aws

Amazon MWAA Serverless User Guide

Amazon Managed Workflows for Apache Airflow Serverless

Amazon Managed Workflows for Apache Airflow Serverless: Amazon MWAA Serverless User Guide

Table of Contents

What is Amazon MWAA Serverless?

Amazon MWAA Serverless is a deployment option for MWAA that eliminates the operational overhead of managing Apache Airflow environments while providing cost-effective serverless scaling. This solution simplifies managing operational scale, optimizing costs, and handling access control.

With Amazon MWAA Serverless, teams can concentrate on developing their workflow logic instead of worrying about capacity management. The service automatically provides resources for both scheduled and on-demand workflow execution, with a cost model based on the actual compute time during task execution.

Amazon MWAA Serverless also features an enhanced security architecture leveraging AWS Identity and Access Management (IAM). Individual workflows can be assigned specific IAM permissions with tasks run in your chosen Virtual Private Cloud (VPC), enabling precise security controls without requiring separate Airflow environments. This design reduces security administration overhead while strengthening overall security measures.

> **Note**
>
> Amazon MWAA Serverless uses Apache Airflow v3 with Python 3.12.

Topics

Key features

The following are key features of Amazon MWAA Serverless:

- **Serverless architecture**: Provides an Amazon MWAA deployment option that eliminates the need to provision, configure, and tune Apache Airflow infrastructure.
- **Automatic scaling**: Dynamically scales compute resources to meet your workflow demands without manual intervention.
- **Workflow Isolation**: Each workflow runs with its own execution role and worker, providing enhanced security and preventing cross-workflow interference.
- **Enhanced security**: Workflow-specific IAM permissions for least-privilege access (compared to shared permissions in provisioned Amazon MWAA). Additionally, you can choose to run workflows in your VPC, where you can control the security for each workflow.
- **Cost optimization**: You only pay for actual workflow run time, which removes costs associated with idle resources. There are no upfront costs and no minimum provisioning is required.
- **Flexible workflow format**: Use YAML-based definition files to define your workflows.
- **AWS integration**: Use built-in access to other AWS services through open source Apache Airflow operators.
- **Workflow versioning**: Built-in versioning system allows you to manage workflow evolution and rollback to previous versions when needed.

Amazon MWAA Serverless versus traditional Amazon MWAA

Understanding when to choose Amazon MWAA Serverless over Amazon MWAA helps you select the right orchestration approach for your use case.

Category	Amazon MWAA Serverless	Traditional Amazon MWAA
Infrastructure management	Fully managed - no infrastru cture to configure or maintain	Managed Airflow environme nt with configurable capacity
Scaling	Automatic scaling based on workflow demand	Fixed always-on environment capacity , from micro to 2XL sizes, with automatic scaling of Airflow workers

Category	Amazon MWAA Serverless	Traditional Amazon MWAA
Cost model	Pay-per-use: charged only when workflows run	Hourly rate for environment, schedulers, web servers, and additional workers
Workflow isolation	Each workflow has its own execution role and access permissions. Tasks each run in isolated compute assuming the workflow's execution role.	Each MWAA environment provides the same access to all tasks. Airflow's role-base d authentication supports UI-level level isolation within an environment
Startup time	Each task provisions its own compute before execution	Environment always running, fast execution of tasks when worker capacity exists
Workflow definition	YAML workflows definitions using the DAG factory format, supporting AWS operators, with option to convert Python definitions	Python DAG support with AWS and custom operators
Airflow UI access	No direct Airflow UI access (monitoring via logs)	Airflow web interface access

When to use Amazon MWAA Serverless

Choose Amazon MWAA Serverless when you want to focus on workflow logic rather than infrastructure management. Amazon MWAA Serverless is particularly well-suited for:

- **Variable or unpredictable workloads**: Workflows that run sporadically or have varying resource requirements benefit from automatic scaling and pay-per-use pricing.
- **Cost-sensitive environments**: Development, testing, or production workloads where you want to minimize costs by avoiding idle infrastructure charges.
- **Multi-team organizations**: Teams that need workflow isolation for security or compliance reasons, with each workflow running under its own execution role.

- **AWS-native data pipelines**: Workflows primarily using AWS services like S3, Glue, EMR, Redshift, and SageMaker through built-in operators.
- **Simplified operations**: Organizations that want to reduce operational overhead by eliminating Apache Airflow environment management.

Consider provisioned Amazon MWAA when you need:

- Custom Python operators or complex DAG logic
- Direct access to the Airflow web interface
- Consistently high-volume workloads that benefit from always-on infrastructure
- Specific Airflow plugins or configurations not supported in the serverless model

How Amazon MWAA Serverless works

Amazon MWAA Serverless uses a multi-tenant architecture that automatically manages infrastructure while maintaining strong isolation between workflows. Here's how it works:

- **Workflow submission**: You submit YAML-based DAG definitions through AWS CLI, or API.
- **Automatic scheduling**: Amazon MWAA Serverless uses EventBridge Scheduler for reliable, timezone-aware scheduling of your workflows.
- **Resource allocation**: When a workflow runs, Amazon MWAA Serverless automatically provisions the necessary compute resources.
- **Isolated execution**: Each workflow runs with its own execution role and isolated compute environment, ensuring security and preventing interference.
- **Automatic cleanup**: Resources are automatically released when workflows complete, ensuring you only pay for actual usage.

This architecture provides the benefits of Apache Airflow workflow orchestration without the operational complexity of managing Airflow infrastructure.

Accessing Amazon MWAA Serverless

You can access Amazon MWAA Serverless through multiple interfaces:

- **AWS Management Console**: Monitor workflows through the AWS Management Console.

- **AWS CLI**: Use the AWS CLI for programmatic workflow management and automation.
- **API**: Integrate Amazon MWAA Serverless into your applications using the REST API for workflow lifecycle management.
- **SDKs**: Use AWS SDKs in your preferred programming language to build applications that manage workflows.

Regions

Amazon MWAA Serverless is available in the following AWS Regions.

- Europe (Ireland): eu-west-1
- Europe (London): eu-west-2
- Europe (Paris): eu-west-3
- Europe (Frankfurt): eu-central-1
- Europe (Stockholm): eu-north-1
- US East (N. Virginia): us-east-1
- US East (Ohio): us-east-2
- US West (Oregon): us-west-2
- South America (São Paulo): sa-east-1
- Asia Pacific (Tokyo): ap-northeast-1
- Asia Pacific (Seoul): ap-northeast-2
- Asia Pacific (Mumbai): ap-south-1
- Asia Pacific (Singapore): ap-southeast-1
- Asia Pacific (Sydney): ap-southeast-2
- Canada (Central): ca-central-1

Are you a first-time Amazon MWAA Serverless user?

If you are a first-time user of Amazon MWAA Serverless, we recommend that you begin by reading the following sections:

- Prerequisites for using Amazon MWAA Serverless

- Get started with Amazon MWAA Serverless
- Workflows

Key concepts

Understanding Amazon MWAA Serverless concepts helps you design, deploy, and manage your workflow orchestration solutions. This section explains the core concepts and how they relate to traditional Apache Airflow terminology.

- **Workflow**: The Amazon MWAA Serverless resource that represents your orchestration logic. A workflow is created through the Amazon MWAA Serverless API, or CLI. It contains metadata about scheduling, execution roles, and the orchestrated data processing steps (tasks).
- **Workflow Definition**: The Apache Airflow concept that defines the structure and dependencies of your tasks. While creating your Workflow, you provide your workflow definition file in Amazon S3. You can use the Python to YAML DAG converter to convert existing Python based DAG to YAML definitions.
- **Task**: An individual unit of work within a workflow. Each task represents a specific operation that runs on a worker.
- **Operator**: An Operator is a template for a predefined task.

Topics

- Workflow isolation
- Workflow versioning
- Execution model
- YAML support
- Monitoring

Workflow isolation

One of the key benefits of Amazon MWAA Serverless is workflow isolation, which provides security and operational benefits:

- **Execution role isolation**: Each workflow runs with its own IAM execution role, ensuring that workflows can only access the resources they're explicitly granted permission to use.

- **Compute isolation**: Workflows run on dedicated compute resources that are provisioned when the workflow starts and released when it completes.
- **Network isolation**: If you choose to specify a VPC, each workflow's tasks run in isolated network environments with their own security group configurations. VPCs are optional with Amazon MWAA Serverless.

This isolation model contrasts with traditional Amazon MWAA where all workflows share the same Airflow environment and execution context.

Workflow versioning

Amazon MWAA Serverless automatically manages workflow versions to help you track changes and enable rollbacks:

- **Automatic versioning**: Each time you update a workflow, Amazon MWAA Serverless creates a new version while preserving previous versions.
- **Version identification**: Versions are identified by alphanumeric ID.
- **Immutable versions**: Once created, workflow versions cannot be modified, ensuring consistency and enabling reliable rollbacks.

This versioning system allows you to safely evolve your workflows while maintaining the ability to rollback to previous working versions if issues arise.

Execution model

Amazon MWAA Serverless uses a serverless execution model that differs significantly from traditional Airflow deployments:

On-demand provisioning

Resources are provisioned only when workflows need to run:

- **Workflow startup**: When a workflow is triggered (by schedule or on-demand), Amazon MWAA Serverless provisions the necessary compute resources.
- **Task execution**: Individual tasks run on isolated compute instances with the appropriate execution role and network configuration.

- **Automatic cleanup**: Resources are automatically released when the workflow completes, ensuring you only pay for actual usage.

Scheduling model

Amazon MWAA Serverless uses EventBridge Scheduler for reliable workflow scheduling:

- **Timezone support**: Schedules can be defined with specific timezones, ensuring workflows run at the correct local time regardless of AWS region.
- **Flexible scheduling**: Support for cron expressions, rate expressions, and one-time schedules.
- **Reliable delivery**: EventBridge Scheduler provides built-in retry logic and dead letter queue support for failed workflow triggers.

YAML support

Amazon MWAA Serverless uses YAML based workflow definitions that provide a declarative approach to workflow authoring. You can use the Python to YAML converter tool to convert exiting Python based DAGs to YAML definition.

- **Declarative syntax**: Define your workflow structure, dependencies, and task configurations using YAML syntax.
- **Version control-friendly**: YAML files are easy to version control, review, and collaborate on.
- **Validation**: Amazon MWAA Serverless validates your YAML definitions before execution to catch configuration errors early.
- **Template support**: Use Jinja2 templating within your YAML definition for dynamic configuration.

Example YAML structure:

YAML

```
dag_id: my_data_pipeline
description: "Process daily data from S3 to Redshift"
schedule_interval: "0 2 * * *"  # Daily at 2 AM
start_date: "2024-01-01"

tasks:
```

```
extract_data:
operator: S3ListOperator
bucket: my-data-bucket
prefix: daily/

transform_data:
operator: GlueJobOperator
job_name: transform-daily-data
depends_on: [extract_data]

load_data:
operator: RedshiftSQLOperator
sql: "COPY target_table FROM 's3://processed-data/'"
depends_on: [transform_data]
```

Monitoring

You can observe your Amazon MWAA Serverless resources using AWS native services:

- **CloudWatch logs**: Access your workflow and task logs in CloudWatch for analysis and troubleshooting.
- **CloudTrail integration**: API calls and workflow management actions are logged in CloudTrail for audit and compliance.
- **Console**: View workflow status using the Amazon MWAA Serverless console.

Unlike traditional Amazon MWAA, you don't have direct access to the Apache Airflow web interface. Instead, you can build your custom monitoring and observability using AWS native tools.

Apache Airflow Parameter Support

Amazon MWAA Serverless leverages Apache Airflow as its workflow orchestration engine and supports YAML workflow definitions that are build using allowlisted Supported operators. To provide the serverless experience using these operators, Amazon MWAA Serverless supports a limited set of Apache Airflow parameters. While building your workflow definitions, we recommend you to refer to the follwoing paramter support:

Topics

DAG level parameters that Amazon MWAA Serverless will validate

The following table lists the supported Apache Airflow DAG level parameters that are subject to validation by Amazon MWAA Serverless during a create or update of workflow.

Parameter	Validation Rule	Default Value
dag_id	Must be a valid, non-empty string	
`schedule`	Must be a valid CRON expression format	
`start_date`	Must be in the future	
`end_date`	Must be after or equal to `start_date`	
`max_active_runs`	Must be smaller than account limit. Please refer to Quotas for Amazon MWAA Serverless	16

Task level parameters that Amazon MWAA Serverless will validate

The following table lists the Apache Airflow Task level parameters that are are subject to validation by Amazon MWAA Serverless during a task execution.

Parameter	Validation Rule	Default Value
`task_id`	Must be a valid string	

Parameter	Validation Rule	Default Value
retries	Must be between 0 to 3	1
retry_delay	Must be between 0 to 300 seconds	300 seconds
execution_timeout	Maximum 3600 seconds	3600 seconds

DAG level paramters that are not supported by Amazon MWAA Serverless

The following table lists the Apache Airflow DAG level parameters that are not supported by Amazon MWAA Serverless and will be ignored.

