aws

Migration Guide

Amazon Managed Workflows for Apache Airflow

Amazon Managed Workflows for Apache Airflow: Migration Guide

Table of Contents

What is the migration guide? 1
Network architecture 2
Amazon MWAA components 2
Connectivity 4
Key considerations 5
Authentication 5
Execution role 5
Migrating to a new Amazon MWAA environment 7
Prerequisites 7
Step one: Create a new environment 7
Step two: Migrate your workflow resources 14
Step three: exporting the metadata 15
Step four: importing the metadata 17
Next steps 19
Related resources 20
Migrating workloads from AWS Data Pipeline to Amazon MWAA 21
Choosing Amazon MWAA 21
Architecture and concept mapping 22
Example implementations 24
Pricing comparison 24
Related resources 25
Document History 26

What is the Amazon MWAA migration guide?

Amazon Managed Workflows for Apache Airflow is a managed orchestration service for Apache Airflow that allows you to operate data pipelines in the cloud at scale. Amazon MWAA manages the provisioning and ongoing maintenance of Apache Airflow so you no longer need to worry about patching, scaling, or securing instances.

Amazon MWAA automatically scales the compute resources that execute tasks to provide consistent performance on demand. Amazon MWAA secures your data by default. Your workloads run in your own isolated and secure cloud environment using Amazon Virtual Private Cloud. This ensures that data is automatically encrypted using AWS Key Management Service.

Use this guide to migrate your self-managed Apache Airflow workflows to Amazon MWAA, or upgrade an existing Amazon MWAA environment to a new Apache Airflow version. The migration tutorial describes how you can create, or clone a new Amazon MWAA environment, migrate your workflow resources, and transfer your workflow metadata and logs to your new environment.

Before you attempt the migration tutorial, we recommend reviewing the following topics.

- *Network architecture*
- *Key considerations*

Amazon MWAA network architecture

The following section describes the main components that make up an Amazon MWAA environment, and the set of AWS services that each environment integrates with to manage its resources, keep your data secure, and provide monitoring and visibility for your workflows.

Topics

- Amazon MWAA components
- Connectivity

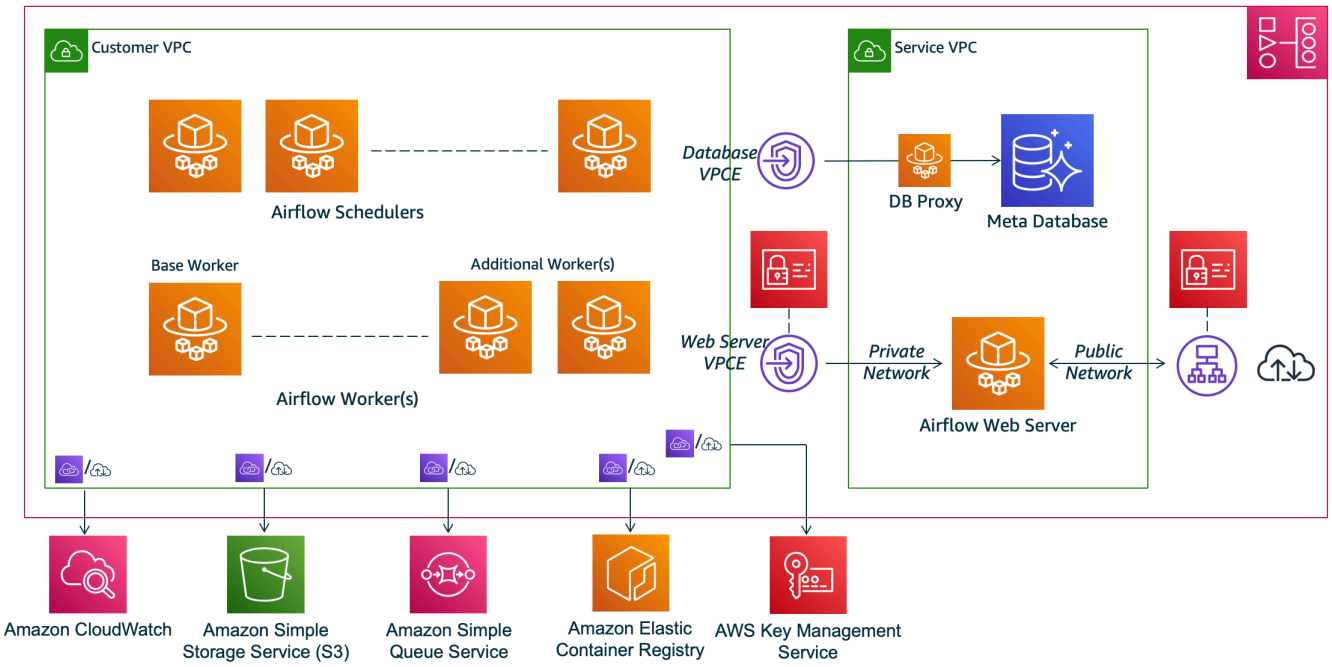
Amazon MWAA components

Amazon MWAA environments consist of the following four main components:

1. **Scheduler** — Parses and monitors all of your DAGs, and queues tasks for execution when a DAG's dependencies are met. Amazon MWAA deploys the scheduler as a AWS Fargate cluster with a minimum of 2 schedulers. You can increase the scheduler count up to five, depending on your workload. For more information about Amazon MWAA environment classes, see Amazon MWAA environment class.
2. **Workers** — One or more Fargate tasks that runs your scheduled tasks. The number of workers for your environment is determined by a range between a *minimum* and *maximum* number that you specify. Amazon MWAA starts auto-scaling workers when the number of queued and running tasks is more than your existing workers can handle. When running and queued tasks sum to zero for more than two minutes, Amazon MWAA scales back the number of workers to its minimum. For more information about how Amazon MWAA handles auto-scaling workers, see Amazon MWAA automatic scaling.
3. **Web server** — Runs the Apache Airflow web UI. You can configure the web server with private or public network access. In both cases, access to your Apache Airflow users is controlled by the access control policy you define in AWS Identity and Access Management (IAM). For more information about configuring IAM access policies for your environment, see Accessing an Amazon MWAA environment.
4. **Database** — Stores metadata about the Apache Airflow environment and your workflows, including DAG run history. The database is a single-tenant Aurora PostgreSQL database managed by AWS, and accessible to the *Scheduler* and *Workers*' Fargate containers via a privately-secured Amazon VPC endpoint.

Every Amazon MWAA environment also interacts with a set of AWS services to handle a variety of tasks, including storing and accessing DAGs and task dependencies, securing your data at rest, and logging and monitoring you environment. The following diagram demonstrates the different components of an Amazon MWAA environment.

Amazon MWAA Architecture



> **Note**
>
> The service Amazon VPC is not a shared VPC. Amazon MWAA creates an AWS owned VPC for every environment you create.

- **Amazon S3** — Amazon MWAA stores all of your workflow resources, such as DAGs, requirements, and plugin files in an Amazon S3 bucket. For more information about creating the bucket as part of environment creation, and uploading your Amazon MWAA resources, see Create an Amazon S3 bucket for Amazon MWAA in the *Amazon MWAA User Guide*.
- **Amazon SQS** — Amazon MWAA uses Amazon SQS for queueing your workflow tasks with a Celery executor.
- **Amazon ECR** — Amazon ECR hosts all Apache Airflow images. Amazon MWAA only supports AWS managed Apache Airflow images.

- **AWS KMS** — Amazon MWAA uses AWS KMS to ensure your data is secure at rest. By default, Amazon MWAA uses AWS managed AWS KMS keys, but you can configure your environment to use your own customer-managed AWS KMS key. For more information about using your own customer-managed AWS KMS key, see Customer managed keys for Data Encryption in the *Amazon MWAA User Guide*.
- **CloudWatch** — Amazon MWAA integrates with CloudWatch and delivers Apache Airflow logs and environment metrics to CloudWatch, allowing you to monitor your Amazon MWAA resources and troubleshoot issues.

Connectivity

Your Amazon MWAA environment needs access to all AWS services it integrates with. The Amazon MWAA execution role controls how access is granted to Amazon MWAA to connect to other AWS services on your behalf. For network connectivity, you can either provide public internet access to your Amazon VPC or create Amazon VPC endpoints. For more information on configuring Amazon VPC endpoints (AWS PrivateLink) for your environment, see Managing access to VPC endpoints on Amazon MWAA in the *Amazon MWAA User Guide*.

Amazon MWAA installs requirements on the scheduler and worker. If your requirements are sourced from a public PyPi repository, your environment needs connectivity to the internet to download the required libraries. For private environments, you can either use a private PyPi repository, or bundle the libraries in `.whl` files as custom plugins for your environment.

When you configure the Apache Airflow in private mode, the Apache Airflow UI can only be accessible to your Amazon VPC though Amazon VPC endpoints.

For more information about networking, see Networking in the *Amazon MWAA User Guide*.

Key considerations

Review the following topics before migrating to a new Amazon MWAA environment.

