aws

Developer Guide

Amazon MQ

Amazon MQ: Developer Guide

Table of Contents

What is Amazon MQ?

Amazon MQ is a managed message broker service for Apache ActiveMQ Classic and RabbitMQ that manages the setup, operation, and maintenance of message brokers. You can create a new Amazon MQ broker using industry standard messaging protocols, or migrate existing message brokers to Amazon MQ without rewriting messaging code.

A *broker* is a message broker environment running on Amazon MQ. It is the basic building block of Amazon MQ. A *message* broker allows software applications and components to communicate using various programming languages, operating systems, and formal messaging protocols. You can use Amazon MQ brokers for communication between large scale, cloud native applications and components.

Topics

Amazon MQ features

Managed maintenance and version upgrades

Amazon MQ performs maintenance and version upgrades for a message broker during your scheduled maintenance window.

Monitor brokers with CloudWatch

Amazon MQ is integrated with Amazon CloudWatch so you can view and analyze metrics for your brokers and queues. You can view and analyze metrics from the Amazon MQ console, the CloudWatch console, command line, and API. Metrics are automatically collected and pushed to CloudWatch every minute.

Security

Amazon MQ provides encryption of your messages at rest and in transit. Connections to the broker use SSL, and access can be restricted to a private endpoint within your Amazon VPC. Additonality,

you can use AWS Identity and Access Management (IAM) to control the actions your IAM users and groups can take on specific Amazon MQ brokers.

Quorum queues for RabbitMQ on Amazon MQ

Quorum queues are a replicated queue type made up of a leader node (primary replica) and follower nodes (other replicas). Each node is in a different availability zone, so if one node is temporarily unavailable, message delivery continues with a newly elected leader replica in another availability zone. Quorum queues are useful for handling poison messages, which occur when a message fails and is requeued multiple times.

Cross-Region data replication for ActiveMQ on Amazon MQ

Cross-Region data replication (CRDR) allows for asynchronous message replication from the primary broker in a primary AWS Region to the replica broker in a replica Region. By issuing a failover request to the Amazon MQ API, the current replica broker is promoted to the primary broker role, and the current primary broker is demoted to the replica role.

How can I get started with Amazon MQ?

To get started with *ActiveMQ on Amazon MQ*, review the following documentation:

- Getting started: Creating and connecting to an ActiveMQ broker
- the section called “Deploying a broker”
- ActiveMQ tutorials
- the section called “Amazon MQ for ActiveMQ best practices”

To get started with *RabbitMQ on Amazon MQ*, review the following documentation:

- Getting started: Creating and connecting to a RabbitMQ broker
- the section called “Deploying a RabbitMQ broker”
- the section called “RabbitMQ tutorials”
- the section called “Amazon MQ for RabbitMQ best practices”

To learn about Amazon MQ REST APIs, see the *Amazon MQ REST API Reference*.

To learn about Amazon MQ AWS CLI commands, see Amazon MQ in the *AWS CLI Command Reference*.

How can I provide feedback to Amazon MQ?

We welcome and encourgae your feedback on the documentation. You can use the thumbs up and thumbs down icons on the right hand side to submit feedback, or you can use the "Provide feedback" form linked below.

To contact the Amazon MQ team, use the Amazon MQ Discussion Forum.

Setting up Amazon MQ

Before you can use Amazon MQ, you must complete the following steps.

Topics

Step 1: Prerequisites

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*.

Step 2: create a user and get your AWS credentials

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 AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions for the interface that you want to use. • 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*.

Which user needs programmatic access?	To	By
		• 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*.
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*.

Step 3: get ready to use the example codes

The following tutorials show how you can work with Amazon MQ brokers using the AWS Management Console as well as how to connect to your Amazon MQ for ActiveMQ and Amazon

MQ for RabbitMQ brokers programatically. To use the ActiveMQ Java example code, you must install the Java Standard Edition Development Kit and make some changes to the code.

You can also create and manage brokers programmatically using Amazon MQ REST API and AWS SDKs.

Next steps

Now that you're prepared to work with Amazon MQ, get started by creating a broker. Depending on your broker engine type, you can then connect a Java application to your Amazon MQ for ActiveMQ broker or use the RabbitMQ Java client library to connect a JVM-based application to your Amazon MQ for RabbitMQ broker.

Getting started: Creating and connecting to an ActiveMQ broker

A *broker* is a message broker environment running on Amazon MQ. It is the basic building block of Amazon MQ. The combined description of the broker instance *class* (`m5`) and *size* (`large`, `medium`) is called the *broker instance type* (for example, `mq.m5.large`). For more information, see What is an Amazon MQ for ActiveMQ broker?.

Create an ActiveMQ broker

The first and most common Amazon MQ task is creating a broker. The following example shows how you can use the AWS Management Console to create a basic broker.

1. Sign in to the Amazon MQ console.
2. On the **Select broker engine** page, choose **Apache ActiveMQ**.
3. On the **Select deployment and storage** page, in the **Deployment mode and storage type** section, do the following:

   a. Choose the **Deployment mode** (for example, **Active/standby broker**). For more information, see Deployment options for Amazon MQ for ActiveMQ brokers.

   - A **Single-instance broker** is comprised of one broker in one Availability Zone. The broker communicates with your application and with an Amazon EBS or Amazon EFS storage volume. For more information, see Option 1: Amazon MQ single-instance brokers.
   - An **Active/standby broker for high availability** is comprised of two brokers in two different Availability Zones, configured in a *redundant pair*. These brokers communicate synchronously with your application, and with Amazon EFS. For more information, see Option 2: Amazon MQ active/standby brokers for high availability.

   b. Choose the **Storage type** (for example, **EBS**). For more information, see Storage.