Parameter		
dag_id		
template_searchpath		
template_undefined		
user_defined_macros		
user_defined_filters		
catchup		
access_control		
jinja_environment_kwargs		
render_template_as_native_obj		
tags		

Parameter		
owner_links		
auto_register		
fail_fast		
dag_display_name		
depends_on_past		
email_on_failure		
email_on_retry		
description		
max_consecutive_failed_dag_runs		
dagrun_timeout		
sla_miss_callback		
on_failure_callback		
on_success_callback		
is_paused_upon_creation		

Task level paratmers that are not suppored by Amazon MWAA Serverless

The following table lists the Apache Airflow Task level parameters that are not supported by Amazon MWAA Serverless and will be ignored.

Parameter		
email_on_retry		
email_on_failure		
retry_exponential_backoff		
depends_on_past		
ignore_first_depends_on_past		
wait_for_downstream		
priority_weight		
sla		
max_active_tis_per_dag		
max_active_tis_per_dagrun		
task_concurrency		
resources		
run_as_user		
executor_config		
doc		
doc_md		
doc_rst		
doc_json		

Parameter		
doc_yaml		
task_display_name		
logger_name		
allow_nested_operators		
inlets		
outlets		
map_index_template		
email		
owner		
max_retry_delay		
on_execute_callback		
on_failure_callback		
on_success_callback		
on_retry_callback		
on_skipped_callback		
wait_for_past_depends_before_skipping		
do_xcom_push		
multiple_outputs		
start_date		

Parameter		
end_date		
weight_rule		
queue		
pool		
pool_slots		
pre_execute		
post_execute		
trigger_rule		
executor		
task_group		

AWS base operator attributes

Parameter	Amazon MWAA	Amazon MWAA Serverless
aws_conn_id	Supported	Controlled by service
verify	Supported	Not supported
botocore_config	Supported	Not supported
region_name	Supported	Gets Region from environment

Template Variables, Filters and Macros

Amazon MWAA Serverless supports Jinja templates that come out of the box with Apache Airflow. Please refer below for the supported list of Jinja Variables, and Macros.

Topics

Supported Jinja template Variables

The following table lists the supported Vriabls available for Jinja templating in Amazon MWAA Serverless.

Variable	Type	Description
`{{ macros }}`		A reference to the macros package. See Macros below.
`{{ task_instance }}`	TaskInstance	The currently running TaskInstance.
`{{ ti }}`	TaskInstance	Same as {{ task_instance }}.
`{{ params }}`	dict[str, Any]	The user-defined params. This can be overridden by the mapping passed to trigger_dag -c if dag_run_conf_overrides_params is enabled in airflow.cfg.
`{{ ds }}`	str	The Dag run's logical date as YYYY-MM-DD. Same as {{ logical_date | ds }}.
`{{ ds_nodash }}`	str	Same as {{ logical_date | ds_nodash }}.

Variable	Type	Description
{{ ts }}	str	Same as {{ logical_date | ts }}. Example: 2018-01-01T00:00:00+00:00.
{{ ts_nodash }}	str	Same as {{ logical_date | ts_nodash }}. Example: 20180101T000000.
{{ macros }}		

Supported Macros

The following table lists the supported Macros that are supported in Amazon MWAA Serverless.

Macro	Type	
macros.datetime	The standard lib's datetime.datetime	
macros.timedelta	The standard lib's datetime.timedelta	
macros.dateutil	A reference to the dateutil package	
macros.time	The standard lib's time	
macros.uuid	The standard lib's uuid	
macros.random	The standard lib's random.random	
datetime_diff_for_humans	datetime_diff_for_humans(dt, since=None): Return a human-readable/approximate difference between datetimes	

Macro	Type	
	. When only one datetime is provided, the comparison will be based on now.	
`ds_add`	ds_add(ds, days): Add or subtract days from a YYYY-MM-DD.	
`ds_format`	ds_format(ds, input_format, output_format): Output datetime string in a given format.	
`random`	The standard lib's random.random	

Prerequisites for using Amazon MWAA Serverless

Learn about the prerequisites you need before using Amazon MWAA Serverless. These include account setup and permissions management. You must have an AWS account before completing these tasks.

Topics

Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call or text message and entering a verification code on the phone keypad.

   When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform tasks that require root user access.

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to https://aws.amazon.com/ and choosing **My Account**.

Create a user with administrative access

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

1. Sign in to the AWS Management Console as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

   For help signing in by using root user, see Signing in as the root user in the *AWS Sign-In User Guide*.

2. Turn on multi-factor authentication (MFA) for your root user.

   For instructions, see Enable a virtual MFA device for your AWS account root user (console) in the *IAM User Guide*.

Create a user with administrative access

1. Enable IAM Identity Center.

   For instructions, see Enabling AWS IAM Identity Center in the *AWS IAM Identity Center User Guide*.

2. In IAM Identity Center, grant administrative access to a user.

   For a tutorial about using the IAM Identity Center directory as your identity source, see Configure user access with the default IAM Identity Center directory in the *AWS IAM Identity Center User Guide*.

Sign in as the user with administrative access

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

  For help signing in using an IAM Identity Center user, see Signing in to the AWS access portal in the *AWS Sign-In User Guide*.

Assign access to additional users

1. In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

   For instructions, see Create a permission set in the *AWS IAM Identity Center User Guide*.

2. Assign users to a group, and then assign single sign-on access to the group.

   For instructions, see Add groups in the *AWS IAM Identity Center User Guide*.

Grant IAM permissions

We recommend that you use finer-grained policies. For policy setup, refer to How Amazon MWAA Serverless works with IAM. To learn more about access management, refer to Access management for AWS resources in the IAM User Guide. For example, you can use policy similar to following.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "MWAAServerlessFullAccess",
      "Effect": "Allow",
      "Action": [
        "airflow-serverless:*"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "MWAAServerlessSLRCreation",
      "Effect": "Allow",
      "Action": [
        "iam:CreateServiceLinkedRole"
      ],
      "Resource": [
        "*"
      ]
    },
```

```
    {
      "Sid": "MWAAServerlessServicePassroleAccess",
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "MWAAServerlessServiceS3ReadAccess",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        's3:GetObjectVersion'
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "MWAAServerlessServiceCWLogsAccess",
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:DescribeLogGroups"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "MWAAServerlessServiceKMSAccess",
      "Effect": "Allow",
      "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:GenerateDataKey",
        "kms:DescribeKey"
      ],
      "Resource": [
        "*"
      ]
```

```
    }
  ]
}
```

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*.

Grant programmatic access

Users need programmatic access if they want to interact with AWS outside of the AWS Management Console. The way to grant programmatic access depends on the type of user that's accessing AWS.

To grant users programmatic access, choose one of the following options.

Which user needs programmatic access?	To	By
IAM	(Recommended) Use console credentials as temporary credentials to sign programmatic requests to the	Following the instructions for the interface that you want to use.

Which user needs programmatic access?	To	By
	AWS CLI, AWS SDKs, or AWS APIs.	• For the AWS CLI, see Login for AWS local development in the *AWS Command Line Interface User Guide*. • For AWS SDKs, see Login for AWS local development in the *AWS SDKs and Tools Reference Guide*.
Workforce identity (Users managed in IAM Identity Center)	Use temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions for the interface that you want to use. • For the AWS CLI, see Configuring the AWS CLI to use AWS IAM Identity Center in the *AWS Command Line Interface User Guide*. • For AWS SDKs, tools, and AWS APIs, see IAM Identity Center authentication in the *AWS SDKs and Tools Reference Guide*.
IAM	Use temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions in Using temporary credentials with AWS resources in the *IAM User Guide*.

Which user needs programmatic access?	To	By
IAM	(Not recommended) Use long-term credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions for the interface that you want to use. • For the AWS CLI, see Authenticating using IAM user credentials in the *AWS Command Line Interface User Guide*. • For AWS SDKs and tools, see Authenticate using long-term credentials in the *AWS SDKs and Tools Reference Guide*. • For AWS APIs, see Managing access keys for IAM users in the *IAM User Guide*.

Install and configure the AWS CLI

If you want to use Amazon MWAA Serverless APIs with the AWS Command Line Interface (AWS CLI), use the following instructions to install the latest version.

> **Tip**
>
> If you prefer to use Amazon MWAA Serverless with SDKs, you can access AWS builder tools in the Builder Center. Tools are listed by programming language.

To set up the AWS CLI

1. To install the latest version of the AWS CLI for macOS, Linux, or Windows, refer to Installing or updating the latest version of the AWS CLI.

2. To configure the AWS CLI and secure setup of your access to AWS services, including Amazon MWAA Serverless, refer to Quick configuration with `aws configure`.
3. To verify the setup, enter the following command at the command prompt.

   CLI

   ```
   aws mwaa-serverless help
   ```

   AWS CLI commands use the default AWS Region from your configuration, unless you set it with a parameter or a profile. To set your AWS Region with a parameter, you can add the `--region` parameter to each command.

   To set your AWS Region with a profile, first add a named profile in the `~/.aws/config` file or the `%UserProfile%/.aws/config` file (for Microsoft Windows). Follow the steps in Named profiles for the AWS CLI. Next, set your AWS Region and other settings with a command similar to the one in the following example.

   CLI

   ```
   [profile mwaa-serverless]
   aws_access_key_id = ACCESS-KEY-ID-OF-IAM-USER
   aws_secret_access_key = SECRET-ACCESS-KEY-ID-OF-IAM-USER
   region = us-east-1
   output = text
   ```

Get started with Amazon MWAA Serverless

To create a workflow in Amazon MWAA Serverless you need an Amazon S3 Bucket and an Execution role. You can choose to provide optional KMS key, Amazon VPC and VPC Security groups.

- The **Amazon S3 bucket** stores your workflow files. Your Amazon S3 bucket must block all public access and have versioning enabled. When you upload workflow files, Amazon MWAA Serverless parses these files before scheduling tasks. To learn more, refer to Create an Amazon S3 bucket for Amazon MWAA Serverless.
- **Execution role** enables Amazon MWAA Serverless workflow tasks to access your AWS resources. To learn more, refer to Execution roles.
- (Optional) **KMS key** is used to encrypt your workflow data. If you do not provide a KMS key, Amazon MWAA Serverless will use a default key for encryption. To learn more, refer to Using customer-managed keys for encryption.
- (Optional) **Amazon VPC** to connect to your private resources. If you don't provide a VPC, Amazon MWAA Serverless uses a default VPC with internet connectivity. To learn more, refer to Create your Amazon VPC network.
- (Optional) **VPC security group** lets Amazon MWAA Serverless access other AWS resources in your VPC network. You can provide your security group information while creating a workflow. To learn more, refer to Security in your VPC on Amazon MWAA Serverless.

After you complete these steps, you're ready to use Amazon MWAA Serverless. Choose one of two paths:

- Initialize a workflow using the AWS CLI (you must first upload your YAML workflow file to your S3 bucket). For more information, refer to Manage workflows.
- Migrate a Python DAG to YAML using dag-converter. For more information, refer to Convert Python DAG to YAML definition.

Create an Amazon S3 bucket for Amazon MWAA Serverless

Learn how to create an Amazon S3 bucket to store your workflow files.

Before you begin, note that:

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

Create the bucket

To create an Amazon S3 bucket for your workflow:

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
   - Contain between 3 and 63 characters
   - 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 Serverless workflow.

   - 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 Serverless, or specify a Customer-managed key for encryption of your Amazon MWAA Serverless workflow.

        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 Serverless. The AWS KMS key you specify must also be used for encryption on your Amazon MWAA Serverless workflow.

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, refer to 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 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?.

- Learn how to create a Amazon VPC network for an Amazon MWAA Serverless workflow in Networking.

Execution roles

An execution role is an AWS Identity and Access Management (IAM) role with a permissions policy that grants Amazon MWAA Serverless permission to invoke the resources of other AWS services on your behalf. This can include resources such as your Amazon S3 bucket, AWS KMS key, and CloudWatch Logs. Amazon MWAA Serverless needs one execution role per workflow. This topic describes how to use and configure the execution role to allow Amazon MWAA Serverless to access other AWS resources that are required by the workflow.

Amazon MWAA Serverless workflows aquire permissions to use other AWS services from the execution role. You must grant following permissions to Amazon MWAA Serverless execution role to allow your workflos to use these AWS services:

- Amazon CloudWatch (CloudWatch) to send Amazon MWAA Serverless workflow task logs to customer provided log group.
- AWS Key Management Service (AWS KMS) for data encryption (using either an AWS-owned key or your Customer managed key).

> **Note**
>
> In order for your workflow's execution role to access arbitrary KMS keys, a KMS key in a third-party account must allow this cross-account access via its resource policy.
> After you choose an encryption option, you cannot change your selection for an existing workflow.

Create an execution role

You use the IAM console to create a new execution role. When you create a new execution role, do not reuse the name of a deleted execution role. Unique names can help prevent conflicts and ensure proper resource management.

To create a new execution role, follow these steps:

1. Open the IAM console (https://console.aws.amazon.com/iam/).

2. In the navigation pane, choose **Roles**.
3. Choose **Create role**.
4. For **Trusted entity type**, choose **AWS service**.
5. For **Service or use case**, choose **Amazon MWAA Serverless**. Then choose **Amazon MWAA Serverless workflow**.
6. Choose **Next**.
7. For Permissions policies, search for **your customer managed policy**.
8. Choose the check box to the left of the **policy**, then choose Next.
9. For Role Name, enter **role name**, then choose **Create role**.

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

Update an execution role

Amazon MWAA Serverless can't add or edit permission policies to an existing execution role. You must update your execution role with additional permission policies needed by your workflow when you update that workflow. For example, if your DAG requires access to AWS Glue, Amazon MWAA Serverless can't automatically detect these permissions are required by your workflow, 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.

To view the execution role and update the JSON policy for the role on the IAM console:

1. Open the IAM console.
2. Choose the execution role name to open the permissions policy.

3. Choose **Edit policy**.
4. Choose the **JSON** tab.
5. Update your JSON policy.
6. Choose **Review policy**.
7. Choose **Save changes**.

Assuming the execution role is already associated with your workflow, Amazon MWAA Serverless can start using the added permission policies immediately. This also means if you remove any required permissions from an execution role, your workflow might fail.

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 Amazon EC2.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:Describe*",
                "ec2:GetSecurityGroupsForVpc"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "elasticloadbalancing:Describe*",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:ListMetrics",
                "cloudwatch:GetMetricStatistics",
                "cloudwatch:Describe*"
```

```
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "autoscaling:Describe*",
            "Resource": "*"
        }
    ]
}
```