Topics

- Authentication
- Execution role

Authentication

Amazon MWAA uses AWS Identity and Access Management (IAM) to control access to the Apache Airflow UI. You must create and manage IAM policies that grant your Apache Airflow users permission to access the web server and manage DAGs. You can manage both authentication and authorization for Apache Airflow's default roles using IAM across different accounts.

You can further manage and restrict Apache Airflow users to access only a subset of your workflow DAGs by creating custom Airflow roles and mapping them to your IAM principals. For more information and a step-by-step tutorial, see Tutorial: Restricting an Amazon MWAA user's access to a subset of DAGs.

You can also configure federated identities to access Amazon MWAA. For more information see the following.

- **Amazon MWAA environment with public access** — Using Okta as an identity provider with Amazon MWAA on the *AWS Compute Blog*.
- **Amazon MWAA environment with private access** — Accessing a private Amazon MWAA environment using federated identities.

Execution role

Amazon MWAA uses an execution role that grants permissions to your environment to access other AWS services. You can provide your workflow with access to AWS services by adding the relevant permissions to the role. If you choose the default option to create a new execution role when you first create the environment, Amazon MWAA attaches the minimal permissions needed to the role, except in the case of CloudWatch Logs for which Amazon MWAA adds all log groups automatically.

Once the execution role is created, Amazon MWAA cannot manage its permission policies on your behalf. To update the execution role, you must edit the policy to add and remove permissions as needed. For example, you can integrate your Amazon MWAA environment with AWS Secrets Manager as a backend to securely store secrets and connection strings to use in your Apache Airflow workflows. To do so, attach the following permission policy to your environment's execution role.

```
{
    "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": "*"
        }
    ]
}
```

Integrating with other AWS services follows a similar pattern: you add the relevant permission policy to your Amazon MWAA execution role, granting permission to Amazon MWAA to access the service. For more information about managing the Amazon MWAA execution role, and to see additional examples, visit Amazon MWAA execution role in the *Amazon MWAA User Guide*.

Migrating to a new Amazon MWAA environment

The following topic describes the steps to migrate your existing Apache Airflow workload to a new Amazon MWAA environment. You can use the following steps to migrate from an older version of Amazon MWAA to a new version release, or migrate your self-managed Apache Airflow deployment to Amazon MWAA. This tutorial assumes you are migrating from an existing Apache Airflow v1.10.12 to a new Amazon MWAA running Apache Airflow v2.5.1, but you can use the same procedures to migrate from, or to different Apache Airflow versions.

Topics

- Prerequisites
- Step one: Create a new Amazon MWAA environment running the latest supported Apache Airflow version
- Step two: Migrate your workflow resources
- Step three: Exporting the metadata from your existing environment
- Step four: Importing the metadata to your new environment
- Next steps
- Related resources

Prerequisites

To be able to complete the steps and migrate your environment, you'll need the following:

- An Apache Airflow deployment. This can be a self-managed or existing Amazon MWAA environment.
- Docker installed for your local operating system.
- AWS Command Line Interface version 2 installed.

Step one: Create a new Amazon MWAA environment running the latest supported Apache Airflow version

You can create an environment using the detailed steps in Getting started with Amazon MWAA in the *Amazon MWAA User Guide*, or by using an AWS CloudFormation template. If you're migrating

from an existing Amazon MWAA environment, and used an AWS CloudFormation template to create your old environment, you can change the `AirflowVersion` property to specify the new version.

```
MwaaEnvironment:
    Type: AWS::MWAA::Environment
    DependsOn: MwaaExecutionPolicy
    Properties:
      Name: !Sub "${AWS::StackName}-MwaaEnvironment"
      SourceBucketArn: !GetAtt EnvironmentBucket.Arn
      ExecutionRoleArn: !GetAtt MwaaExecutionRole.Arn
      AirflowVersion: 2.5.1
      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
```

Alternatively, if migrating from an existing Amazon MWAA environment, you can copy the following Python script that uses the AWS SDK for Python (Boto3) to clone your environment. You can also download the script.