   > **Note**
   >
   > Amazon EBS replicates data within a single Availability Zone and doesn't support the ActiveMQ active/standby deployment mode.

c. Choose **Next**.

4. On the **Configure settings** page, in the **Details** section, do the following:

   a. Enter the **Broker name**.

> **Important**
>
> Do not add personally identifiable information (PII) or other confidential or sensitive information in broker names. Broker names are accessible to other AWS services, including CloudWatch Logs. Broker names are not intended to be used for private or sensitive data.

> **Note**
>
> In the **Additional settings** section, you can also configure the following:
>
> - Configurations
> - CloudWatch logs
> - Private access
> - Broker maintenance window

   b. Choose the **Broker instance type** (for example, **mq.m5.large**). For more information, see Broker instance types.

5. In the **ActiveMQ Web Console access** section, provide a **Username** and **Password**. The following restrictions apply to broker usernames and passwords:

- Your username can contain only alphanumeric characters, dashes, periods, underscores, and tildas (- . _ ~).
- Your password must be at least 12 characters long, contain at least 4 unique characters and must not contain commas, colons, or equal signs (,:=).

> **Important**
>
> Do not add personally identifiable information (PII) or other confidential or sensitive information in broker usernames. Broker usernames are accessible to other AWS

services, including CloudWatch Logs. Broker usernames are not intended to be used for private or sensitive data.

6. Choose **Deploy**.

   While Amazon MQ creates your broker, it displays the **Creation in progress** status.

   Creating the broker takes about 15 minutes.

   When your broker is created successfully, Amazon MQ displays the **Running** status.

7. Choose ***MyBroker***.

   On the ***MyBroker*** page, in the **Connect** section, note your broker's **ActiveMQ web console** URL, for example:

   ```
   https://b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-east-2.amazonaws.com:8162
   ```

   Also, note your broker's wire-level protocol **Endpoints**. The following is an example of an OpenWire endpoint:

   ```
   ssl://b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-east-2.amazonaws.com:61617
   ```

Getting started: Creating and connecting to a RabbitMQ broker

A *broker* is a message broker environment running on Amazon MQ. It is the basic building block of Amazon MQ. The combined description of the broker instance *class* (`m5`) and *size* (`large`, `medium`) is called the *broker instance type* (for example, `mq.m5.large`). For more information, see What is an Amazon MQ for RabbitMQ broker?

Create a RabbitMQ broker

The first and most common Amazon MQ task is creating a broker. The following example shows how you can use the AWS Management Console to create a basic broker.

When creating an Amazon MQ for RabbitMQ broker, follow the broker setup best practices for RabbitMQ to maximize broker performance and optimize message throughput efficiency.

1. Sign in to the Amazon MQ console.
2. On the **Select broker engine** page, choose **RabbitMQ**, and then choose **Next**.
3. On the **Select deployment mode** page, choose the **Deployment mode**, for example, **Cluster deployment**, and then choose **Next**.

   - A **single-instance broker** is comprised of one broker in one Availability Zone behind a Network Load Balancer (NLB). The broker communicates with your application and with an Amazon EBS storage volume. For more information, see Option 1: Amazon MQ for RabbitMQ single-instance broker.
   - A **RabbitMQ cluster deployment for high availability** is a logical grouping of three RabbitMQ broker nodes behind a Network Load Balancer, each sharing users, queues, and a distributed state across multiple Availability Zones (AZ). For more information, see Option 2: Amazon MQ for RabbitMQ cluster deployment.
4. On the **Configure settings** page, in the **Details** section, the following:

   a. Enter the Broker name.

> **Important**
>
> Do not add personally identifiable information (PII) or other confidential or sensitive information in broker names. Broker names are accessible to other AWS services, including CloudWatch Logs. Broker names are not intended to be used for private or sensitive data.

b. Choose the **Broker instance type** (for example, **mq.m7g.large**). For more information, see Broker instance types.

5. On the **Configure settings** page, in the **RabbitMQ access** section, provide a **Username** and **Password**. The following restrictions apply to broker sign-in credentials:

- Your username can contain only alphanumeric characters, dashes, periods, and underscores (- . _). This value must not contain any tilde (~) characters. Amazon MQ prohibits using `guest` as a username.
- Your password must be at least 12 characters long, contain at least 4 unique characters and must not contain commas, colons, or equal signs (,:=).

> **Important**
>
> Do not add personally identifiable information (PII) or other confidential or sensitive information in broker usernames. Broker usernames are accessible to other AWS services, including CloudWatch Logs. Broker usernames are not intended to be used for private or sensitive data.

> **Note**
>
> In the **Additional settings** section, you can also configure the following:
>
> - Configurations
> - CloudWatch logs
> - Private access
> - Broker maintenance window

6. Choose **Next**.
7. On the **Review and create** page, you can review your selections and edit them as needed.
8. Choose **Create broker**.

   While Amazon MQ creates your broker, it displays the **Creation in progress** status.

   Creating the broker takes about 15 minutes.

   When your broker is created successfully, Amazon MQ displays the **Running** status.
9. Choose ***MyBroker***.

   On the ***MyBroker*** page, in the **Connect** section, note your broker's **RabbitMQ web console** URL, for example:

   ```
   https://b-c8349341-ec91-4a78-ad9c-a57f23f235bb.mq.us-west-2.on.aws
   ```

   Also, note your broker's secure-AMQP **Endpoint**. The following is an example of an `amqps` endpoint exposing listener port 5671.

   ```
   amqps://b-c8349341-ec91-4a78-ad9c-a57f23f235bb.mq.us-west-2.on.aws:5671
   ```

Managing an Amazon MQ broker

After you create a broker, you can manage and maintain the different components of your Amazon MQ broker.