To attach a policy to your execution role:

1. Open the IAM console.
2. Choose your execution role.
3. Choose **Attach policies**.
4. Choose **Create policy**.
5. Choose **JSON**.
6. Paste the JSON policy.
7. Choose **Next: Tags**, **Next: Review**.
8. Enter a descriptive name (such as `SecretsManagerReadPolicy`) and a description for the policy.
9. Choose **Create policy**.

Sample JSON policies for an execution role

The sample permission policies in this section show the policy to create a new execution role that can be uses for your workflow. These policies contain Resource ARN placeholders for Apache Airflow log groups, an Amazon S3 bucket.

Sample policy for for Amazon S3 operations

The following example shows an execution role policy you can use for a S3 operations.

> **Note**
>
> CloudWatchLogsAccess and VPCAccess are required for all operations, while KMSAccess is optional.

JSON

```
{
"Version": "2012-10-17",
"Statement": [
    {
    "Sid": "S3OperationSpecificPermissions",
    "Effect": "Allow",
    "Action": [
        "s3:*"
    ],
    "Resource": "*"
    },
    {
    "Sid": "CloudWatchLogsAccess",
    "Effect": "Allow",
    "Action": [
        "logs:CreateLogStream",
        "logs:PutLogEvents"
    ],
    "Resource": "*"
    },
    {
    "Sid": "KmsAccess",
    "Effect": "Allow",
    "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:GenerateDataKey",
        "kms:DescribeKey"
    ],
    "Resource": "arn:aws:kms:us-east-1:111122223333:key/keyId"
    }
]
}
```

Next, you need to allow Amazon MWAA Serverless to assume this role in order to perform actions on your behalf. This can be done by adding the "airflow-serverless.amazonaws.com" service principal 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:

JSON

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

Workflows

To use Amazon MWAA Serverless for orchestration, first create a workflow as a YAML definition file. If you already have a Python DAG file, convert it to YAML with the DAG converter tool. For instructions, refer to Convert Python DAG to YAML definition.

> **Important**
>
> Each workflow needs an IAM role. This role defines what AWS resources the workflow can access during execution.

Amazon MWAA Serverless workflows can be run on-demand or scheduled. Amazon MWAA Serverless workflows can be one of the following three types:

- **Scheduled**: Your workflow is run on the schedule you define in the workflow definition. You can still run your workflow on-demand outside of this schedule. This is similar to an `unpaused` DAG in Apache Airflow.
- **Disabled**: A disabled workflow can not be run on schedule or on-demand. This is like a `paused` DAG in Apache Airflow.
- **Manual only**: Your workflow ignores any defined schedules and can be only run on-demand. This option is best for testing or temporarily disabling automatic runs.

> **Note**
>
> Unlike Apache Airflow, Amazon MWAA Serverless doesn't automatically stop executing workflow runs when a workflow is set to **Manual only** or **Disabled**. Amazon MWAA Serverless uses a separate API `StopWorkflowRun` call to stop all workflow runs.

Topics

Manage your Amazon MWAA Serverless workflows

An Amazon MWAA Serverless workflow can be created using CLI or APIs. After uploading the workflow definition YAML in an Amazon S3 bucket, create the workflow. If you already have a Python workflow (DAG file), you can migrate it to Amazon MWAA Serverless by converting it into a YAML file. For more information, refer to Convert Python DAG to YAML definition.

To create your workflow using CLI, use `create-workflow`

CLI

```
aws airflow-serverless create-workflow --name my-workflow-name --definition-s3-
location'{"Bucket": "my-bucket-name", "ObjectKey": "my-yaml-file"}'
```

To run your workflow, use `start-workflow-run` and provide the `workflow-arn`. This command starts a new workflow execution.

CLI

```
aws airflow-serverless get-workflow --workflow-arn arn:aws:airflow-serverless:us-
east-1:111122223333:workflow/workflow-name
```

The following table contains all the actions you can perform on an Amazon MWAA Serverless workflow:

Action	API
Create workflow (the YAML file must already be in your Amazon S3 bucket).	`CreateWorkflow`
Delete a workflow and all its versions. You must stop a workflow before you can delete it.	`DeleteWorkflow` .

Action	API
Retrieve detailed information about a workflow.	GetWorkflow
List all workflows.	ListWorkflows
List all the verisions of a specified workflow.	ListWorkflowVersions
Start a workflow run.	StartWorkflowRun
Stop a workflow run.	StopWorkflowRun
Edit the configuration settings of a workflow.	UpdateWorkflow
List tags for a workflow.	ListTagsForResource
Tag a workflow.	TagResource
Untag a workflow.	UntagResource

Workflow states

When you create a workflow with Amazon MWAA Serverless, it can have a `READY` or a `DELETING` state. When you run a workflow, it can adopt the following states.

State	Description
`STARTING`	The workflow is ready to run.
`QUEUED`	The workflow is waiting in the queue to run.
`RUNNING`	The workflow is in progress.
`SUCCESS`	The workflow completed successfully.
`FAILED`	The workflow failed.
`TIMEOUT`	The workflow timed out before it completed.

State	Description
STOPPING	The workflow run is being stopped.
STOPPED	The workflow is stopped and no resources are running.

Convert Python DAG to YAML definition

Amazon MWAA Serverless provides a Python to YAML DAG converter tool that helps convert you Python based DAG to YAML based definition that is ready to create a Amazon MWAA Serverless workflow. Refer to the following steps to convert your Python based DAGs to YAML based definition using AWS CLI:

CLI

1. Install the library (dag-converter):

```
pip install python-to-yaml-dag-converter-mwaa-serverless
```

2. (Optional) If your Python DAG files are in S3, copy them locally:

```
aws s3 cp s3://source-bucket/dags/source.py/ my-local-folder/source.py
```

3. Run following command. This tool only converts Python based DAG files and provides an output file *dag_id*.yml

```
dag-converter convert my-local-folder/source.py --output /output_yaml/
destination
```

> **Important**
>
> A single Python DAG file can potentially create multiple YAML definitions workflows.

4. Move the YAML workflow to an Amazon S3 location.

```
aws s3 cp /output_yaml/dag_id.yml s3://destination-bucket/dags/ddag_id.yml
```

5. Refer to Workflows for steps to create Amazon MWAA Serverless workflows

Tag a workflow

You can assign tags to each Amazon MWAA Serverless workflow to help you manage resources. This section provides an overview of the tag functions and shows you how to create tags.

What is a tag?

A tag is a label that you assign to an AWS resource. Each tag consists of a key and a value, both of which you define. Tags enable you to categorize your AWS resources by attributes such as purpose, owner, and workflow. When you have many resources of the same type, you can quickly identify a specific resource based on the tags you've assigned to it. For example, you can define a set of tags for your Amazon MWAA Serverless applications to help you track each application's owner and stack level. We recommend that you use consistent set of tag keys for each resource type.

Tags are not automatically assigned to your resources. After you add a tag to a resource, you can modify a tag's value or remove the tag from the resource at any time. Tags do not have any semantic meaning to Amazon MWAA Serverless and are interpreted strictly as strings of characters. If you add a tag that has the same key as an existing tag on that resource, the new value overwrites the earlier value.

You can control which user in your AWS account has permissions to manage tags through IAM For tag-based access control policy examples, see ABAC with Amazon MWAA Serverless.

Tagging your resources

Workflows are the only Amazon MWAA Serverless resources that you can tag. If you're using the Amazon MWAA Serverless API, the AWS CLI, or an AWS SDK, you can apply tags to new workflows

using the `tags` parameter on the relevant API action. You can apply tags to existing workflows using the `TagResource` API action.

You can use some resource-creating actions to specify tags for a resource when the resource is created. In this case, if tags cannot be applied while the resource is being created, the resource fails to be created. This mechanism ensures that resources you intended to tag on creation are either created with specified tags or not created at all.

The following table describes the Amazon MWAA Serverless resources that can be tagged.

Resource	Supports tags	Supports tag propagation	Supports tagging on creation (Amazon MWAA Serverless API, AWS CLI, and AWS SDK)	API for creation (tags can be added during creation)
Workflow	Yes	No. Tags associated with a workflow do not propagate to tasks within the workflow.	Yes	`CreateWorkflow`

Tagging limitations

The following basic limitations apply to tags:

- Each workflow can have a maximum of 50 user-created tags.
- For each workflow, each tag key must be unique, and each tag key can have only one value.
- The maximum key length is 128 Unicode characters in UTF-8.
- The maximum value length is 256 Unicode characters in UTF-8.
- Allowed characters are letters, numbers, spaces representable in UTF-8, and the following characters: _ . : / = + - @.
- A tag key cannot be an empty string. A tag value can be an empty string, but not null.

- Tag keys and values are case sensitive.
- Do not use `AWS:` or any upper or lowercase combination of such as a prefix for either keys or values. These are reserved only for AWS use.

Working with tags using the AWS CLI and the Amazon MWAA Serverless API

Use the following AWS CLI commands or Amazon MWAA Serverless API operations to add, update, list, and delete the tags for your workflows.

Resource	Supports tags	Supports tag propagation
Add or overwrite one or more tags	`tag-resource`	`TagResource`
List tags for a resource	`list-tags-for-resource`	`ListTagsForResource`
Delete one or more tags	`untag-resource`	`UntagResource`

The following examples show how to tag or untag resources using the AWS CLI.

Tag an existing workflow

The following command tags an existing workflow.

CLI

```
aws mwaa-serverless tag-resource --resource-arn workflow-arn --tags your-tag-key=your-tag-value
```

Untag an existing workflow

The following command deletes a tag from an existing workflow.

CLI

```
aws mwaa-serverless untag-resource --resource-arn workflow-arn --tag-keys your-tag-key
```

List tags for a workflow

The following command lists the tags associated with an existing workflow.

CLI

```
aws mwaa-serverless list-tags-for-resource --resource-arn workflow-arn
```

Security in Amazon MWAA Serverless

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from data centers and network architectures that are built to meet the requirements of the most security-sensitive organizations. Security is a shared responsibility between AWS and you. 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 Serverless, refer to 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 MWAA Serverless. The following topics show you how to configure Amazon MWAA Serverless 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 Serverless resources.

Topics

Data protection in Amazon MWAA Serverless

The AWS shared responsibility model applies to data protection in Amazon MWAA Serverless for Apache Airflow Serverless. As described in this model, AWS is responsible for protecting the

global infrastructure that runs all of the AWS Cloud. ou are responsible for maintaining control over your content that is hosted on this infrastructure. You are also responsible for 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 users 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 require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with CloudTrail. For information about using CloudTrail trails to capture AWS activities, see Working with CloudTrail trails in the CloudTrail User Guide.
- 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 sensitive 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 text fields such as a **Name** field. This includes when you work with Amazon MWAA Serverless or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text 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.

Data encryption

The following topics describe how Amazon MWAA Serverless protects your data at rest, and in transit. Use this information to learn how Amazon MWAA Serverless integrates with AWS KMS to encrypt data at rest, and how data is encrypted using Transport Layer Security (TLS) protocol in transit.

Encryption at rest

Amazon MWAA Serverless by default encrypts data at rest and uses the KMS key you provide while creating the workflow. If you do not provide a key, Amazon MWAA Serverless encrypts this data

using service keys. 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 workflow.

