MWAA frequently asked questions. • Added a bash workaround for SSH on an environment to Amazon MWAA frequently asked questions. • Added topic for CloudTrail ResourceAlreadyExistsException error to Troubleshooting Amazon Managed Workflows for Apache Airflow.	March 26, 2021

New topics and use cases	Added the following changes: • Added list of AWS services used to Amazon MWAA execution role. • Added list of AWS services used to Service-linked role for Amazon MWAA. • Added question for Python 3.7 version for Amazon MWAA to Amazon MWAA frequently asked questions. • Added question for PythonVirtualenvOperator to Amazon MWAA frequently asked questions. • Added the troubleshooting script as next steps for all topics related to VPC and environment configuration at Troubleshooting Amazon Managed Workflows for Apache Airflow. • Clarified the docs that a linux bastion must be in the same Region as an environment at Tutorial: Configuring private network access using a Linux Bastion Host.	March 19, 2021

New guides	Added the following changes: • Created Apache Airflow connections guide for AWS Secrets Manager at Configuring an Apache Airflow connection using a AWS Secrets Manager secret. • Created quick start tutorial using a AWS CloudForm ation template to create the Amazon VPC infrastru cture, Amazon S3 bucket, and Amazon MWAA environment at Quick start tutorial for Amazon Managed Workflows for Apache Airflow.	March 19, 2021
New topics and use cases	Added the following changes: • Added the create Amazon S3 bucket troublesh ooting topic Troublesh ooting Amazon Managed Workflows for Apache Airflow. • Added steps to create and attach a JSON policy to Amazon MWAA execution role.	March 12, 2021

New sample code	Added the following changes: • Added sample code to add a configuration when triggering a DAG to Accessing Apache Airflow.	March 12, 2021
New guides	Added the following changes: • Created best practices guide at Managing Python dependencies in requirements.txt.	March 12, 2021
New topics and use cases	Added the following changes: • Added Google/GCP/BigQuery troubleshooting topic to Troubleshooting Amazon Managed Workflows for Apache Airflow. • Added Cython troubleshooting topic to Troubleshooting Amazon Managed Workflows for Apache Airflow. • Added MySQL troubleshooting topic to Troubleshooting Amazon Managed Workflows for Apache Airflow. • Added 5xx web server error troubleshooting topic to Troubleshooting Amazon Managed Workflows for Apache Airflow.	March 5, 2021

Now supported	Added the following changes: • Previously, `backend_kwargs` was not supported for AWS Secrets Manager and you needed a workaround to override the Secrets Manager function call. Now, `backend_kwargs` is supported. See the AWS Secrets Manager troubleshooting topic in Troubleshooting Amazon Managed Workflows for Apache Airflow.	March 4, 2021
Fixes	Added the following changes: • Oops! We've updated the size of each environment class to reflect the actual GB in Configuring the Amazon MWAA environment class.	March 4, 2021

New topics and use cases	Added the following changes: • Added private network access using a VPC endpoint policy to Apache Airflow access modes. • Added additional checks for the creating an environme nt troubleshooting topic to Troubleshooting Amazon Managed Workflows for Apache Airflow. • Added steps to view logs for `requirements.txt` to Installing Python dependencies.	February 26, 2021
New topics and use cases	Added the following changes: • Added Apache Hive use case to Installing Python dependencies. • Clarified the docs that the required dependencies for an Apache Airflow package needs to be included in the `requirements.txt` file at Installing Python dependencies. • Added *Updating requireme nts.txt* troubleshooting topic to Troublesh ooting Amazon Managed Workflows for Apache Airflow.	February 25, 2021

New tutorials	Added the following changes: • Added private network tutorial to Tutorial: Configuring private network access using a Linux Bastion Host.	February 22, 2021
New topics and use cases	Added the following changes: • Added private and public network configurations to Apache Airflow access modes. • Added development group use case and user scenarios to Amazon MWAA execution role.	February 22, 2021
New sample code	Added the following changes: • Added sample Python scripts for web login token and CLI token to Accessing Apache Airflow. • Added sample code to trigger DAG in another environment to Code examples for Amazon Managed Workflows for Apache Airflow. • Added sample code to trigger DAG using a Lambda function to Invoking DAGs with a Lambda function.	February 22, 2021

New commands and procedures	Added the following changes: • Added step by step procedures to all scripts at Accessing Apache Airflow.	February 22, 2021
New sample code	Added the following changes: • Updated curl example for web login token at Accessing Apache Airflow. • Added sample code to connect to an Amazon RDS Microsoft SQL Server to Using Amazon MWAA with Amazon RDS for Microsoft SQL Server.	February 17, 2021

New commands and procedures	Added the following changes: • Added AWS CLI commands to Working with DAGs on Amazon MWAA pages. • Apache Airflow doesn't support serialized DAGs in CLI commands. Since the CLI runs on the web server, which doesn't have plugins or requirements for security reasons, any MWAA environments with a plugins.zip or requirements.txt will not support these commands. Moved Apache Airflow `list_dags` and `backfill` commands to unsupported commands at Accessing Apache Airflow.	February 17, 2021
GitHub launch	User guide docs are now open source on GitHub. Choose "Edit this page on GitHub" on any page.	February 17, 2021

New topics and use cases	Added the following changes: • Added question for Step Functions v. Amazon MWAA use case to Troubleshooting Amazon Managed Workflows for Apache Airflow. • Added CLI access policy to Accessing an Amazon MWAA environment. • Clarified the docs that any supported Apache Airflow configuration option can be specified at Using Apache Airflow configuration options on Amazon MWAA. • Clarified the docs that if a Fargate container in one availability zone fails, MWAA switches to the other container in a different availability zone at Create the VPC network.	February 12, 2021
New topics and use cases	Added the following changes: • Added Configuring the Amazon MWAA environment class.	February 5, 2021

Removed or moved topics	Added the following changes: • Removed requirement for Amazon S3 bucket name to start with `airflow-` at Get started with Amazon Managed Workflows for Apache Airflow. • Moved Accessing an Amazon MWAA environment and Amazon MWAA execution role to Managing access to an Amazon MWAA environment.	February 4, 2021
Amazon MWAA CloudFormation	Update the parameters to create an environment at Amazon MWAA CloudFormation. • Remove SubnetList. • Remove TagList. • Add NetworkConfiguration. • Add TagMap. • Add create environment request examples.	February 4, 2021

New topics and use cases	Added the following changes: • Added example email configuration to Using Apache Airflow configuration options on Amazon MWAA. • Added PostgresHook troubleshooting topic to Troubleshooting Amazon Managed Workflows for Apache Airflow. • Added AWS Secrets Manager troubleshooting topic to Troubleshooting Amazon Managed Workflows for Apache Airflow. • Added high performance use case to Configuring Amazon MWAA worker automatic scaling.	January 29, 2021
Amazon MWAA launch	General availability launch of Amazon Managed Workflows for Apache Airflow. • User guide documentation • AWS CloudFormation documentation	November 24, 2020