Topics

Connecting to Amazon MQ

You can connect to Amazon MQ from other AWS services using service endpoints and broker endpoints.

Service endpoints

The following connection methods are used for the **Amazon MQ service API:**

Domains	Connection method
`mq.region.amazonaws.com`	IPv4
`mq.region.api.aws`	Dual-stack (IPv4 and IPv6)
`mq-fips.region.amazonaws.com`	FIPS with IPv4 only

Domains	Connection method
`mq-fips.region.api.aws`	FIPS with Dual-stack

Broker endpoints

The following connection methods are used for **Amazon MQ brokers:**

Domains	Connection method
`brokerId.mq.region.amazonaws.com`	IPv4
`brokerId.mq.region.on.aws`	Dual-stack (IPv4 and IPv6) **Note** Amazon MQ for ActiveMQ brokers do not support dual-stack.

Connect to Amazon MQ using Dual-stack (IPv4 and IPv6) endpoints

Dual-stack endpoints support both IPv4 and IPv6 traffic. When you make a request to a dual-stack endpoint, the endpoint URL resolves to an IPv4 or an IPv6 address. For more information on dual-stack and FIPS endpoints, see the SDK Reference guide.

Amazon MQ supports Regional dual-stack endpoints, which means that you must specify the AWS Region as part of the endpoint name. Dual-stack endpoint names use the following naming convention: `mq.region.api.aws`. For example, the dual-stack endpoint name for the `eu-west-1` Region is `mq.eu-west-1.api.aws`.

For the full list of Amazon MQ endpoints, see the AWS General Reference.

Connect to Amazon MQ using AWS PrivateLink

AWS PrivateLink endpoints for Amazon MQ API with support for IPv4 and IPv6 provides private connectivity between virtual private clouds (VPCs) and the Amazon MQ API without exposing your traffic to the public internet.

> **Note**
>
> Support for PrivateLink is only available for the Amazon MQ API endpoint, not the broker endpoint. For more information on privately connecting to a broker endpoint, see Configuring a private Amazon MQ broker.

To access Amazon MQ API using PrivateLink, you must first create an interface VPC endpoint in the specific VPC you want to connect from. When you create the VPC endpoint, use the service name `com.amazonaws.region.mq` or `com.amazonaws.region.mq-fips` for FIPS endpoints.

When you call Amazon MQ using the AWS CLI or SDK, you must specify the endpoint URL to use the dual-stack domain name: `mq.region.api.aws` or `mq-fips.region.api.aws`. PrivateLink for Amazon MQ does not support the default domain name ending in `amazonaws.com`. For more information , see Dual-stack and FIPS endpoints in the SDK Reference Guide.

The following CLI example shows how to call the `describe-broker-engine-type` in the Asia Pacific (Sydney) Region through an Amazon MQ VPC endpoint.

```
AWS_USE_DUALSTACK=true aws mq describe-broker-engine-types --region ap-southeast-2
```

For other ways to configure the endpoint in CLI, see Using endpoints in the AWS CLI

You can also determine user access to the VPC endpoints using VPC endpoint policies. For more information, see Control access to VPC endpoints using endpoint policies.

Authentication and authorization for Amazon MQ brokers

Amazon MQ offers multiple authentication and authorization methods to secure your messaging infrastructure according to your organization's requirements.

Authentication and authorization for Amazon MQ for RabbitMQ

Amazon MQ for RabbitMQ supports the following authentication and authorization methods:

Simple authentication and authorization

In this method, broker users are stored internally in the RabbitMQ broker and managed through the web console or management API. Permissions for vhosts, exchanges, queues, and topics are

configured directly in RabbitMQ. This is the default method. For more information, see Simple authentication and authorization.

OAuth 2.0 authentication and authorization

In this method, broker users and their permissions are managed by an external OAuth 2.0 identity provider (IdP). User authentication and resource permissions for vhosts, exchanges, queues, and topics are centralized through the OAuth 2.0 provider's scope system. This simplifies user management and enables integration with existing identity systems. For more information, see OAuth 2.0 authentication and authorization.

IAM authentication and authorization

In this method, broker users authenticate using AWS IAM credentials through IAM outbound federation. IAM credentials are used to obtain JWT tokens from AWS Security Token Service (STS), and these JWT tokens serve as OAuth 2.0 tokens for authentication. This method leverages the existing OAuth 2.0 support in Amazon MQ for RabbitMQ, where AWS acts as the OAuth 2.0 identity provider. User authentication is handled by AWS IAM, while resource permissions for vhosts, exchanges, queues, and topics are managed through IAM policies and scope aliases configured in RabbitMQ. For more information, see IAM authentication and authorization.

LDAP authentication and authorization

In this method, broker users and their permissions are managed by an external LDAP directory service. User authentication and resource permissions are centralized through the LDAP server, allowing users to access RabbitMQ using their existing directory service credentials. For more information, see LDAP authentication and authorization.

HTTP authentication and authorization

In this method, broker users and their permissions are managed by an external HTTP server. User authentication and resource permissions are centralized through the HTTP server, allowing users to access RabbitMQ using their own Authentication and Authorization provider. For more information about this method, see HTTP authentication and authorization.

SSL certificate authentication

Amazon MQ supports mutual TLS (mTLS) for RabbitMQ brokers. The SSL authentication plugin uses client certificates from mTLS connections to authenticate users. In this method, broker users are authenticated using X.509 client certificates instead of username and password credentials.

The client's certificate is validated against a trusted Certificate Authority (CA), and the username is extracted from a field in the certificate, such as the Common Name (CN) or Subject Alternative Name (SAN). This method provides strong authentication without transmitting credentials over the network. For more information, see SSL certificate authentication.