To use a customer-managed KMS key, you must attach the required policy statement for CloudWatch access to your key policy and make the call to API using the KMS policy. Amazon MWAA Serverless attaches two grant to your customer-managed KMS key to encrypt data at rest.

> **Note**
>
> You pay for the storage and use of AWS-owned and customer-managed KMS keys on Amazon MWAA Serverless. For more information, refer to AWS KMS Pricing.

Encryption artifacts

You specify the encryption artifacts used for at rest encryption by providing a customer-managed key when you create your Amazon MWAA Serverless workflow. Amazon MWAA Serverless adds the grants needed to your specified key.

CloudWatch Logs: If you to use a customer-managed KMS key, you must add a key policy to your key to allow CloudWatch Logs to use your key. If you do not provide a customer-managed KMS key, CloudWatch Logs are encrypted using Server Side Encryption with service owned key.

Encryption in transit

Data in transit is data that may be intercepted as it travels the network.

Transport Layer Security (TLS) encrypts the Amazon MWAA Serverless objects in transit between your Amazon MWAA Serverless workflow components and other AWS services that integrate with Amazon MWAA Serverless, such as Amazon S3. For more information about Amazon S3 encryption, refer to Protecting data using encryption.

Key management

You can configure AWS KMS to automatically rotate your KMS keys. This rotates your keys once a year while saving old keys indefinitely so that your data can still be decrypted. For more information, refer to Rotate AWS KMS keys.

Using customer-managed keys for encryption

You can optionally provide a Customer managed key for data encryption on your workflow. You must create the customer managed KMS key in the same Region as your Amazon MWAA Serverless workflow instance and your Amazon S3 bucket where you store resources for your workflows.

Key support

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

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.

When you create an Amazon MWAA Serverless workflow and specify a customer managed KMS key, Amazon MWAA Serverless attaches the grant to your customer managed KMS key.

Amazon MWAA Serverless creates, and attaches, additional grants to a specified KMS key on your behalf. This includes policies to retire a grant if you delete your workflow, to use your customer managed KMS key for Client-Side Encryption (CSE), and for execution role that needs to access secrets protected by your customer managed key in Secrets Manager.

Grants

Amazon MWAA Serverless 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 Serverless) to perform actions defined in the policy.

Grant 1: Create data plane resources

JSON

```
{
 "GranteePrincipal": "airflow-serverless.us-east-1.amazonaws.com",
 "RetiringPrincipal": "airflow-serverless.us-east-1.amazonaws.com",
 "IssuingAccount": "AWS Internal",
 "Operations": [
  "Decrypt",
  "Encrypt",
  "GenerateDataKey",
  "RetireGrant"
 ],
 "Constraints": {
  "EncryptionContextSubset": {
   "aws:airflow-serverless:workflow-arn": "arn:aws:airflow-serverless:eu-
west-1:111122223333:workflow/workflow_name"
  }
    }
}
```

Grant 2: For workflow execution

JSON

```
{
 "GranteePrincipal": "airflow-serverless.us-east-1.amazonaws.com",
 "RetiringPrincipal": "airflow-serverless.us-east-1.amazonaws.com",
 "IssuingAccount": "AWS Internal",
 "Operations": [
  "Decrypt",
  "Encrypt",
```

```
  "GenerateDataKey",
  "RetireGrant"
  ],
 "Constraints": {
  "EncryptionContextSubset": {
   "aws:airflow-serverless:workflow-arn": "arn:aws:airflow-serverless:eu-
west-1:111122223333:workflow/workflow_name"
   }
  }
}
```

Sample polices

Execution role: To provide permissions to the Execution role permission for KMS keys, attach following add-on policies to the execution role.

Note that you can not use kms:viaService in the execution role since the execution role credentials are directly used to make encrypt/decrypt calls while execution of the workflow.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "kms:Decrypt",
            "kms:GenerateDataKey"
        ],
        "Resource": "arn:aws:kms:eu-west-1:111122223333:key/keyId",
        "Condition": {
            "ArnLike": {
                "kms:EncryptionContext:aws:airflow-serverless:workflow-arn":
 "arn:aws:airflow-serverless:us-east-1:*:*"
            }
        }
    }
  ]
}
```

Trust relationship for the workflow execution role:

JSON

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

Add KMS resource policy for CloudWatch logs

JSON

```
{
    "Version":"2012-10-17",
    "Id": "key-default-1",
    "Statement": [
        {
            "Sid": "Enable IAM User Permissions",
            "Effect": "Allow",
            "Principal": {
            "AWS": "arn:aws:iam::111122223333:root"
            },
            "Action": "kms:*",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "logs.us-east-1.amazonaws.com"
            },
```

```
            "Action": [
                "kms:Encrypt",
                "kms:Decrypt",
                "kms:ReEncrypt*",
                "kms:GenerateDataKey*",
                "kms:Describe*"
            ],
            "Resource": "*",
            "Condition": {
                "ArnEquals": {
                    "kms:EncryptionContext:aws:logs:arn": "arn:aws:logs:us-
east-1:111122223333:log-group:log-group-name"
                },
                {
                }
            }
        }
    ]
}
```

Attaching key policies to a customer-managed key

If the customer managed KMS key you used for your Amazon MWAA Serverless workflow is not already configured to work with CloudWatch, you must update the key policy to allow for encrypted CloudWatch Logs. For more information, refer to the Encrypt log data in CloudWatch using AWS KMS.

The following example represents a key policy for CloudWatch Logs. Substitute the sample values provided for the region.

JSON

```
{
    "Effect": "Allow",
    "Principal": {
        "Service": "logs.us-east-1.amazonaws.com"
    },
    "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
```

```
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:Describe*"
    ],
    "Resource": "*",
    "Condition": {
        "ArnEquals": {
           "kms:EncryptionContext:aws:logs:arn": "arn:aws:logs:us-
east-1:111122223333:log-group:log-group-name"
        }

    }
}
```

You could add a condition on kms:viaService to scope down the usage by adding the following statement to the key policy.

JSON

```
{
            "Sid": "Enable Encrypt Decrypt with Context",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:caller and execution role"
            },
            "Action": [
                "kms:Encrypt",
                "kms:Decrypt",
                "kms:GenerateDataKey",
                "kms:CreateGrant"
            ],
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "kms:ViaService": "airflow-serverless.us-east-1.amazonaws.com"
                }
            }
}
```

API Policies

This section applies only if you are using a customer-managed KMS key. Use the following add-on policies on the role that is calling these APIs when using a customer-managed KMS key.

CreateWorkflow API:

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "airflow-serverless:CreateWorkflow",
            "Resource": "*"
        }
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:GenerateDataKey",
                "kms:CreateGrant"
            ],
            "Resource": "arn:aws:kms:us-east-1:111122223333:key/keyId",
            "Condition": {
                "ArnLike": {
                    "kms:EncryptionContext:aws:airflow-serverless:workflow-arn":
 "arn:aws:airflow-serverless:us-east-1:*:*"
                },
                "StringEquals": {
                    "kms:ViaService": "airflow-serverless.us-east-1.amazonaws.com"
                }
            }
        }
    ]
}
```

GetWorkflow and **GetWorkflowRun**

Note: only permission needed is `kms:Decrypt`. You can scope down to `kms:viaService`

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "airflow-serverless:GetWorkflow",
                "airflow-serverless:GetWorkflowRun"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "kms:Decrypt",
            "Resource": "arn:aws:kms:us-east-1:111122223333:key/key_id",
            "Condition": {
                "ArnLike": {
                    "kms:EncryptionContext:aws:airflow-serverless:workflow-arn":
 "arn:aws:airflow-serverless:us-east-1:*:*"
                },
                "StringEquals": {
                    "kms:ViaService": "airflow-serverless.us-east-1.amazonaws.com"
                }
            }
        }
    ]
}
```

StartWorkflowRun

JSON

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

```
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "airflow-serverless:StartWorkflowRun",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "kms:Decrypt",
        "kms:GenerateDataKey"
      ],
      "Resource": "arn:aws:kms:us-east-1:111122223333:key/key_id",
      "Condition": {
        "ArnLike": {
          "kms:EncryptionContext:aws:airflow-serverless:workflow-arn":
 "arn:aws:airflow-serverless:us-east-1:*:*"
        },
        "StringEquals": {
            "kms:ViaService": "airflow-serverless.us-east-1.amazonaws.com"
        }
      }
    }
  ]
}
```

Identity and access management for Amazon MWAA Serverless

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 MWAA Serverless resources. IAM is an AWS service that you can use with no additional charge.

Amazon MWAA Serverless supports distinct authorization permissions for each workflow in order to communicate with customer-managed services and data. Each workflow has its own IAM resource identifier, which is used to restrict workflow access to individual IAM principals. Each workflow also has an IAM role that it assumes in order to interact with AWS resources. A workflow IAM role can be passed through the Amazon MWAA Serverless API.

Amazon MWAA Serverless also supports user-level access control for each workflow. You can define which users can create, view, edit, delete, and revert each workflow or group of workflows.

Topics

Audience

How you use AWS Identity and Access Management (IAM) differs based on your role:

- **Service user** - request permissions from your administrator if you cannot access features (see Troubleshooting Amazon MWAA Serverless identity and access)
- **Service administrator** - determine user access and submit permission requests (see How Amazon MWAA Serverless works with IAM)
- **IAM administrator** - write policies to manage access (see Identity-based policy examples for Amazon MWAA Serverless)

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be authenticated as the AWS account root user, an IAM user, or by assuming an IAM role.

You can sign in as a federated identity using credentials from an identity source like AWS IAM Identity Center (IAM Identity Center), single sign-on authentication, or Google/Facebook credentials. For more information about signing in, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

For programmatic access, AWS provides an SDK and CLI to cryptographically sign requests. For more information, see AWS Signature Version 4 for API requests in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity called the AWS account *root user* that has complete access to all AWS services and resources. We strongly recommend that you don't use the root user for everyday tasks. For tasks that require root user credentials, see Tasks that require root user credentials in the *IAM User Guide*.

Federated identity

As a best practice, require human users to use federation with an identity provider to access AWS services using temporary credentials.

A *federated identity* is a user from your enterprise directory, web identity provider, or Directory Service that accesses AWS services using credentials from an identity source. Federated identities assume roles that provide temporary credentials.

For centralized access management, we recommend AWS IAM Identity Center. For more information, see What is IAM Identity Center? in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An *IAM user* is an identity with specific permissions for a single person or application. We recommend using temporary credentials instead of IAM users with long-term credentials. For more information, see Require human users to use federation with an identity provider to access AWS using temporary credentials in the *IAM User Guide*.

An *IAM group* specifies a collection of IAM users and makes permissions easier to manage for large sets of users. For more information, see Use cases for IAM users in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity with specific permissions that provides temporary credentials. You can assume a role by switching from a user to an IAM role (console) or by calling an AWS CLI or AWS API operation. For more information, see Methods to assume a role in the *IAM User Guide*.

IAM roles are useful for federated user access, temporary IAM user permissions, cross-account access, cross-service access, and applications running on Amazon EC2. For more information, see Cross account resource access in IAM 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 defines permissions when associated with an identity or resource. AWS evaluates these policies when a principal makes a request. Most policies are stored in AWS as JSON documents. For more information about JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Using policies, administrators specify who has access to what by defining which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. An IAM administrator creates IAM policies and adds them to roles, which users can then assume. IAM policies define permissions regardless of the method used to perform the operation.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you attach to an identity (user, group, or role). These policies control what actions identities 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 *inline policies* (embedded directly into a single identity) or *managed policies* (standalone policies attached to multiple identities). To learn how to choose between managed and inline policies, 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 include 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. You must specify a principal in a resource-based policy.

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.

Other policy types

AWS supports additional policy types that can set the maximum permissions granted by more common policy types:

- **Permissions boundaries** – Set the maximum permissions that an identity-based policy can grant to an IAM entity. For more information, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – Specify the maximum permissions for an organization or organizational unit in AWS Organizations. For more information, see Service control policies in the *AWS Organizations User Guide*.
- **Resource control policies (RCPs)** – Set the maximum available permissions for resources in your accounts. For more information, see Resource control policies (RCPs) in the *AWS Organizations User Guide*.
- **Session policies** – Advanced policies passed as a parameter when creating a temporary session for a role or federated user. 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*.

How Amazon MWAA Serverless works with IAM

Before you use IAM to manage access to Amazon MWAA Serverless, learn what IAM features are available to use with Amazon MWAA Serverless.

IAM feature	Amazon MWAA Serverless support
Identity-based policies	Yes
Resource-based policies	No
Policy actions	Yes
Policy resources	Yes
Policy condition keys	No
ACLs	No
ABAC (tags in policies)	Yes

IAM feature	Amazon MWAA Serverless support
Temporary credentials	Yes
Principal permissions	Yes
Service roles	No
Service-linked roles	Yes

To get a high-level view of how Amazon MWAA Serverless and other AWS services work with most IAM features, refer to AWS services that work with IAM in the *IAM User Guide*.

Identity-based policies for Amazon MWAA Serverless

Supports identity-based policies: Yes

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*.

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. To learn about all of the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Identity-based policy examples for Amazon MWAA Serverless

To view examples of Amazon MWAA Serverless identity-based policies, see Identity-based policy examples for Amazon MWAA Serverless.

Resource-based policies within Amazon MWAA Serverless

Supports resource-based policies: No

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.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

Policy actions for Amazon MWAA Serverless

Supports policy actions: Yes

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. Include actions in a policy to grant permissions to perform the associated operation.

For a list of Amazon MWAA Serverless actions, refer to Actions Defined by Amazon MWAA Serverless in the *Service Authorization Reference*.

Policy actions in Amazon MWAA Serverless use the following prefix before the action:

```
airflow-serverless
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
  "airflow-serverless:CreateWorkflow",
  "airflow-serverless:GetWorkflow",
  "airflow-serverless:UpdateWorkflow",
  "airflow-serverless:DeleteWorkflow",
  "airflow-serverless:ListWorkflows"
]
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `Describe`, include the following action:

```
"Action": "airflow-serverless:List*"
```

> **Important**
>
> It is best practice to grant least privilege and use only the permissions required in a policy. Do **not** use wildcards to specify all of the actions for a service.