Python Script

```
# This Python file uses the following encoding: utf-8
'''
Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
SPDX-License-Identifier: MIT-0

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 __future__ import print_function
import argparse
import json
import socket
import time
import re
import sys
from datetime import timedelta
from datetime import datetime
import boto3
from botocore.exceptions import ClientError, ProfileNotFound
from boto3.session import Session
ENV_NAME = ""
REGION = ""

def verify_boto3(boto3_current_version):
    '''
    check if boto3 version is valid, must be 1.17.80 and up
    return true if all dependenceis are valid, false otherwise
    '''
    valid_starting_version = '1.17.80'
    if boto3_current_version == valid_starting_version:
        return True
```

```
    ver1 = boto3_current_version.split('.')
    ver2 = valid_starting_version.split('.')
    for i in range(max(len(ver1), len(ver2))):
        num1 = int(ver1[i]) if i < len(ver1) else 0
        num2 = int(ver2[i]) if i < len(ver2) else 0
        if num1 > num2:
            return True
        elif num1 < num2:
            return False
    return False


def get_account_id(env_info):
    '''
    Given the environment metadata, fetch the account id from the
    environment ARN
    '''
    return env_info['Arn'].split(":")[4]


def validate_envname(env_name):
    '''
    verify environment name doesn't have path to files or unexpected input
    '''
    if re.match(r"^[a-zA-Z][0-9a-zA-Z-_]*$", env_name):
        return env_name
    raise argparse.ArgumentTypeError("%s is an invalid environment name value" %
 env_name)


def validation_region(input_region):
    '''
    verify environment name doesn't have path to files or unexpected input
    REGION: example is us-east-1
    '''
    session = Session()
    mwaa_regions = session.get_available_regions('mwaa')
    if input_region in mwaa_regions:
        return input_region
    raise argparse.ArgumentTypeError("%s is an invalid REGION value" % input_region)


def validation_profile(profile_name):
    '''
```

```
    verify profile name doesn't have path to files or unexpected input
    '''
    if re.match(r"^[a-zA-Z0-9]*$", profile_name):
        return profile_name
    raise argparse.ArgumentTypeError("%s is an invalid profile name value" %
 profile_name)

def validation_version(version_name):
    '''
    verify profile name doesn't have path to files or unexpected input
    '''
    if re.match(r"[1-2].\d.\d", version_name):
        return version_name
    raise argparse.ArgumentTypeError("%s is an invalid version name value" %
 version_name)

def validation_execution_role(execution_role_arn):
    '''
    verify profile name doesn't have path to files or unexpected input
    '''
    if re.match(r'(?i)\b((?:[a-z][\w-]+:(?:/{1,3}|[a-z0-9%])|www\d{0,3}[.]|[a-z0-9.
\-]+[.][a-z]{2,4}/)(?:[^\s()<>]+|\(([^\s()<>]+|(\([^\s()<>]+\)))*\))+(?:\(([^\s()<>]+|
(\([^\s()<>]+\)))*\)|[^\s`!()\[\]{};:\'".,<>?«»“”‘’]))', execution_role_arn):
        return execution_role_arn
    raise argparse.ArgumentTypeError("%s is an invalid execution role ARN" %
 execution_role_arn)

def create_new_env(env):
    '''
    method to duplicate env
    '''
    mwaa = boto3.client('mwaa', region_name=REGION)

    print('Source Environment')
    print(env)
    if (env['AirflowVersion']=="1.10.12") and (VERSION=="2.2.2"):
        if env['AirflowConfigurationOptions']
['secrets.backend']=='airflow.contrib.secrets.aws_secrets_manager.SecretsManagerBackend':
            print('swapping',env['AirflowConfigurationOptions']['secrets.backend'])
            env['AirflowConfigurationOptions']
['secrets.backend']='airflow.providers.amazon.aws.secrets.secrets_manager.SecretsManagerBackend
    env['LoggingConfiguration']['DagProcessingLogs'].pop('CloudWatchLogGroupArn')
    env['LoggingConfiguration']['SchedulerLogs'].pop('CloudWatchLogGroupArn')
    env['LoggingConfiguration']['TaskLogs'].pop('CloudWatchLogGroupArn')
```

```
    env['LoggingConfiguration']['WebserverLogs'].pop('CloudWatchLogGroupArn')
    env['LoggingConfiguration']['WorkerLogs'].pop('CloudWatchLogGroupArn')
    env['AirflowVersion']=VERSION
    env['ExecutionRoleArn']=EXECUTION_ROLE_ARN
    env['Name']=ENV_NAME_NEW
    env.pop('Arn')
    env.pop('CreatedAt')
    env.pop('LastUpdate')
    env.pop('ServiceRoleArn')
    env.pop('Status')
    env.pop('WebserverUrl')
    if not env['Tags']:
        env.pop('Tags')
    print('Destination Environment')
    print(env)

    return mwaa.create_environment(**env)

def get_mwaa_env(input_env_name):

    # https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/
mwaa.html#MWAA.Client.get_environment
    mwaa = boto3.client('mwaa', region_name=REGION)
    environment = mwaa.get_environment(
        Name=input_env_name
    )['Environment']

    return environment

def print_err_msg(c_err):
    '''short method to handle printing an error message if there is one'''
    print('Error Message: {}'.format(c_err.response['Error']['Message']))
    print('Request ID: {}'.format(c_err.response['ResponseMetadata']['RequestId']))
    print('Http code: {}'.format(c_err.response['ResponseMetadata']['HTTPStatusCode']))

#
# Main
#
# Usage:
# python3 clone_environment.py --envname MySourceEnv --envnamenew MyDestEnv --region
 us-west-2 --execution_role AmazonMWAA-MyDestEnv-ExecutionRole --version 2.2.2
#
# based on https://github.com/awslabs/aws-support-tools/blob/master/MWAA/verify_env/
verify_env.py
```

```
#

if __name__ == '__main__':
    if sys.version_info[0] < 3:
        print("python2 detected, please use python3. Will try to run anyway")
    if not verify_boto3(boto3.__version__):
        print("boto3 version ", boto3.__version__, "is not valid for this script. Need
 1.17.80 or higher")
        print("please run pip install boto3 --upgrade --user")
        sys.exit(1)
    parser = argparse.ArgumentParser()
    parser.add_argument('--envname', type=validate_envname, required=True, help="name
 of the source MWAA environment")
    parser.add_argument('--region', type=validation_region,
 default=boto3.session.Session().region_name,
                        required=False, help="region, Ex: us-east-1")
    parser.add_argument('--profile', type=validation_profile, default=None,
                        required=False, help="AWS CLI profile, Ex: dev")
    parser.add_argument('--version', type=validation_version, default="2.2.2",
                        required=False, help="Airflow destination version, Ex: 2.2.2")
    parser.add_argument('--execution_role', type=validation_execution_role,
 default=None,
                        required=True, help="New environment execution role ARN, Ex:
 arn:aws:iam::112233445566:role/service-role/AmazonMWAA-MyEnvironment-ExecutionRole")
    parser.add_argument('--envnamenew', type=validate_envname, required=True,
 help="name of the destination MWAA environment")

    args, _ = parser.parse_known_args()
    ENV_NAME = args.envname
    REGION = args.region
    PROFILE = args.profile
    VERSION = args.version
    EXECUTION_ROLE_ARN = args.execution_role
    ENV_NAME_NEW = args.envnamenew

    try:
        print("PROFILE",PROFILE)
        if PROFILE:
            boto3.setup_default_session(profile_name=PROFILE)
        env = get_mwaa_env(ENV_NAME)
        response = create_new_env(env)
        print(response)
    except ClientError as client_error:
        if client_error.response['Error']['Code'] == 'LimitExceededException':
```

```
            print_err_msg(client_error)
            print('please retry the script')
        elif client_error.response['Error']['Code'] in ['AccessDeniedException',
'NotAuthorized']:
            print_err_msg(client_error)
            print('please verify permissions used have permissions documented in
readme')
        elif client_error.response['Error']['Code'] == 'InternalFailure':
            print_err_msg(client_error)
            print('please retry the script')
        else:
            print_err_msg(client_error)
    except ProfileNotFound as profile_not_found:
        print('profile', PROFILE, 'does not exist, please doublecheck the profile
name')
    except IndexError as error:
        print("Error:", error)
```