> **Note**
>
> RabbitMQ supports multiple authentication and authorization methods to be used simultaneously. For example, you can enable both OAuth 2.0 and simple (internal) authentication. For more information, see the OAuth 2.0 tutorial section on enabling both OAuth 2.0 and simple (internal) authentication and the RabbitMQ access control documentation.
>
> Amazon MQ recommends creating an internal user when testing authentication configurations. This allows access configuration to be validated using RabbitMQ management API. For more information, see Access validation.

Authentication and authorization for Amazon MQ for ActiveMQ

Amazon MQ for ActiveMQ supports the following authentication and authorization methods:

Simple authentication and authorization

In this method, broker users are created and managed through the Amazon MQ console or API. Users can be configured with specific permissions to access queues, topics, and the ActiveMQ Web Console. For more information about this method, see Creating an ActiveMQ broker user.

LDAP authentication and authorization

In this method, broker users authenticate through credentials stored in your LDAP server. You can add, delete, and modify users and assign permissions to topics and queues through the LDAP server, providing centralized authentication and authorization. For more information about this method, see Integrating ActiveMQ brokers with LDAP.

Upgrading an Amazon MQ broker engine version

Amazon MQ regularly provides new broker engine versions for all supported broker engine types. New engine versions include security patches, bug fixes, and other broker engine improvements.

Amazon MQ organizes version numbers according to semantic versioning specification as `X.Y.Z`. In Amazon MQ implementations, `X` denotes the major version, `Y` represents the minor version, and `Z` denotes the patch version number. Amazon MQ supports two types of upgrades:

- **Major version upgrade** – Occurs when the major engine version numbers change. For example, upgrading from RabbitMQ version **3**.13 to version **4**.2 is considered a major version upgrade.
- **Minor version upgrade** – Occurs when only the minor engine version number changes. For example, upgrading from version 3.**11** to version 3.**12** is considered a minor version upgrade.

You can manually upgrade your broker at any time to the next supported major or minor version. Amazon MQ manages upgrade to the latest supported patch version for all brokers during the scheduled maintenance window. Both manual and automatic version upgrades occur during the scheduled maintenance window, or after you reboot your broker. Amazon MQ upgrades your broker to the next minor version when the current minor version reaches end of support.

Manually upgrading the engine version

You can upgrade the engine version of a broker by using the AWS Management Console, the AWS CLI, or the Amazon MQ API.

AWS Management Console

To upgrade the engine version of a broker by using the AWS Management Console

1. On the broker details page, choose **Edit**.
2. Under **Specifications**, for **Broker engine version** choose the new version number from the dropdown list.
3. Scroll to the bottom of the page, and choose **Schedule modifications**.
4. On the **Schedule broker modifications** page, for **When to apply modifications**, choose one of the following.

    - Choose **After the next reboot**, if you want Amazon MQ to complete the version upgrade during the next scheduled maintenance window.
    - Choose **Immediately**, if you want to reboot the broker and upgrade the engine version immediately.

> **Important**
>
> Single instance brokers are offline while being rebooted. For cluster brokers, only one node is down at a time while the broker reboots.

5. Choose **Apply** to finish applying the changes.

AWS CLI

To upgrade the engine version of a broker by using the AWS CLI

1. Use the update-broker CLI command and specify the following parameters, as shown in the example.

   - `--broker-id` – The unique ID that Amazon MQ generates for the broker. You can parse the ID from your broker ARN. For example, given the following ARN, `arn:aws:mq:us-east-2:123456789012:broker:MyBroker:b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9`, the broker ID would be `b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9`.
   - `--engine-version` – The engine version number for the broker engine to upgrade to.

   ```
   aws mq update-broker --broker-id broker-id --engine-version version-number
   ```

2. (Optional) Use the reboot-broker CLI command to reboot your broker if you want to upgrade the engine version immediately.

   ```
   aws mq reboot-broker --broker-id broker-id
   ```

   If you do not want to reboot your broker and apply the changes immediately, Amazon MQ will upgrade the broker during the next scheduled maintenance window.

   > **Important**
   >
   > Single instance brokers are offline while being rebooted. For cluster brokers, only one node is down at a time while the broker reboots.

Amazon MQ API

To upgrade the engine version of a broker by using the Amazon MQ API

1. Use the UpdateBroker API operation. Specify `broker-id` as a path parameter. The following examples assumes a broker in the `us-west-2` region. For more information about available Amazon MQ endpoints, see Amazon MQ endpoints and quotas. in the *AWS General Reference*

```
PUT /v1/brokers/broker-id HTTP/1.1
Host: mq.us-west-2.amazonaws.com
Date: Mon, 7 June 2021 12:00:00 GMT
x-amz-date: Mon, 7 June 2021 12:00:00 GMT
Authorization: authorization-string
```

   Use `engineVersion` in the request payload to specify the version number for the broker to upgrade to.

```
{
    "engineVersion": "engine-version-number"
}
```

2. (Optional) Use the RebootBroker API operation to reboot your broker if you want to upgrade the engine version immediately. `broker-id` is specified as a path parameter.

```
POST /v1/brokers/broker-id/reboot-broker HTTP/1.1
Host: mq.us-west-2.amazonaws.com
Date: Mon, 7 June 2021 12:00:00 GMT
x-amz-date: Mon, 7 June 2021 12:00:00 GMT
Authorization: authorization-string
```

   If you do not want to reboot your broker and apply the changes immediately, Amazon MQ will upgrade the broker during the next scheduled maintenance window.

> **Important**
>
> Single instance brokers are offline while being rebooted. For cluster brokers, only one node is down at a time while the broker reboots.