To view examples of Amazon MWAA Serverless identity-based policies, see Identity-based policy examples for Amazon MWAA Serverless.

Policy resources for Amazon MWAA Serverless

Supports policy resources: Yes

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 `Resource` JSON policy element specifies the object or objects to which the action applies. As a best practice, specify a resource using its Amazon Resource Name (ARN). For actions that don't support resource-level permissions, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

To see a list of Amazon MWAA Serverless resource types and their ARNs, refer to Resources Defined by Amazon MWAA Serverless in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, refer to Actions Defined by Amazon MWAA Serverless.

To view examples of Amazon MWAA Serverless identity-based policies, see Identity-based policy examples for Amazon MWAA Serverless.

Policy condition keys for Amazon MWAA Serverless

Supports service-specific policy condition keys: No

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 `Condition` element specifies when statements execute based on defined criteria. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

Amazon MWAA Serverless supports global context keys and does not have its own set of condition keys.

To view examples of Amazon MWAA Serverless identity-based policies, see Identity-based policy examples for Amazon MWAA Serverless.

ACLs in Amazon MWAA Serverless

Supports ACLs: No

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.

ABAC with Amazon MWAA Serverless

Supports ABAC (tags in policies): Yes

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes called tags. You can attach tags to IAM entities and AWS resources, then design ABAC policies to allow operations when the principal's tag matches the tag on the resource.

To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/`*`key-name`*, `aws:RequestTag/`*`key-name`*, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see Define permissions with ABAC authorization in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the *IAM User Guide*.

For more information about tagging Amazon MWAA Serverless resources, refer to Tag a workflow.

To view an example identity-based policy for limiting access to a resource based on the tags on that resource, refer to ADD LINK .

Using temporary credentials with Amazon MWAA Serverless

Supports temporary credentials: Yes

Temporary credentials provide short-term access to AWS resources and are automatically created when you use federation or switch roles. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM and AWS services that work with IAM in the *IAM User Guide*.

Cross-service principal permissions for Amazon MWAA Serverless

Supports forward access sessions (FAS): Yes

Forward access sessions (FAS) use the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. For policy details when making FAS requests, see Forward access sessions.

Service roles for Amazon MWAA Serverless

Supports service roles: No

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*.

> **Note**
>
> Changing the permissions for a service role can break service functionality. The only service role supported by Amazon MWAA Serverless is the execution role. Changing permissions for an execution role can affect the workflow that is associated with it.

Service-linked roles for Amazon MWAA Serverless

Supports service-linked roles: Yes

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.

For details about creating or managing Amazon MWAA Serverless service-linked roles, see Using service-linked roles for Amazon MWAA Serverless.

For details about creating or managing service-linked roles, refer to AWS services that work with IAM. Find a service in the table that includes a Yes in the **Service-linked role** column. Choose the **Yes** link to view the service-linked role documentation for that service.

Identity-based policy examples for Amazon MWAA Serverless

By default, users and roles don't have permission to create or modify Amazon MWAA Serverless resources. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see Create IAM policies (console) in the *IAM User Guide*.

For details about actions and resource types defined by Amazon MWAA Serverless, including the format of the ARNs for each of the resource types, see Actions, Resources, and Condition Keys for Amazon MWAA Serverless in the *Service Authorization Reference*.

Topics

Policy best practices

Identity-based policies determine whether someone can create, access, or delete Amazon MWAA Serverless 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 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*.

Allow 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": "*"
        }
    ]
}
```

Using service-linked roles for Amazon MWAA Serverless

Amazon MWAA Serverless uses an AWS Identity and Access Management (IAM) service-linked role named `AWSServiceRoleForAmazonMWAAServerless`. This service-linked role is an (IAM) role that's linked directly to Amazon MWAA Serverless. It's predefined by Amazon MWAA Serverless, and it includes all the permissions that Amazon MWAA Serverless requires to call other AWS services and monitor AWS resources on your behalf. Amazon MWAA Serverless uses this service-linked role in all the AWS Regions where Amazon MWAA Serverless is available.

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

You can delete the Amazon MWAA Serverless service-linked role only after you disable Amazon MWAA Serverless in all the Regions where it's enabled. This protects your Amazon MWAA Serverless resources because you can't inadvertently remove permissions to access them.

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.

Topics

Service-linked role permissions for Amazon MWAA Serverless

Amazon MWAA Serverless uses the service-linked role named `AWSServiceRoleForAmazonMWAAServerless`. It's a service-linked role required for Amazon MWAA Serverless to access your resources. This service-linked role allows Amazon MWAA Serverless Provides access to Amazon MWAA Serverless to manage networking for your workflows and access other AWS services on your behalf. The `AWSServiceRoleForAmazonMWAAServerless` service-linked role trusts the `airflow-serverless.amazonaws.com` service to assume the role.

The `AWSServiceRoleForAmazonMWAAServerless` service-linked role uses the managed policy AmazonMWAAServerlessServiceRolePolicy.

You must grant permissions to allow the `airflow-serverless.amazonaws.com` service principal to assume `AWSServiceRoleForAmazonMWAAServerless` service-linked role. Add the following trust policy:

JSON

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

```
      "Principal": {
        "Service": "airflow-serverless.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

You must configure permissions to allow your users, groups, or roles to create, edit, or delete a service-linked role. For more information, refer to Service-linked role permissions in the *IAM User Guide*.

Creating a service-linked role for Amazon MWAA Serverless

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

> **Important**
>
> This service-linked role can appear in your account if you completed an action in another service that uses the features supported by this role. To learn more, refer to A new role appeared in my AWS account.

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 CreateWorkflow, Amazon MWAA Serverless creates the service-linked role for you again.

You can also use the IAM console to create a service-linked role with the **MWAA-S - Networking** use case. In the AWS CLI or the AWS API, create a service-linked role with the `airflow-serverless.amazonaws.com` service name. For more information, refer to Creating a service-linked role in the *IAM User Guide*. If you delete this service-linked role, you can use this same process to create the role again.

Editing a service-linked role for Amazon MWAA Serverless

Amazon MWAA Serverless does not allow you to edit the AWSServiceRoleForAmazonMWAAServerless 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, refer to Editing a service-linked role in the *IAM User Guide*.

Deleting a service-linked role for Amazon MWAA Serverless

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. However, you must clean up the resources for your service-linked role before you can manually delete it.

When you delete a workflow, Amazon MWAA Serverless deletes all the associated resources it uses as a part of the service. However, you must wait before Amazon MWAA Serverless completes deleting your workflow, before attempting to delete the service-linked role. If you delete the service-linked role before Amazon MWAA Serverless deletes the workflow, Amazon MWAA Serverless might be unable to delete all of the workflow'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 AWSServiceRoleForAmazonMWAAServerless service-linked role. For more information, refer to Deleting a service-linked role in the *IAM User Guide*.

AWS managed policy

An AWS managed policy is a standalone policy that is created and administered by AWS. AWS managed policies are designed to provide permissions for many common use cases so that you can start assigning permissions to users, groups, and roles.

Keep in mind that AWS managed policies might not grant least-privilege permissions for your specific use cases because they're available for all AWS customers to use. We recommend that you reduce permissions further by defining customer managed policies that are specific to your use cases.

You cannot change the permissions defined in AWS managed policies. If AWS updates the permissions defined in an AWS managed policy, the update affects all principal identities (users, groups, and roles) that the policy is attached to. AWS is most likely to update an AWS managed policy when a new AWS service is launched or new API operations become available for existing services.

For more information, see AWS managed policies in the IAM User Guide.

AWS managed policy: AmazonMWAAServerlessServiceRolePolicy

The `AWSServiceRoleForAmazonMWAAServerless` service-linked role used by Amazon MWAA Serverless to perform actions on your behalf. You cannot attach this policy to your users, groups, or roles.

This policy grants administrative permissions that allow the service-linked role full access to perform tasks required to manage network configuration for your Amazon EC2 instances.

Permission details

This policy includes the following permissions:

- `ec2` - Allows users to perform actions to manage network configuration of EC2 such as Attach, Create, Delete etc.

To review the permissions for this policy, see AmazonMWAAServerlessServiceRolePolicy in the AWS Managed Policy Reference Guide.

Amazon MWAA Serverless updates to AWS managed policies

The following table provides details about updates to AWS managed policies for Amazon MWAA Serverless since this service began tracking these changes. For automatic alerts about updates to the policies, subscribe to the RSS feed on the Amazon MWAA Serverless document history page.

Change	Description	Date
Amazon MWAA Serverless update its service-linked role permission policy	AmazonMWAAServerlessServiceRolePolicy – Amazon MWAA updates the permission policy for its service-linked role to grant Amazon MWAA Serverless permission to perform actions on all Amazon MWAA Serverless-supported AWS resources	November 17, 2025
Amazon MWAA Serverless started tracking changes	Amazon MWAA Serverless started tracking changes for	November 17, 2025

Change	Description	Date
	its AWS-managed service-l inked role permission policy.	

Troubleshooting Amazon MWAA Serverless identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with Amazon MWAA Serverless and IAM.

Topics

I am not authorized to perform an action in Amazon MWAA Serverless

If you receive an error that you're not authorized to perform an action, your policies must be updated to allow you to perform the action.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a fictional *`my-example-widget`* resource but doesn't have the fictional `airflow-serverless:`*`GetWidget`* permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 airflow-serverless:GetWidget on resource: my-example-widget
```

In this case, the policy for the `mateojackson` user must be updated to allow access to the *`my-example-widget`* resource by using the `airflow-serverless:`*`GetWidget`* action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

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 Serverless.

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 Serverless. 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 Serverless 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 Serverless supports these features, see How Amazon MWAA Serverless 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*.

Compliance validation for Amazon MWAA Serverless

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. For more information about your compliance responsibility when using AWS services, see AWS Security Documentation.

Resilience in Amazon MWAA Serverless

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with 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, refer to AWS Global Infrastructure.

Infrastructure Security in Amazon MWAA Serverless

For information about AWS security services and how AWS protects infrastructure, see AWS Cloud Security. To design your workflows using the best practices for infrastructure security, see Infrastructure Protectiony in Security Pillar AWS Well-Architected Framework.

You use AWS published API calls to access Amazon MWAA Serverless through the network. Clients must support

- Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later.
- Clients must also support 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.

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 resource policies to limit the permissions that Amazon MWAA Serverless gives another service to 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-serverless:*:123456789012:*`.

If the `aws:SourceArn` value does not contain the account ID, such as an Amazon S3 bucket ARN, you must use both global condition context keys to limit permissions.

The value of `aws:SourceArn` must be a workflow.

The following example shows how you can use the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in Amazon MWAA Serverless to prevent the confused deputy problem.

JSON

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