Step two: Migrate your workflow resources

Apache Airflow v2 is a major version release. If you are migrating from Apache Airflow v1, you must prepare your workflow resources and verify the changes you make to your DAGs, requirements, and plugins. To do so, we recommend configuring a *bridge* version of Apache Airflow on your local operating system using Docker and the Amazon MWAA local runner. The Amazon MWAA local runner provides a command line interface (CLI) utility that replicates an Amazon MWAA environment locally.

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

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 the `v1.10.15` branch of the aws-mwaa-local-runner repository. Apache Airflow released v1.10.15 as a *bridge release* to assist in migrating to Apache Airflow v2, and although Amazon MWAA does not support v1.10.15, you can use the Amazon MWAA local runner to test your resources.

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. Using Apache Airflow running locally, follow the steps described in Upgrading from 1.10 to 2 in the Apache Airflow documentation website.
    a. To update your `requirements.txt`, follow the best practices we recommend in Managing Python dependencies, in the *Amazon MWAA User Guide*.
    b. If you have bundled your custom operators and sensors with your plugins for your existing Apache Airflow v1.10.12 environment, move them to your DAG folder. For more information on module management best practices for Apache Airflow v2+, see Module Management in the Apache Airflow documentation website.
5. After you have made the required changes to your workflow resources, checkout the `v2.5.1` branch of the aws-mwaa-local-runner repository, and test your updated workflow DAGs, requirements, and custom plugins locally. If you're migrating to a different Apache Airflow version, you can use the appropriate local runner branch for your version, instead.
6. After you have successfully tested your workflow resources, copy your DAGs, `requirements.txt`, and plugins to the Amazon S3 bucket you configured with your new Amazon MWAA environment.