Upgrading an Amazon MQ broker instance type

> **Important**
>
> `mq.m7g.x` instances are only available for Amazon MQ for RabbitMQ brokers. Amazon MQ for ActiveMQ brokers only use `mq.m5.x` instances.

The combined description of the broker instance class (`m7g`) and size (`large`) is called the broker instance type (for example, `mq.m7g.large`). When choosing an instance type, it is important to consider factors that will affect broker performance:

- the number of clients and queues
- the volume of messages sent
- messages kept in memory
- redundant messages

Smaller broker instance types (`mq.m7g.medium`) are recommended only for testing application performance. We recommend larger broker instance types (`mq.m7g.large` and above) for production levels of clients and queues, high throughput, messages in memory, and redundant messages.

We recommend upgrading to a larger instance type (i.e. from `micro` to `large`) if you are experiencing performance issues, or if you are moving from testing to a production environment. To upgrade your instance type, you can use the AWS Management Console, the AWS CLI, or the Amazon MQ API.

AWS Management Console

To upgrade to a larger instance type using the AWS Management Console, do the following:

1. Sign in to the Amazon MQ console.
2. In the left navigation pane, choose **Brokers**, and then choose the broker that you want to upgrade from the list.
3. On the broker details page, choose **Edit**.
4. Under **Specifications**, for **Broker instance type** choose the new instance type from the dropdown list.

5. At the bottom of the page, choose **Schedule modifications**.
6. On the **Schedule broker modifications** page, for **When to apply modifications**, choose one of the following.

   - Choose **After the next reboot**, if you want Amazon MQ to complete the upgrade during the next scheduled maintenance window.
   - Choose **Immediately**, if you want to reboot the broker and upgrade the instance type immediately.

   > **Important**
   >
   > Single instance brokers are offline while being rebooted. For cluster brokers, only one node is down at a time while the broker reboots.

7. Choose **Apply** to finish applying the changes.

AWS CLI

To upgrade the instance type of a broker by using the AWS CLI

1. Use the modify-broker CLI command and specify the following parameters, as shown in the example.

   - `--broker-id` – The unique ID that Amazon MQ generates for the broker.
   - `--host-instance-type` – The engine version number for the broker engine to upgrade to.

   ```
   aws mq modify-broker --broker-id broker-id --host-instance-type instance-type
   ```

2. (Optional) Use the reboot-broker CLI command to reboot your broker if, you want to upgrade the instance type immediately.

   ```
   aws mq reboot-broker --broker-id broker-id
   ```

   If you do not want to reboot your broker and apply the changes immediately, Amazon MQ will upgrade the broker during the next scheduled maintenance window.

> **Important**
>
> Single instance brokers are offline while being rebooted. For cluster brokers, only one node is down at a time while the broker reboots.

Amazon MQ API

To upgrade the instance-type of a broker by using the Amazon MQ API

1. Use the UpdateBroker API operation. Specify `broker-id` as a path parameter. The following examples assumes a broker in the `us-west-2` region. For more information about available Amazon MQ endpoints, see Amazon MQ endpoints and quotas in the *AWS General Reference*.

```
PUT /v1/brokers/broker-id HTTP/1.1
Host: mq.us-west-2.amazonaws.com
Date: Mon, 7 June 2021 12:00:00 GMT
x-amz-date: Mon, 7 June 2021 12:00:00 GMT
Authorization: authorization-string
```

   Use `host-instance-type` in the request payload to specify the instance type for the broker to upgrade to.

```
{
    "host-instance-type": "host-instance-type"
}
```

2. (Optional) Use the RebootBroker API operation to reboot your broker, if you want to upgrade the engine version immediately. `broker-id` is specified as a path parameter.

```
POST /v1/brokers/broker-id/reboot-broker HTTP/1.1
Host: mq.us-west-2.amazonaws.com
Date: Mon, 7 June 2021 12:00:00 GMT
x-amz-date: Mon, 7 June 2021 12:00:00 GMT
Authorization: authorization-string
```

   If you do not want to reboot your broker and apply the changes immediately, Amazon MQ will upgrade the broker during the next scheduled maintenance window.

> **Important**
>
> Single instance brokers are offline while being rebooted. For cluster brokers, only one node is down at a time while the broker reboots.

Amazon MQ for ActiveMQ storage types

Amazon MQ for ActiveMQ supports Amazon Elastic File System (EFS) and Amazon Elastic Block Store (EBS). By default, ActiveMQ brokers use Amazon EFS for broker storage. To take advantage of high durability and replication across multiple Availability Zones, use Amazon EFS. To take advantage of low latency and high throughput, use Amazon EBS.

> **Important**
>
> - You can use Amazon EBS only with the `mq.m5` broker instance type family.
> - Although you can change the *broker instance type*, you can't change the *broker storage type* after you create the broker.
> - Amazon EBS replicates data within a single Availability Zone and doesn't support the ActiveMQ active/standby deployment mode.

Differences between Storage Types

The following table provides a brief overview of the differences between in-memory, Amazon EFS, and Amazon EBS storage types for ActiveMQ brokers.

Storage Type	Persistence	Example Use Case	Approximate Maximum Number of Messages Enqueued per Producer, per Second (1KB Message)	Replication
In-memory	Non-persistent	• Stock quotes • Location data updates • Frequently changed data	5,000	None
Amazon EBS	Persistent	• High volumes of text • Order processing	500	Multiple copies within a single Availability Zone (AZ)
Amazon EFS	Persistent	Financial transactions	80	Multiple copies across multiple AZs

In-memory message storage provides the lowest latency and the highest throughput. However, messages are lost during instance replacement or broker restart.