```
      "Service": "airflow-serverless.amazonaws.com"
    },
    "Action": "airflow-serverless:ActionName",
    "Resource": [
      "arn:aws:airflow-serverless:::ResourceName/*"
    ],
    "Condition": {
      "ArnLike": {
        "aws:SourceArn": "arn:aws:airflow-serverless:*:123456789012:*"
      },
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      }
    }
  }
}
```

Security best practices on Amazon MWAA Serverless

Amazon MWAA Serverless 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 workflow, 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.

Security best practices in Apache Airflow

To implement security boundaries for your workflows:

- Store secrets in AWS Secrets Manager. While this will not prevent users who can write workflow definitions from reading secrets, it prevents them from modifying the secrets that your workflow uses.

Monitoring Amazon MWAA Serverless

Monitoring helps maintain the reliability, availability, and performance of Amazon MWAA Serverless and your other AWS solutions. Amazon MWAA Serverless uses CloudTrail and CloudWatch to capture different logs and metrics. You can provide an option loggorup while creating Amazon MWAA Serverless workflows. If you do not provide a loggoroup name, Amazon MWAA Serverless will create a new loggroup where all the logs for the workflows are published.

You can leverage a mix of AWS services such as AWS Lambda and EventBridge to build a custom CloudWatch dashboard to monitor the health of your workflows:

- *Amazon CloudWatch* monitors your AWS resources and and the applications you run on AWS in real time. You can collect and track metrics, create customized dashboards, and set alarms that notify you or take actions when a specified metric reaches a threshold that you specify. For example, you can have CloudWatch track CPU usage or other metrics of your Amazon EC2 instances and automatically launch new instances when needed. For more information, refer to the Amazon CloudWatch User Guide.
- *Amazon CloudWatch Logs* enables you to monitor, store, and access your log files from Amazon EC2 instances, CloudTrail, and other sources. CloudWatch Logs can monitor information in the log files and notify you when certain thresholds are met. You can also archive your log data in highly durable storage. For more information, refer to the Amazon CloudWatch Logs User Guide.
- *Amazon EventBridge* can be used to automate your AWS services and respond automatically to system events, such as application availability issues or resource changes. Events from AWS services are delivered to EventBridge in near real time. You can write simple rules to indicate which events are of interest to you and which automated actions to take when an event matches a rule. For more information, refer to Amazon EventBridge User Guide.
- *AWS CloudTrail* captures API calls and related events made by or on behalf of your AWS account and delivers the log files to an Amazon S3 bucket that you specify. You can identify which users and accounts called AWS, the source IP address from which the calls were made, and when the calls occurred. For more information, refer to the AWS CloudTrail User Guide.
- *AWS Lambda* enables you to run code without provisioning or managing servers. You pay only for the compute time that you consume—there's no charge when your code isn't running. You can run code for virtually any type of application or backend service—all with zero administration. Just upload your code and Lambda takes care of everything required to run and scale your code with high availability.

Topics

Observability with CloudWatch

You may view and manage your workflows on the Amazon MWAA Serverless Console however, for advanced observability Amazon MWAA Serverless Serverless provides integration with Amazon CloudWatch where tasks logs are stored in a log group provided by you. If you do not provide a log group name while creating a workflow, Amazon MWAA Serverless will create a new log group.

Amazon MWAA Serverless workflow execution status is returned via the `GetWorkflowRun` function. The results from function returns details for a particular workflow run. If there are errors in the workflow definition, they are returned under `RunDetail` in the `ErrorMessage` field as in the following example:

JSON

```
{
  "WorkflowVersion": "7bcd36ce4d42f5cf23bfee67a0f816c6",
  "RunId": "d58cxqdClpTVjeN",
  "RunType": "SCHEDULE",
  "RunDetail": {
    "ModifiedAt": "2025-11-03T08:02:47.625851+00:00",
    "ErrorMessage": "expected token ',', got 'create_test_table'",
    "TaskInstances": [],
    "RunState": "FAILED"
  }
}
```

Workflows that are properly defined, but whose tasks fail, will return `"ErrorMessage": "Workflow execution failed":`

JSON

```
{
  "WorkflowVersion": "0ad517eb5e33deca45a2514c0569079d",
  "RunId": "ABC123456789def",
  "RunType": "SCHEDULE",
  "RunDetail": {
    "StartedOn": "2025-11-03T13:12:09.904466+00:00",
    "CompletedOn": "2025-11-03T13:13:57.620605+00:00",
    "ModifiedAt": "2025-11-03T13:16:08.888182+00:00",
    "Duration": "107",
    "ErrorMessage": "Workflow execution failed",
    "TaskInstances": [
      "ex_5496697b-900d-4008-8d6f-5e43767d6e36_create_bucket_1"
    ],
    "RunState": "FAILED"
  }
}
```

Log groups created by Amazon MWAA Serverless are available in Amazon CloudWatch log group `/aws/mwaa-serverless/`*`workflow id`*`/` (where */`*`workflow id`* is the same string as the unique workflow id in the ARN of the workflow). For specific task log streams, list the tasks for the workflow run and then get each task's information. You can combine these operations into a single CLI command as shown below

CLI

```
aws mwaa-serverless list-task-instances
--workflow-arn arn:aws:airflow-serverless:us-east-2:111122223333:workflow/
simple_s3_test-abc1234def
--run-id ABC123456789def
--region us-east-2
--query 'TaskInstances[].TaskInstanceId'
--output text | xargs -n 1 -I {} aws mwaa-serverless get-task-instance
--workflow-arn arn:aws:airflow-serverless:us-east-2:111122223333:workflow/
simple_s3_test-abc1234def
--run-id ABC123456789def
--task-instance-id {}
--region us-east-2
--query '{Status: Status, StartedAt: StartedAt, LogStream: LogStream}'
```

The response of the above command will be similar to following when it runs successfully. When it fails, the status will be returned as “FAILED”

JSON

```
{
  "Status": "SUCCESS",
  "StartedAt": "2025-10-28T21:21:31.753447+00:00",
  "LogStream": "//aws/mwaa-serverless/simple_s3_test_3-abc1234def//
workflow_id=simple_s3_test-abc1234def/run_id=ABC123456789def/task_id=list_objects/
attempt=1.log"
}
```

Use the Amazon CloudWatch LogStream output to debug your workflow. For samples of creating detailed metrics and monitoring dashboard using AWS Lambda, Amazon CloudWatch, Amazon DynamoDB, and Amazon EventBridge, review the example in this GitHub repository.

Logging Amazon MWAA Serverless APIs with CloudTrail

Amazon MWAA Serverless is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in the Amazon MWAA Serverless. AWS CloudTrail captures API calls for Amazon MWAA Serverless as events including calls from Amazon MWAA Serverless console and code calls to the Amazon MWAA Serverless API operations. If you create a trail, you can enable continuous delivery of AWS CloudTrail events to an Amazon S3 bucket, including events for Amazon MWAA Serverless. If you don't configure a trail, you can still view the most recent events in the AWS CloudTrail console in Event history. Using the information collected by AWS CloudTrail, you can determine the request that was made to Amazon MWAA Serverless, the IP address it was made from, who made it, when it was made, and additional details.

To learn more about AWS CloudTrail, including how to configure and enable it, see the AWS CloudTrail User Guide.

Data encryption

AWS CloudTrailis enabled on your AWS account when you create it. AWS CloudTrail logs the activity taken by an IAM entity or an AWS service, such as , which is recorded as a CloudTrail event. You can view, search, and download the past 90 days of event history in the AWS CloudTrail console. AWS

CloudTrail captures all events on the Amazon MWAA Serverless console and all calls to Amazon MWAA Serverless APIs.

Topics

Creating a trail in CloudTrail

You need to create a trail to access an ongoing record of events in your AWS account, including events for Amazon MWAA Serverless. A trail enables AWS CloudTrail to deliver log files to an Amazon S3 bucket. If you don't create a trail, you can still access available event history in the AWS CloudTrail console. For example, using the information collected by AWS CloudTrail, you can determine the request that was made to Amazon MWAA Serverless, the IP address from which the request was made, who made the request, when it was made, and additional details. To learn more, refer to the Creating a trail for your AWS account.

Accessing events with CloudTrail Event History

You can troubleshoot operational and security incidents over the past 90 days in the AWS CloudTrail console by viewing event history. For example, you can access 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, refer to the Accessing Events with CloudTrail Event History.

1. Open the AWS CloudTrail console.
2. Choose **Event history**.
3. Select the events you want to view, and then choose **Compare event details**.

Accessing events with CloudTrail Event History

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 aren't listed in any specific order. The following example is a log entry for the `CreateWorkflow` action that is denied due to lacking permissions.

CLI


```
{
  "eventVersion": "1.11",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE",
    "arn": "arn:aws:iam::111122223333:role/mwaa-serverless-role",
    "accountId": "111122223333",
    "accessKeyId": "ASIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AIDA123456789EXAMPLE",
        "arn": "arn:aws:iam::111122223333:role/Admin",
        "accountId": "111122223333",
        "userName": "Admin"
      },
      "attributes": {
        "creationDate": "2025-11-03T23:30:24Z",
        "mfaAuthenticated": "false"
      }
    }
  },
  "eventTime": "2025-11-03T23:31:54Z",
  "eventSource": "airflow-serverless.amazonaws.com",
  "eventName": "CreateWorkflow",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "192.0.2.1",
  "userAgent": "aws-cli/2.28.17",
  "requestParameters": {
    "name": "my-mwaa-serverless-workflow",
    "clientToken": "c52de4a9-4f0f-4c87-af44-0fc9417e0765",
    "definitionS3Location": {
      "bucket": "amzn-s3-demo-bucket",
      "objectKey": "my-mwaa-serverless-workflow.yml"
    },
    "roleArn": "arn:aws:iam::111122223333:role/mwaa-serverless-workflow-role",
```

```
    "encryptionConfiguration": {
      "type": "CUSTOMER_MANAGED_KEY",
      "kmsKeyId": "arn:aws:kms:us-east-1:111122223333:key/37edb8d7-ff19-4456-9b65-
cbef9c61b53f"
    },
    "loggingConfiguration": {
      "logGroupName": "my-mwaa-serverless-workflow-log-group"
    },
    "triggerMode": "manual_only"
  },
  "responseElements": {
    "workflowArn": "arn:aws:airflow-serverless:us-east-1:111122223333:workflow/my-
mwaa-serverless-workflow-Tc2lQzso0N",
    "createdAt": "2025-11-03T23:31:54.406Z",
    "revisionId": "41f42323-f8cb-463b-a7d7-3d1a9d16cdea",
    "workflowVersion": "7a1bf53101df1c62969f81221c58c5f7"
  },
  "requestID": "14f37e54-e6f3-49f7-97fb-bd32b578c68d",
  "eventID": "a82b4d5f-74f6-4817-9591-066488a63859",
  "readOnly": "false",
  "resources": [
    {
      "accountId": "111122223333",
      "type": "AWS::MWAAServerless::Workflow",
      "ARN": "arn:aws:airflow-serverless:us-east-1:111122223333:workflow/my-mwaa-
serverless-workflow-Tc2lQzso0N"
    }
  ],
  "eventType": "AwsApiCall",
  "managementEvent": "true",
  "recipientAccountId": "111122223333",
  "eventCategory": "Management"
}
```

Networking

> **Note**
>
> Creating a VPC network is **optional** with Amazon MWAA Serverless.

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* (Amazon VPC network has access to the internet) or *private routing* (Amazon VPC network doees not have access to the internet).

To learn more about VPCs in AWS, refer to AWS PrivateLink concepts.

VPC support

The following table describes the types of Amazon VPCs Amazon MWAA Serverless supports.

Amazon VPC types	Supported	
An Amazon VPC owned by the customer that is attempting to create the workflow.	Yes	
A shared Amazon VPC owned by service to host customer tasks.	Yes	

VPC infrastructure overview

VPC 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 that are used by your workflow.

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 a workflow 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 groups **must** be part of the same VPC.
  - 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`; for IPv6, use `::/0`).
  - The security group **must** allow all traffic in the self-referencing rule. For example, (Recommended) Example all access self-referencing security group .
- **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.
  - **Minimum 2** and **Minimum 16** subnets are supported in Amazon MWAA Serverless.
  - These subnets **must** be private.
  - At least two subnets **must** be in different Availability Zones. For example, `us-east-1a`, `us-east-1b`. This allows Amazon MWAA Serverless to run your workflow tasks in your other availability zone, if one container fails.
  - The subnets **must** have a route table to a NAT device (gateway or instance).
  - The subnets **must not** route to an internet gateway.
  - Set `assignIpV6AddressOnCreation` to `true` for IPv6 subnets.
  - For IPv6 private subnets, you **must** have a connection to an egress-only internet gateway (EIGW).
- **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`; for IPv6, use `::/0`).
  - The NACL **must** have an outbound rule that allows all traffic (`0.0.0.0/0`; for IPv6, use `::/0`).

- **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 a workflow 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 groups **must** be part of the same VPC.
  - 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`; for IPv6, use `::/0`).
- **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.
  - **Minimum 2** and **Minimum 16** subnets are supported in Amazon MWAA Serverless.
  - These subnets **must** be private.
  - At least two subnets **must** be in different Availability Zones. For example, `us-east-1a`, `us-east-1b`. This allows Amazon MWAA Serverless to run your workflow tasks in your other availability zone, if one container fails.
  - The subnets **must** have a route table to your VPC endpoints.
  - The subnets **must** have a route table to an EIGW in order to download from the internet as part of a DAG.
  - 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`; for IPv6, use `::/0`).
  - The NACL **must** have an outbound rule that denies all traffic (`0.0.0.0/0`; for IPv6, use `::/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 that your workflow uses, and Apache Airflow VPC endpoints in the same AWS Region and Amazon VPC as your Amazon MWAA Serverless workflow.
  - A VPC endpoint for each AWS service that your workflow uses and VPC endpoints for Apache Airflow.
  - The VPC endpoints **must** have private DNS enabled.
  - The VPC endpoints **must** be associated to your workflow's two private subnets.
  - The VPC endpoints **must** be associated to your workflow's security group.
  - The VPC endpoint policy for each endpoint should be configured to allow access to AWS services used by the workflow. 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.

Create a VPC network

Creating a VPC network is **optional** with Amazon MWAA Serverless. This section describes the different options you can use if you choose to create a Amazon VPC network.

To learn how to manage access to your VPC, refer to Control access to VPC endpoints using endpoint policies.

Tip

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 CloudWatch and AWS KMS. For more information about configuring AWS PrivateLink for Amazon MWAA Serverless, refer to Access Amazon MWAA Serverless using an interface endpoint (AWS PrivateLink).

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:

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

Create your Amazon VPC network

You have the following options to create a Amazon VPC network:

- Create an Amazon VPC network with internet access
- Create an Amazon VPC network without internet access

Note

Amazon MWAA Serverless 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 Serverless in the US East (N. Virginia) region, you must explicitly assign the `AvailabilityZone` in the CloudFormation en(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
```

For more information about VPCs and networking, refer to Get started with AWS PrivateLink in the AWS PrivateLink User Guide.