Step three: Exporting the metadata from your existing environment

Apache Airflow metadata tables such as `dag`, `dag_tag`, and `dag_code` automatically populate when you copy the updated DAG files to your environment's Amazon S3 bucket and the scheduler parses them. Permission related tables also populate automatically based on your IAM execution role permission. You do not need to migrate them.

You can migrate data related to DAG history, `variable`, `slot_pool`, `sla_miss`, and if needed, `xcom`, `job`, and `log` tables. Task instance log is stored in the CloudWatch Logs under the `airflow-`*`{environment_name}`* log group. If you want to see the task instance logs for older runs, those logs must be copied over to the new environment log group. We recommend that you move only a few days worth of logs in order to reduce associated costs.

If you're migrating from an existing Amazon MWAA environment, there is no direct access to the metadata database. You must run a DAG to export the metadata from your existing Amazon MWAA

environment to an Amazon S3 bucket of your choice. The following steps can also be used to export Apache Airflow metadata if you're migrating from a self-managed environment.

After the data is exported, you can then run a DAG in your new environment to import the data. During the export and the import process, all other DAGs are paused.

To export the metadata from your existing environment

1. Create an Amazon S3 bucket using the AWS CLI to store the exported data. Replace the `UUID` and `region` with your information.

```
$ aws s3api create-bucket \
    --bucket mwaa-migration-{UUID}\
    --region {region}
```

> **Note**
>
> If you are migrating sensitive data, such as connections you store in variables, we recommend that you enable default encryption for the Amazon S3 bucket.

2.

> **Note**
>
> Does not apply to migration from a self-managed environment.

Modify the execution role of the existing environment and add the following policy to grant write access to the bucket you created in step one.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject*"
            ],
            "Resource": [
                "arn:aws:s3:::mwaa-migration-{UUID}/*"
            ]
        }
```

```
    ]
}
```

3. Clone the amazon-mwaa-examples repository, and navigate to the `metadata-migration` subdirectory for your migration scenario.

```
$ git clone https://github.com/aws-samples/amazon-mwaa-examples.git
$ cd amazon-mwaa-examples/usecases/metadata-migration/existing-version-new-version/
```

4. In `export_data.py`, replace the string value for S3_BUCKET with the Amazon S3 bucket you created to store exported metadata.

```
S3_BUCKET = 'mwaa-migration-{UUID}'
```

5. Locate the `requirements.txt` file in the `metadata-migration` directory. If you already have a requirements file for your existing environment, add the additional requirements specified in `requirements.txt` to your file. If you do not have an existing requirements file, you can simply use the one provided in the `metadata-migration` directory.
6. Copy `export_data.py` to the DAG directory of the Amazon S3 bucket associated with your existing environment. If migrating from a self-managed environment, copy `export_data.py` to your `/dags` folder.
7. Copy your updated `requirements.txt` to the Amazon S3 bucket associated with your existing environment, then edit the environment to specify the new `requirements.txt` version.
8. After the environment is updated, access the Apache Airflow UI, unpause the db_export DAG, and trigger the workflow to run.
9. Verify that the metadata is exported to `data/migration/existing-version_to_new-version/export/` in the `mwaa-migration-{UUID}` Amazon S3 bucket, with each table in it's own dedicated file.

Step four: Importing the metadata to your new environment

To import the metadata to your new environment

1. In `import_data.py`, replace the string values for the following with your information.
   - For migration from an existing Amazon MWAA environment:

```
S3_BUCKET = 'mwaa-migration-{UUID}'
OLD_ENV_NAME='{old_environment_name}'
NEW_ENV_NAME='{new_environment_name}'
TI_LOG_MAX_DAYS = {number_of_days}
```

MAX_DAYS controls how many days worth of log files the workflow copies over to the new environment.

- For migration from a self-managed environment:

```
S3_BUCKET = 'mwaa-migration-{UUID}'
NEW_ENV_NAME='{new_environment_name}'
```

2. (Optional) `import_data.py` copies only failed task logs. If you want to copy all task logs, modify the `getDagTasks` function, and remove `ti.state = 'failed'` as shown in the following code snippet.