Amazon EFS is designed to be highly durable, replicated across multiple AZs to prevent the loss of data resulting from the failure of any single component or an issue that affects the availability of an AZ. Amazon EBS is optimized for throughput and replicated across multiple servers within a single AZ.

Configuring a private Amazon MQ broker

A private broker does not have public accessibility and cannot be accessed from outside of your VPC. Before configuring a private broker, view the following information about VPCs, subnets, and security groups:

- **VPCs**
  - A broker's subnet(s) and security group(s) must be in the same VPC.
  - When you are using a private broker, you may see IP addresses that you did not configure with your VPC. These are IP addresses from the Amazon MQ infrastructure, and they require no action.
- **Subnets**
  - If subnets are within a shared VPC, the VPC must be owned by the same account creating the broker.
  - If no subnets are provided, the default subnets in the default VPC will be used.
  - Once the broker is created, the subnets used cannot be changed.
  - For cluster and active/standby brokers, subnets must be in different Availability Zones.
  - For single instance brokers, you can specify which subnet to use and the broker will be created within the same Availability Zone.
- **Security groups**
  - If no security group is provided, the default security groups in the default VPC will be used.
  - Single-instance, cluster, and active/standby brokers require at least one security group (for example, the default security group).

    > **Note**
    >
    > Public RabbitMQ brokers do not use subnets or security groups.

  - Once the broker is created, the security group used cannot be changed. The security groups can themselves still be modified.

Configuring a private broker in the AWS Management Console

To configure a private broker, begin creating a new broker in the AWS Management Console. Then, in the **Network settings** section, to configure your broker's connectivity, do the following:

1. Choose **Private access** for your broker. To connect to a private broker, you can use IPv4, IPv6, or dual-stack (IPv4 and IPv6). For more information, see Connecting to Amazon MQ.
2. Next, choose **Use the default VPC, subnet(s), and security group(s)**, or choose **Select existing VPC, subnet(s), and security group(s)**. If you do not wish to use the default or existing VPC, subnet(s), or security group(s), you must create a new one to connect to the private broker.

> **Note**
>
> For private broker access, the connection method will be the same as the selected IP type of the subnet. Once the broker is created, the VPC endpoint cannot be changed and will always have the IP type of the selected subnets. If you want to use a new IP type, you must create a new broker.

> **Note**
>
> Amazon MQ for ActiveMQ does not use VPC endpoints. When you first create an ActiveMQ broker, Amazon MQ provisions an elastic network interface (ENI) in the VPC. Security groups are placed in the ENI and can be used for both public and private brokers.

Accessing the Amazon MQ broker web console without public accessibility

When you turn off public accessibility for your broker, the AWS account ID that created the broker can access the private broker. If you turn off public accessibility for your broker, you must perform the following steps to access the broker web console.

1. Create a Linux EC2 instance in `public-vpc` (with a public IP, if necessary).
2. To verify that your VPC is configured correctly, establish an `ssh` connection to the EC2 instance and use the `curl` command with the URI of your broker.
3. From your machine, create an `ssh` tunnel to the EC2 instance using the path to your private key file and the IP address of your public EC2 instance. For example:

```
ssh -i ~/.ssh/id_rsa -N -C -q -f -D 8080 ec2-user@203.0.113.0
```

A forward proxy server is started on your machine.

4. Install a proxy client such as FoxyProxy on your machine.
5. Configure your proxy client using the following settings:

   - For proxy type, specify SOCKS5.
   - For IP address, DNS name, and server name, specify `localhost`.
   - For port, specify `8080`.
   - Remove any existing URL patterns.
   - For the URL pattern, specify `*.mq.*.amazonaws.com*`
   - For the connection type, specify `HTTP(S)`.

   When you enable your proxy client, you can access the web console on your machine.

> **Important**
>
> If you are using a private broker, you may see IP addresses that you did not configure with your VPC. These are IP addresses from the RabbitMQ on Amazon MQ infrastructure, and they require no action.

Scheduling the maintenance window for an Amazon MQ broker

Periodically, Amazon MQ performs maintenance to the hardware, operating system, or the engine software of a message broker during the maintenance window. For example, if you changed the broker instance type, Amazon MQ will apply your changes during the next scheduled maintenance window. The duration of the maintenance can last up to two hours depending on the operations that are scheduled for your message broker. You can minimize downtime during a maintenance window by selecting a broker deployment mode with high availability across multiple Availability Zones (AZ).

Amazon MQ for ActiveMQ provides active/standby deployments for high availability. In active/standby mode, Amazon MQ performs maintenance operations one instance at a time, and at least one instance remains available. In addition, you can configure a network of brokers with maintenance windows varied across the week. Amazon MQ for RabbitMQ provides the cluster

deployments for high availability. In cluster deployments, Amazon MQ performs maintenance operations one node at a time by keeping at least two running nodes at all times.

When you first create your broker, you can schedule the maintenance window to occur once a week at a specified time. You can only adjust the maintenance window of a broker up to four times before the next scheduled maintenance window. Once a broker maintenance window is completed, Amazon MQ resets the limit, and you can adjust the schedule again before the next maintenance window occurs. Broker availability is not affected when adjusting the broker maintenance window.

To adjust the broker maintenance window, you can use the AWS Management Console, the AWS CLI, or the Amazon MQ API.

Schedule the broker maintenance window using the AWS Management Console

To adjust the broker maintenance window by using the AWS Management Console

1. Sign in to the Amazon MQ console.
2. In the left navigation pane, choose **Brokers**, and then choose the broker that you want to upgrade from the list.
3. On the broker details page, choose **Edit**.
4. Under **Maintenance**, do the following.
    a. For **Start day**, choose a day of the week, for example, **Sunday**, from the drop-down list.
    b. For **Start time**, choose the hour and minute of the day that you want to schedule for the next broker maintenance window, for example, **12:00**.