Create an Amazon VPC network with internet access

The following 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`.

```
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-serverless-workflow --template-
body file://cfn-vpc-public-private.yaml
```

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

Create an Amazon VPC network without internet access

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

This option uses private routing without internet access. You can use this template for an Apache Airflow *Web server* with **Private network** access mode only. It creates the required VPC endpoints for the AWS services that are used by a workflow. For more information, refer to Attaching the required VPC endpoints in the Amazon MWAA User Guide.

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

```
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

  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-serverless-security-group"

  SecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: !Ref SecurityGroup
      IpProtocol: "-1"
      SourceSecurityGroupId: !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-serverless-private-workflow --
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, refer to Managing access to service-specific Amazon VPC endpoints on Amazon MWAA in the Amazon MWAA User Guide.

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

Security in your VPC on Amazon MWAA Serverless

Learn about the Amazon VPC components used to secure your Amazon MWAA Serverless workflow and the configurations needed for these components.

Contents

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 that you specify.

- Network traffic to and from a subnet can be controlled by ACLs. You only need one ACL, and the same ACL can be used on multiple workflows.
- Network traffic to and from an instance can be controlled by an Amazon VPC security group. You can use between one and five security groups per workflow.
- 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 is required. You can optionally attach a policy to the endpoint to further restrict access to specific resources that are relevant to the service for which the endpoint was created. For example, if you have a AWS KMS VPC endpoint, you can write a policy that restricts actions to certain AWS KMS keys.

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* (Amazon VPC network has access to the internet) or *private routing* (Amazon VPC network doees not have access to the internet).

Rule number	Type	Protocol	Port range	Source	Allow or 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-serverless-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		

test

(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-serverless-	

Type	Protocol	Port range	Source type	Source	
				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-us-east-1-starport-layer-bucket/*"]
    }
  ]
}
```

Access Amazon MWAA Serverless using an interface endpoint (AWS PrivateLink)

You can use AWS PrivateLink to create a private connection between your VPC and Amazon MWAA Serverless. You can access Amazon MWAA Serverless as if it were in your VPC, without the use of an internet gateway, NAT device, VPN connection, or Direct Connect connection. Instances in your VPC don't need public IP addresses to access Amazon MWAA Serverless.

You establish this private connection by creating an *interface endpoint*, powered by AWS PrivateLink. We create an endpoint network interface in each subnet that you enable for the interface endpoint. These are requester-managed network interfaces that serve as the entry point for traffic destined for Amazon MWAA Serverless.

For more information, refer to Access AWS services through AWS PrivateLink in the *AWS PrivateLink Guide*.

Considerations for Amazon MWAA Serverless

Before you set up an interface endpoint for Amazon MWAA Serverless, review Considerations in the *AWS PrivateLink Guide*.

Amazon MWAA Serverless supports making calls to all of its API actions through the interface endpoint.

VPC endpoint policies are supported for Amazon MWAA Serverless. By default, full access to Amazon MWAA Serverless is allowed through the interface endpoint. Alternatively, you can

associate a security group with the endpoint network interfaces to control traffic to Amazon MWAA Serverless through the interface endpoint.

Create an interface endpoint for Amazon MWAA Serverless

You can create an interface endpoint for Amazon MWAA Serverless using either the Amazon VPC console or the AWS Command Line Interface (AWS CLI). For more information, refer to Create an interface endpoint in the *AWS PrivateLink Guide*.

Create an interface endpoint for Amazon MWAA Serverless using the following service name:

```
com.amazonaws.region.airflow-serverless
```

If you enable private DNS for the interface endpoint, you can make API requests to Amazon MWAA Serverless using its default Regional DNS name. For example, `airflow-serverless.us-east-1.api.com`.

Create an endpoint policy for your interface endpoint

An endpoint policy is an IAM resource that you can attach to an interface endpoint. The default endpoint policy allows full access to Amazon MWAA Serverless through the interface endpoint. To control the access allowed to Amazon MWAA Serverless from your VPC, attach a custom endpoint policy to the interface endpoint.

An endpoint policy specifies the following information:

- The principals that can perform actions (AWS accounts, IAM users, and IAM roles).
- The actions that can be performed.
- The resources on which the actions can be performed.

For more information, refer to Control access to services using endpoint policies in the *AWS PrivateLink Guide*.

Example: VPC endpoint policy for Amazon MWAA Serverless actions

The following is an example of a custom endpoint policy. When you attach this policy to your interface endpoint, it grants access to the listed Amazon MWAA Serverless actions for all principals on all resources.

```
{
  "Statement": [
    {
      "Principal": "*",
      "Effect": "Allow",
      "Action": [
        "airflow-serverless:GetWorkflow",
        "airflow-serverless:ListWorkflows"
      ],
      "Resource":"*"
    }
  ]
}
```

Supported operators

Amazon MWAA Serverless currently supports a specific set of Apache Airflow operators that are optimized for AWS service integration. These operators are categorized as follows:

Analytics

- `AthenaOperator`: Submits a Trino or Presto query to Athena.
- `AthenaSensor`: Polls the query state until it reaches a terminal state; fails if the query fails.
- `EmrAddStepsOperator`: Adds steps to an existing Amazon EMR `job_flow`.
- `EmrContainerOperator`: Submits jobs to Amazon EMR on Amazon EKS virtual clusters.
- `EmrContainerSensor`: Polls the state of the job run until it reaches a terminal state; fails if the job run fails.
- `EmrCreateJobFlowOperator`: Creates an Amazon EMR `JobFlow`, reading the config from the Amazon EMR connection.
- `EmrEksCreateClusterOperator`: Creates Amazon EMR on Amazon EKS virtual clusters.
- `EmrJobFlowSensor`: Polls an Amazon EMR `JobFlow` cluster until it reaches a target state; raises `AirflowException` upon failure.
- `EmrModifyClusterOperator`: Modifies an existing Amazon EMR cluster.
- `EmrNotebookExecutionSensor`: Polls the Amazon EMR notebook until it reaches a target state; raises `AirflowException` upon failure.
- `EmrServerlessApplicationSensor`: Polls the state of the application until it reaches a terminal state; fails if the application fails.
- `EmrServerlessCreateApplicationOperator`: Creates an EMR Serverless application.
- `EmrServerlessDeleteApplicationOperator`: Deletes an EMR Serverless application.
- `EmrServerlessJobSensor`: Polls the state of the job run until it reaches a terminal state; fails if the job run fails.
- `EmrServerlessStartJobOperator`: Starts an EMR Serverless job.
- `EmrServerlessStopApplicationOperator`: Stops an EMR Serverless application.
- `EmrStartNotebookExecutionOperator`: Starts an EMR Serverless notebook execution.
- `EmrStepSensor`: Polls the state of the step until it reaches a target state; raises `AirflowException` upon failure.

- `EmrStopNotebookExecutionOperator`: Stops a running EMR Serverless notebook execution.
- `EmrTerminateJobFlowOperator`: Terminates an Amazon EMR `JobFlow`.
- `GlueDataBrewStartJobOperator`: Starts an AWS Glue `DataBrew` job.
- `GlueDataQualityOperator`: Creates a data quality ruleset with DQDL rules applied to a specified AWS Glue table.
- `GlueDataQualityRuleRecommendationRunOperator`: Starts a recommendation run that is used to generate rules. AWS Glue Data Quality analyzes the data and provides recommendations for a potential ruleset.
- `GlueDataQualityRuleRecommendationRunSensor`: Waits for an AWS Glue data quality recommendation run to reach a given status.
- `GlueDataQualityRuleSetEvaluationRunOperator`: Evaluates a ruleset against a data source (AWS Glue table).
- `GlueDataQualityRuleSetEvaluationRunSensor`: Waits for an AWS Glue data quality ruleset evaluation run to reach a given status.
- `GlueJobOperator`: Creates an AWS Glue job.
- `GlueJobSensor`: Waits for an AWS Glue job to reach a given status.
- `KinesisAnalyticsV2CreateApplicationOperator`: Creates an Amazon Managed Service for Apache Flink application.
- `KinesisAnalyticsV2StartApplicationCompletedSensor`: Waits for an Amazon Managed Service for Apache Flink application to start.
- `KinesisAnalyticsV2StartApplicationOperator`: Starts an Amazon Managed Service for Apache Flink application.
- `KinesisAnalyticsV2StopApplicationCompletedSensor`: Waits for Amazon Managed Service for Apache Flink application to stop.
- `KinesisAnalyticsV2StopApplicationOperator`: Stops an Amazon Managed Service for Apache Flink application.
- `OpenSearchServerlessCollectionActiveSensor`: Polls the state of the collection until it reaches a terminal state; fails if the query fails.
- `QuickSightCreateIngestionOperator`: Creates and starts a new SPICE ingestion for a dataset; also helps refresh existing SPICE datasets.
- `QuickSightSensor`: Watches for an Amazon Quick Suite ingestion status.
- `RedshiftClusterSensor`: Waits for an Amazon Redshift cluster to reach a specific status.

- `RedshiftCreateClusterOperator`: Creates a new cluster with the specified parameters.
- `RedshiftCreateClusterSnapshotOperator`: Creates a manual snapshot of the specified cluster; the cluster must be in the available state.
- `RedshiftDataOperator`: Runs SQL statements against an Amazon Redshift cluster using Redshift data.
- `RedshiftDeleteClusterOperator`: Deletes an Amazon Redshift cluster.
- `RedshiftDeleteClusterSnapshotOperator`: Deletes the specified manual snapshot.
- `RedshiftPauseClusterOperator`: Pauses an Amazon Redshift cluster if it has `available` status.
- `RedshiftResumeClusterOperator`: Resumes a paused Amazon Redshift cluster
- `SageMakerNotebookOperator`: Provides artifact execution functionality for Amazon SageMaker Unified Studio workflows.

Application integration

- `AppflowRecordsShortCircuitOperator`: Short-circuits an empty Amazon AppFlow run.
- `AppflowRunAfterOperator`: Updates filters to select only future data, then starts an Amazon AppFlow run.
- `AppflowRunBeforeOperator`: Updates filters to select only past data, then starts an Amazon AppFlow run.
- `AppflowRunDailyOperator`: Updates filters to select only a single day, then starts an Amazon AppFlow run.
- `AppflowRunFullOperator`: Removes all filters, then starts an Amazon AppFlow full run.
- `AppflowRunOperator`: Starts an Amazon AppFlow run as is.
- `EventBridgeDisableRuleOperator`: Disables an Amazon EventBridge rule.
- `EventBridgeEnableRuleOperator`: Enables an Amazon EventBridge rule.
- `EventBridgePutEventsOperator`: Puts events onto Amazon EventBridge.
- `EventBridgePutRuleOperator`: Creates or updates the specified Amazon EventBridge rule.
- `SnsPublishOperator`: Publishes a message to Amazon SNS.
- `SqsPublishOperator`: Publishes a message to an Amazon SQS queue;
- `SqsSensor`: Gets messages from an Amazon SQS queue, then deletes the messages from the queue.

- `StepFunctionGetExecutionOutputOperator`: Returns the output of an AWS Step Functions state machine run.
- `StepFunctionStartExecutionOperator`: Begins an AWS Step Functions state machine run.
- `StepFunctionExecutionSensor`: Polls the AWS Step Functions state machine run until it reaches a terminal state; fails if the task fails.

Catalog

- `GlueCatalogPartitionSensor`: Waits for a partition to show up in the AWS Glue catalog.
- `GlueCrawlerOperator`: Creates, updates, and triggers an AWS Glue crawler.
- `GlueCrawlerSensor`: Waits for an AWS Glue crawler to reach a given status.