```
def getDagTasks():
    session = settings.Session()
    dagTasks = session.execute(f"select distinct ti.dag_id, ti.task_id,
 date(r.execution_date) as ed \
        from task_instance ti, dag_run r where r.execution_date > current_date -
 {TI_LOG_MAX_DAYS} and \
            ti.dag_id=r.dag_id and ti.run_id = r.run_id order by ti.dag_id,
 date(r.execution_date);").fetchall()
    return dagTasks
```

3. Modify the execution role of your new environment and add the following policy. The permission policy allows Amazon MWAA to read from the Amazon S3 bucket where you exported the Apache Airflow metadata, and to copy task instance logs from existing log groups. Replace all placeholders with your information.

> **Note**
>
> If you are migrating from a self-managed environment, you must remove CloudWatch Logs related permissions from the policy.

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:GetLogEvents",
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:{region}:{account_number}:log-
group:airflow-{old_environment_name}*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::mwaa-migration-{UUID}",
                "arn:aws:s3:::mwaa-migration-{UUID}/*"
            ]
        }
    ]
}
```

4. Copy `import_data.py` to the DAG directory of the Amazon S3 bucket associated with your new environment, then access the Apache Airflow UI to unpause the `db_import` DAG and trigger the workflow. The new DAG will appear in the Apache Airflow UI in a few minutes.
5. After the DAG run completes, verify that your DAG run history is copied over by accessing each individual DAG.

Next steps

- For more information about available Amazon MWAA environment classes and capabilities, see Amazon MWAA environment class in the *Amazon MWAA User Guide*.
- For more information about how Amazon MWAA handles autoscaling workers, see Amazon MWAA automatic scaling in the *Amazon MWAA User Guide*.
- For more information about the Amazon MWAA REST API, see the Amazon MWAA REST API.

Related resources

- Apache Airflow models (Apache Airflow Documentation) – Learn more about Apache Airflow metadata database models.

Migrating workloads from AWS Data Pipeline to Amazon MWAA

AWS launched the AWS Data Pipeline service in 2012. At that time, customers wanted a service that let them use a variety of compute options to move data between different data sources. As data transfer needs changed over time, so have the solutions to those needs. You now have the option to choose the solution that most closely meets your business requirements. You can migrate your workloads to any of the following AWS services:

- Use Amazon Managed Workflows for Apache Airflow (Amazon MWAA) to manage workflow orchestration for Apache Airflow.
- Use Step Functions to orchestrate workflows between multiple AWS services.
- Use AWS Glue to run and orchestrate Apache Spark applications.

The option you choose depends on your current workload on AWS Data Pipeline. This topic explains how to migrate from AWS Data Pipeline to Amazon MWAA.

Topics

- Choosing Amazon MWAA
- Architecture and concept mapping
- Example implementations
- Pricing comparison
- Related resources

Choosing Amazon MWAA

Amazon Managed Workflows for Apache Airflow (Amazon MWAA) is a managed orchestration service for Apache Airflow that lets you setup and operate end-to-end 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 the Python programming language to create workflows without having to manage the underlying infrastructure for scalability, availability, and security. Amazon MWAA automatically scales its workflow capacity to meet your needs, and is integrated with AWS security services to help provide you with fast and secure access to your data.

The following highlights some of the benefits of migrating from AWS Data Pipeline to Amazon MWAA:

- **Enhanced scalability and performance** – Amazon MWAA provides a flexible and scalable framework for defining and executing workflows. This allows users to handle large and complex workflows with ease, and take advantage of features such as dynamic task scheduling, data-driven workflows and parallelism.
- **Improved monitoring and logging** – Amazon MWAA integrates with Amazon CloudWatch to enhance monitoring and logging of your workflows. Amazon MWAA automatically sends system metrics and logs to CloudWatch. This means you can track the progress and performance of your workflows in real-time, and identify any issues that arise.
- **Better integrations with AWS services and third-party software** – Amazon MWAA integrates with a variety of other AWS services, such as Amazon S3, AWS Glue, and Amazon Redshift, as well as third-party software such as DBT, Snowflake, and Databricks. This lets you process, and transfer, data across different environments and services.
- **Open-source data pipeline tool** – Amazon MWAA leverages the same open-source Apache Airflow product you are familiar with. Apache Airflow is a purpose-built tool designed to handle all aspects of data pipeline management, including ingestion, processing, transferring, integrity testing, quality checks, and ensuring data lineage.
- **Modern and flexible architecture** – Amazon MWAA leverages containerization and cloud-native, serverless technologies. This means for more flexibility and portability, as well as easier deployment and management of your workflow environments.

Architecture and concept mapping

AWS Data Pipeline and Amazon MWAA have different architectures and components, which can affect the migration process and the way workflows are defined and executed. This section overviews architecture and components for both services, and highlights some of the key differences.