    > **Note**
    >
    > The **Start time** options are configured in UTC+0 time zone.

5. Next, select **Schedule modifications**. Then choose **After the next reboot** or **Immediately**. Choosing **After the next reboot** will immediately update the maintenance window without rebooting the broker. Choosing **Immediately** will immediately reboot the broker.
6. On the broker details page, under **Maintenance window**, verify that your new preferred schedule is displayed.

Schedule the broker maintenance window using the AWS CLI

To adjust the broker maintenance window using the AWS CLI

1. Use the update-broker CLI command and specify the following parameters, as shown in the example.

   - `--broker-id` – The unique ID that Amazon MQ generates for the broker. You can parse the ID from your broker ARN. For example, given the following ARN, `arn:aws:mq:us-east-2:123456789012:broker:MyBroker:b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9`, the broker ID would be `b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9`.
   - `--maintenance-window-start-time` – The parameters that determine the weekly maintenance window start time provided in the following structure.
     - `DayOfWeek` – The day of the week, in the following syntax: `MONDAY| TUESDAY | WEDNESDAY | THURSDAY | FRIDAY | SATURDAY | SUNDAY`
     - `TimeOfDay` – The time, in 24-hour format.
     - `TimeZone` – (Optional) The time zone, in either the Country/City, or the UTC offset format. Set to UTC by default.

   ```
   aws mq update-broker --broker-id broker-id \
   --maintenance-window-start-time DayOfWeek=SUNDAY,TimeOfDay=13:00,TimeZone=America/Los_Angeles
   ```

2. (Optional) Use the describe-broker CLI command to verify that the maintenance window is successfully updated.

   ```
   aws mq describe-broker --broker-id broker-id
   ```

Schedule the broker maintenance window using the Amazon MQ API

To adjust the broker maintenance window using the Amazon MQ API

1. Use the UpdateBroker API operation. Specify `broker-id` as a path parameter. The following examples assumes a broker in the `us-west-2` region. For more information about available Amazon MQ endpoints, see Amazon MQ endpoints and quotas in the *AWS General Reference*.

```
PUT /v1/brokers/broker-id HTTP/1.1
Host: mq.us-west-2.amazonaws.com
Date: Wed, 7 July 2021 12:00:00 GMT
x-amz-date: Wed, 7 July 2021 12:00:00 GMT
Authorization: authorization-string
```

Use the `maintenanceWindowStartTime` parameter and the `WeeklyStartTime` resource type in the request payload.

```
{
"maintenanceWindowStartTime": {
    "dayOfWeek": "SUNDAY",
    "timeZone": "America/Los_Angeles",
    "timeOfDay": "13:00"
  }
}
```

2. (Optional) Use the DescribeBroker API operation to verify that the maintenance window has been successfully updated. `broker-id` is specified as a path parameter.

```
GET /v1/brokers/broker-id HTTP/1.1
Host: mq.us-west-2.amazonaws.com
Date: Wed, 7 July 2021 12:00:00 GMT
x-amz-date: Wed, 7 July 2021 12:00:00 GMT
Authorization: authorization-string
```

Rebooting an Amazon MQ broker

To apply a new configuration to a broker, you can reboot the broker.

> **Note**
>
> If your ActiveMQ broker becomes unresponsive, you can reboot it to recover from a faulty state.

The following example shows how you can reboot an Amazon MQ broker using the AWS Management Console.

To Reboot an Amazon MQ Broker

1. Sign in to the Amazon MQ console.
2. From the broker list, choose the name of your broker (for example, **MyBroker**).
3. On the ***MyBroker*** page, choose **Actions**, **Reboot broker**.

   > **Important**
   >
   > Single instance brokers will be offline while being rebooted. Cluster brokers will be available, but each node is rebooted one at a time.

4. In the **Reboot broker** dialog box, choose **Reboot**.

   Rebooting a broker takes about 5 minutes. If the reboot includes instance size changes or is performed on a broker with high queue depth, the rebooting process can take longer.

Deleting an Amazon MQ broker

If you don't use an Amazon MQ broker (and don't foresee using it in the near future), it is a best practice to delete it from Amazon MQ to reduce your AWS costs.

The following example shows how you can delete a broker using the AWS Management Console.

Deleting an Amazon MQ broker

1. Sign in to the Amazon MQ console.
2. From the broker list, select your broker (for example, **MyBroker**) and then choose **Delete**.
3. In the **Delete *MyBroker*?** dialog box, type `delete` and then choose **Delete**.

   Deleting a broker takes about 5 minutes.

Amazon MQ broker statuses

A broker's current condition is indicated by a *status*. The following table lists the statuses of an Amazon MQ broker.

Console	API	Description
Creation failed	`CREATION_FAILED`	The broker couldn't be created.
Creation in progress	`CREATION_IN_PROGRESS`	The broker is currently being created.
Deletion in progress	`DELETION_IN_PROGRESS`	The broker is currently being deleted.
Reboot in progress	`REBOOT_IN_PROGRESS`	The broker is currently being rebooted.
Running	`RUNNING`	The broker is operational.
Critical action required	`CRITICAL_ACTION_REQUIRED`	The broker is running, but is in a degraded state and requires immediate action. You can find instructions to resolve the issue by chosing the action required code from the list in *Troubleshooting*.

Adding tags to Amazon MQ resources

To organize and identify your Amazon MQ resources for cost allocation, you can add metadata *tags* that identify the purpose of a broker or configuration. This is especially useful when you have many brokers. You can use cost allocation tags to organize your AWS bill to reflect your own cost structure. To do this, sign up to get your AWS account bill to include the tag keys and values. For more information, see Setting Up a Monthly Cost Allocation Report in the *AWS Billing User Guide*.