Compute

- `BatchCreateComputeEnvironmentOperator`: Creates an AWS Batch compute environment.
- `BatchOperator`: Runs a job on AWS Batch.
- `BatchComputeEnvironmentSensor`: Polls the state of the AWS Batch environment until it reaches a terminal state; fails if the environment fails.
- `BatchJobQueueSensor`: Polls the state of the AWS Batch job queue until it reaches a terminal state; fails if the queue fails.
- `BatchSensor`: Polls the state of the AWS Batch job until it reaches a terminal state; fails if the job fails.
- `EC2CreateInstanceOperator`: Creates and starts a specified number of Amazon EC2 instances using Boto3.
- `EC2HibernateInstanceOperator`: Hibernates Amazon EC2 instances.
- `EC2RebootInstanceOperator`: Reboots Amazon EC2 instances.
- `EC2StartInstanceOperator`: Starts Amazon EC2 instances using Boto3.
- `EC2StopInstanceOperator`: Stops Amazon EC2 instances using Boto3.
- `EC2TerminateInstanceOperator`: Terminates Amazon EC2 instances using Boto3.
- `EC2InstanceStateSensor`: Polls the state of the Amazon EC2 instance until the instance reaches the target state.
- `EcsCreateClusterOperator`: Creates an Amazon ECS cluster.

- `EcsDeleteClusterOperator`: Deletes an Amazon ECS cluster.
- `EcsDeregisterTaskDefinitionOperator`: Deregisters a task definition on Amazon ECS.
- `EcsRegisterTaskDefinitionOperator`: Registers a task definition on Amazon ECS.
- `EcsRunTaskOperator`: Runs a task on Amazon ECS.
- `EcsClusterStateSensor`: Polls the cluster state until it reaches a terminal state; raises an `AirflowException` with the failure reason.
- `EcsTaskDefinitionStateSensor`: Polls the task definition until it reaches a terminal state; raises an `AirflowException` with the failure reason.
- `EcsTaskStateSensor`: Polls the task state until it reaches a terminal state; raises `AirflowException` with the failure reason.
- `EksCreateClusterOperator`: Creates an Amazon EKS cluster control plane.
- `EksCreateFargateProfileOperator`: Creates an AWS Fargate profile for an Amazon EKS cluster.
- `EksCreateNodegroupOperator`: Creates an Amazon EKS managed node group for an existing Amazon EKS cluster.
- `EksDeleteClusterOperator`: Deletes the Amazon EKS cluster control plane and all node groups that are attached to it.
- `EksDeleteFargateProfileOperator`: Deletes an AWS Fargate profile from an Amazon EKS cluster.
- `EksDeleteNodegroupOperator`: Deletes an Amazon EKS-managed node group from an Amazon EKS cluster.
- `EksPodOperator`: Runs a task in a Kubernetes pod on the specified Amazon EKS cluster.
- `EksClusterStateSensor`: Checks the state of an Amazon EKS cluster until it reaches the target state or another terminal state.
- `EksFargateProfileStateSensor`: Checks the state of an AWS Fargate profile until it reaches the target state or another terminal state.
- `EksNodegroupStateSensor`: Checks the state of an Amazon EKS-managed node group until it reaches the target state or another terminal state.
- `LambdaCreateFunctionOperator`: Creates an AWS Lambda function.
- `LambdaInvokeFunctionOperator`: Invokes an AWS Lambda function.
- `LambdaFunctionStateSensor`: Polls the deployment state of the AWS Lambda function until it reaches the target state.

Database

- `DynamoDBValueSensor`: Waits for an attribute value to be present for an item in a DynamoDB table.
- `NeptuneStartDbClusterOperator`: Starts an Amazon Neptune DB cluster.
- `NeptuneStopDbClusterOperator`: Stops an Amazon Neptune DB cluster.
- `RdsCancelExportTaskOperator`: Cancels an in-progress export task that is exporting a snapshot to Amazon S3.
- `RdsCopyDbSnapshotOperator`: Copies the specified DB instance or DB cluster snapshot.
- `RdsCreateDbInstanceOperator`: Creates an Amazon RDS DB instance.
- `RdsCreateDbSnapshotOperator`: Creates a snapshot of a DB instance or DB cluster.
- `RdsCreateEventSubscriptionOperator`: Creates an Amazon RDS event notification subscription.
- `RdsDbSensor`: Waits for an Amazon RDS instance or cluster to enter one of a number of states.
- `RdsDeleteDbInstanceOperator`: Deletes an Amazon RDS DB instance.
- `RdsDeleteDbSnapshotOperator`: Deletes a DB instance or cluster snapshot, or terminates the copy operation.
- `RdsDeleteEventSubscriptionOperator`: Deletes an Amazon RDS event notification subscription.
- `RdsExportTaskExistenceSensor`: Waits for Amazon RDS export task with a specific status.
- `RdsSnapshotExistenceSensor`: Waits for Amazon RDS snapshot with a specific status.
- `RdsStartDbOperator`: Starts an Amazon RDS DB instance or cluster.
- `RdsStartExportTaskOperator`: Starts the export of a snapshot to Amazon S3. The specified IAM role must have access to the Amazon S3 bucket.
- `RdsStopDbOperator`: Stops an Amazon RDS DB instance or cluster.

Machine learning

- `BedrockCreateDataSourceOperator`: Sets up an Amazon Bedrock data source to add to an Amazon Bedrock knowledge base.
- `BedrockCreateKnowledgeBaseOperator`: Creates a knowledge base with data sources that are used by Amazon Bedrock LLMs and agents.

- `BedrockCreateProvisionedModelThroughputOperator`: Creates a fine-tuning job to customize a base model.
- `BedrockCustomizeModelCompletedSensor`: Polls the state of the model customization job until it reaches a terminal state; fails if the job fails.
- `BedrockCustomizeModelOperator`: Creates a fine-tuning job to customize a base model.
- `BedrockIngestDataOperator`: Begins an ingestion job in which an Amazon Bedrock data source is added to an Amazon Bedrock knowledge base.
- `BedrockIngestionJobSensor`: Polls the ingestion job status until it reaches a terminal state; fails if creation fails.
- `BedrockInvokeModelOperator`: Invokes the specified Amazon Bedrock model to run inference using the provided input.
- `BedrockKnowledgeBaseActiveSensor`: Polls the knowledge base status until it reaches a terminal state; fails if creation fails.
- `BedrockProvisionModelThroughputCompletedSensor`: Polls the provisioned model throughput job until it reaches a terminal state; fails if the job fails.
- `BedrockRaGOperator`: Queries a knowledge base and generates responses based on the retrieved results with source citations.
- `BedrockRetrieveOperator`: Queries a knowledge base and retrieve results with source citations.
- `ComprehendCreateDocumentClassifierCompletedSensor`: Polls the state of the document classifier until it reaches a completed state; fails if the job fails.
- `ComprehendCreateDocumentClassifierOperator`: Creates an Amazon Comprehend document classifier that can categorize documents.
- `ComprehendStartPiiEntitiesDetectionJobCompletedSensor`: Polls the state of the PII entities detection job until it reaches a completed state; fails if the job fails.
- `ComprehendStartPiiEntitiesDetectionJobOperator`: Creates an Amazon Comprehend PII entities detection job for a collection of documents.
- `SageMakerAutoMLOperator`: Creates an auto ML job and learns to predict the given column from the data provided through Amazon S3.
- `SageMakerAutoMLSensor`: Polls the auto ML job until it reaches a terminal state; raises an `AirflowException` with the failure reason.
- `SageMakerCreateExperimentOperator`: Creates a SageMaker experiment that can then be associated with jobs.

- `SageMakerCreateNotebookOperator`: Creates a SageMaker notebook.
- `SageMakerDeleteModelOperator`: Deletes a SageMaker model.
- `SageMakerDeleteNotebookOperator`: Deletes a notebook instance.
- `SageMakerEndpointConfigOperator`: Creates an endpoint configuration that SageMaker hosting services uses to deploy models.
- `SageMakerEndpointOperator`: Provisions and manages the compute resources for you when you create a serverless endpoint.
- `SageMakerEndpointSensor`: Polls the endpoint state until it reaches a terminal state; raises an `AirflowException` with the failure reason.
- `SageMakerModelOperator`: Creates a model in SageMaker.
- `SageMakerPipelineSensor`: Polls the pipeline until it reaches a terminal state; raises an `AirflowException` with the failure reason.
- `SageMakerProcessingOperator`: Analyzes data and evaluate machine learning models on SageMaker.
- `SageMakerProcessingSensor`: Poll the processing job until it reaches a terminal state; raise AirflowException with the failure reason.
- `SageMakerRegisterModelVersionOperator`: Registers a SageMaker model by creating a model version that specifies the model group to which it belongs.
- `SageMakerStartNoteBookOperator`: Starts a notebook instance.
- `SageMakerStartPipelineOperator`: Starts a SageMaker pipeline execution.
- `SageMakerStopNotebookOperator`: Stops a notebook instance.
- `SageMakerStopPipelineOperator`: Stops a SageMaker pipeline execution.
- `SageMakerTrainingOperator`: Starts a model training job.
- `SageMakerTrainingSensor`: Polls the training job until it reaches a terminal state; raises an `AirflowException` with the failure reason.
- `SageMakerTransformOperator`: Starts a transform job.
- `SageMakerTransformSensor`: Polls the transform job until it reaches a terminal state; raises an `AirflowException` with the failure reason.
- `SageMakerTuningOperator`: Starts a hyperparameter tuning job.
- `SageMakerTuningSensor`: Polls the tuning state until it reaches a terminal state; raise AirflowException with the failure reason.

Management and governance

- `CloudFormationCreateStackOperator`: Creates an CloudFormation stack.
- `CloudFormationDeleteStackOperator`: Deletes an CloudFormation stack.
- `CloudFormationCreateStackSensor`: Waits for a stack to be successfully created on CloudFormation.
- `CloudFormationDeleteStackSensor`: Waits for a stack to be successfully deleted on CloudFormation.
- `DataSyncOperator`: Finds, creates, updates, executes, and deletes DataSync tasks.
- `DmsCreateTaskOperator`: Creates an AWS DMS replication task.
- `DmsDeleteTaskOperator`: Deletes an AWS DMS replication task.
- `DmsDescribeTasksOperator`: Describes an AWS DMS replication task.
- `DmsStartTaskOperator`: Starts an AWS DMS replication task.
- `DmsStopTaskOperator`: Stops an AWS DMS replication task.
- `DmsTaskCompletedSensor`: Pokes an AWS DMS task until it is completed.

Storage

- `GlacierCreateJobOperator`: Initiates an Amazon Glacier inventory-retrieval job.
- `GlacierUploadArchiveOperator`: Adds an archive to an Amazon Glacier vault.
- `GlacierJobOperationSensor`: Checks job state; runs only in reschedule mode.
- `S3CopyObjectOperator`: Creates a copy of an object that is already stored in Amazon S3.
- `S3CreateBucketOperator`: Creates an Amazon S3 bucket.
- `S3CreateObjectOperator`: Creates a new object from data as a string or bytes.
- `S3DeleteBucketOperator`: Deletes an Amazon S3 bucket.
- `S3DeleteBucketTaggingOperator`: Deletes tagging from an Amazon S3 bucket.
- `S3DeleteObjectsOperator`: Deletes objects from a bucket using a single HTTP request.
- `S3FileTransformOperator`: Copies data from a source Amazon S3 location to a temporary location on the local filesystem.
- `S3GetBucketTaggingOperator`: Gets tagging from an Amazon S3 bucket.
- `S3ListOperator`: Lists all objects from the bucket with the specified string prefix in its name.

- `S3ListPrefixesOperator`: Lists all subfolders from the bucket with the specified string prefix in its name.
- `S3PutBucketTaggingOperator`: Puts tagging for an Amazon S3 bucket.
- `S3KeySensor`: Waits for one or multiple keys (a file-like instance on Amazon S3) to be present in an Amazon S3 bucket.
- `S3KeysUnchangedSensor`: Returns `True` if `inactivity_period` has passed with no increase in the number of objects that match the prefix.

Quotas for Amazon MWAA Serverless

Your AWS account has default quotas for each AWS service. These were formerly called limits. Unless noted otherwise, each quota is Region-specific. You can request increases for some quotas, but other quotas can't be increased.

To view the quotas for Amazon MWAA Serverless, open the Service Quotas console. In the navigation pane, choose **AWS services** and select **Amazon MWAA Serverless**.

To request a quota increase, refer to Requesting a Quota Increase in the *Service Quotas User Guide*. If the quota isn't available in Service Quotas, use the limit increase form.

Your AWS account has these quotas related to Amazon MWAA Serverless.

Resource	Default
Maximum workflows per account	100
Maximum workflow versions per workflow	50
Maximum concurrent runs per account	100
Maximum concurrent runs per workflow	20
Maximum XCom data in kilobytes	200KB
Maximum DAG definition size in kilobytes	50KB
Maximum task execution timeout	60 minutes

Document history

The following table describes the important changes to the documentation since the last release of Amazon MWAA Serverless. For notification about updates to this documentation, you can subscribe to an RSS feed.

Change	Description	Date
Initial release	Initial release of the Amazon MWAA Serverless service and user guide.	November 17, 2025