Both AWS Data Pipeline and Amazon MWAA are fully managed services. When you migrate your workloads to Amazon MWAA you might need to learn new concepts to model your existing workflows using Apache Airflow. However, you will not need to manage infrastructure, patch workers, and manage operating system updates.

The following table associates key concepts in AWS Data Pipeline with those in Amazon MWAA. Use this information as a starting point to design a migration plan.

Concept	AWS Data Pipeline	Amazon MWAA
Pipeline definition	AWS Data Pipeline uses JSON-based configuration file that defines the workflow.	Amazon MWAA uses Python-based Directed Acyclic Graphs (DAGs) that define the workflow.
Pipeline execution environment	Workflows run on Amazon EC2 instances. AWS Data Pipeline provisions and manages these instances on your behalf.	Amazon MWAA uses Amazon ECS containerized environments to run tasks.
Pipeline components	*Activities* are processing tasks that run as part of the workflow.	Operators (Tasks) are the fundamental processing units of a workflow.
	Preconditions contain conditional statements that must be true before an activity can run.	Sensors (Tasks) represent conditional statements that can wait for a resource or task to be completed before running.
	A *resource* in AWS Data Pipeline refers to the AWS compute resource that performs the work that a pipeline activity specifies. Amazon EC2 and Amazon EMR are two available resources.	Using tasks in a DAG, you can define a variety of compute resources, including Amazon ECS, Amazon EMR, and Amazon EKS. Amazon MWAA executes Python operations on workers that run on Amazon ECS.
Pipeline execution	AWS Data Pipeline supports scheduling runs with regular rate-based, and cron-based patterns.	Amazon MWAA supports scheduling with cron expressions and presets, as well as custom timetables.

Concept	AWS Data Pipeline	Amazon MWAA
	An *instances* refers to each run of the pipeline.	A DAG run refers to each run of an Apache Airflow workflow.
	An *attempt* refers to a retry of a failed operation.	Amazon MWAA supports retries that you define either at the DAG level, or at the task-level.

Example implementations

In many cases you will be able to re-use resources you are currently orchestrating with AWS Data Pipeline after migrating to Amazon MWAA. The following list contains example implementations using Amazon MWAA for the most common AWS Data Pipeline use-cases.

- Running an Amazon EMR job (AWS workshop)
- Creating a custom plugin for Apache Hive and Hadoop (*Amazon MWAA User Guide*)
- Copying data from S3 to Redshift (AWS workshop)
- Executing a shell script on a remote Amazon ECS instance (*Amazon MWAA User Guide*)
- Orchestrating hybrid (on-prem) workflows (Blog post)

For additional tutorials and examples, see the following:

- Amazon MWAA tutorials
- Amazon MWAA code examples

Pricing comparison

Pricing for AWS Data Pipeline is based on the number of pipelines, as well as how much you use each pipeline. Activities that you run more than once a day (high frequency) cost $1 per month per activity. Activities that you run once a day or less (low frequency) cost $0.60 per month per activity. Inactive Pipelines are priced at $1 per pipeline. For more information, see the AWS Data Pipeline pricing page.

Pricing for Amazon MWAA is based on the duration of time that your managed Apache Airflow environment exists, and any additional auto scaling required to provide more workers, or scheduler capacity. You pay for your Amazon MWAA environment usage on an hourly basis (billed at one-second resolution), with varying fees depending on the size of the environment. Amazon MWAA auto-scales the number of workers based on your environment configuration. AWS calculates the cost of additional workers separately. For more information on the hourly cost of using various Amazon MWAA environment sizes, see the Amazon MWAA pricing page.

Related resources

For more information and best practices for using Amazon MWAA, see the following resources:

- The Amazon MWAA API reference
- Monitoring dashboards and alarms on Amazon MWAA
- Performance tuning for Apache Airflow on Amazon MWAA

Amazon MWAA Document History

The following table describes important additions to the Amazon MWAA migration guide, beginning in March 2022.

Change	Description	Date
New topic on migrating workloads from AWS Data Pipeline to Amazon MWAA	Added new information and guidance on migrating existing workloads from AWS Data Pipeline to Amazon MWAA. Use this informati on to help you design a migration plan. • *Migrating workloads from AWS Data Pipeline to Amazon MWAA*	April 14, 2023
Amazon MWAA Migration Guide launch	Amazon MWAA now offers detailed guidance on migrating to a new Amazon MWAA environment. The steps described in the Amazon MWAA Migration Guide apply to mgirating from an existing Amazon MWAA environment, or from a self-managed Apache Airflow deployment. • About the Amazon MWAA migration guide	March 7, 2022