For instance, you could add tags that represent the cost center and purpose of your Amazon MQ resources:

Resource	Key	Value
Broker1	Cost Center	34567
	Stack	Production
Broker2	Cost Center	34567
	Stack	Production
Broker3	Cost Center	12345
	Stack	Development

This tagging scheme allows you to group two brokers performing related tasks in the same cost center, while tagging an unrelated broker with a different cost allocation tag.

Adding tags in the Amazon MQ Console

You can quickly add tags to the resources you are creating in the Amazon MQ console by following these steps:

1. From the **Create a broker** page, select **Additional settings**.
2. Under **Tags**, select **Add tag**.
3. Enter a **Key** and **Value** pair.
4. (Optional) Select **Add tag** to add multiple tags to your broker.
5. Select **Create broker**.

To add tags as you create a configuration:

1. From the **Create configuration** page, select **Advanced**.
2. Under **Tags** on the **Create configuration** page, select **Add tag**.
3. Enter a **Key** and **Value** pair.
4. (Optional) Select **Add tag** to add multiple tags to your configuration.
5. Select **Create configuration**.

After adding tags, you can view, edit, and remove the tags for your resources in the Amazon MQ console. You can also view the tags of your resources using the REST API. For more information, see the Amazon MQ REST API Reference.

Using Amazon MQ for ActiveMQ

Amazon MQ makes it easy to create a message broker with the computing and storage resources that fit your needs. You can create, manage, and delete brokers using the AWS Management Console, Amazon MQ REST API, or the AWS Command Line Interface.

Amazon MQ for ActiveMQ brokers can be deployment as *single-instance brokers* or *active/standby brokers*. For both deployment modes, Amazon MQ provides high durability by storing its data redundantly.

> **Note**
>
> Amazon MQ uses Apache KahaDB as its data store. Other data stores, such as JDBC and LevelDB, aren't supported.

You can access your brokers by using any programming language that ActiveMQ supports and by enabling TLS explicitly for the following protocols:

- AMQP
- MQTT
- MQTT over WebSocket
- OpenWire
- STOMP
- STOMP over WebSocket

To learn about Amazon MQ REST APIs, see the *Amazon MQ REST API Reference*.

Amazon MQ for ActiveMQ brokers

What is an Amazon MQ for ActiveMQ broker?

A *broker* is a message broker environment running on Amazon MQ. It is the basic building block of Amazon MQ. The combined description of the broker instance *class* (`m5`) and *size* (`large`, `medium`) is called the *broker instance type* (for example, `mq.m5.large`). For more information, see Broker instance types.

- A *single-instance broker* is comprised of one broker in one Availability Zone. The broker communicates with your application and with an Amazon EBS or Amazon EFS storage volume.
- An *active/standby broker* is comprised of two brokers in two different Availability Zones, configured in a *redundant pair*. These brokers communicate synchronously with your application, and with Amazon EFS.

For more information, see Deployment options for Amazon MQ for ActiveMQ brokers.

You can enable *automatic minor version upgrades* to new minor versions of the broker engine, as Apache releases new versions. Automatic upgrades occur during the *maintenance window* defined by the day of the week, the time of day (in 24-hour format), and the time zone (UTC by default).

For information about creating and managing brokers, see the following:

- Getting started: Creating and connecting to an ActiveMQ broker
- Brokers
- Broker statuses

Supported wire-level protocols

You can access your brokers by using any programming language that ActiveMQ supports and by enabling TLS explicitly for the following protocols:

- AMQP
- MQTT
- MQTT over WebSocket
- OpenWire
- STOMP
- STOMP over WebSocket

Attributes

An ActiveMQ broker has several attributes, for example:

- A name (`MyBroker`)
- An ID (`b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9`)

- An Amazon Resource Name (ARN) (`arn:aws:mq:us-east-2:123456789012:broker:MyBroker:b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9`)
- An ActiveMQ Web Console URL (`https://b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-east-2.amazonaws.com:8162`)

  For more information, see Web Console in the Apache ActiveMQ documentation.

  Important

  If you specify an authorization map which doesn't include the `activemq-webconsole` group, you can't use the ActiveMQ Web Console because the group isn't authorized to send messages to, or receive messages from, the Amazon MQ broker.

- Wire-level protocol endpoints:
  - `amqp+ssl://b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-east-2.amazonaws.com:5671`
  - `mqtt+ssl://b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-east-2.amazonaws.com:8883`
  - `ssl://b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-east-2.amazonaws.com:61617`

    Note

    This is an OpenWire endpoint.

  - `stomp+ssl://b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-east-2.amazonaws.com:61614`
  - `wss://b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-east-2.amazonaws.com:61619`

  For more information, see Configuring Transports in the Apache ActiveMQ documentation.

Note

For an active/standby broker, Amazon MQ provides two ActiveMQ Web Console URLs, but only one URL is active at a time. Likewise, Amazon MQ provides two endpoints for each

wire-level protocol, but only one endpoint is active in each pair at a time. The `-1` and `-2` suffixes denote a redundant pair.

For a full list of broker attributes, see the following in the *Amazon MQ REST API Reference*:

- REST Operation ID: Broker
- REST Operation ID: Brokers
- REST Operation ID: Broker Reboot

Broker users

An ActiveMQ *user* is a person or an application that can access the queues and topics of an ActiveMQ broker. You can configure users to have specific permissions. For example, you can allow some users to access the ActiveMQ Web Console.

A *group* is a semantic label. You can assign a group to a user and configure permissions for groups to send to, receive from, and administer specific queues and topics.

Important

Making changes to a user does *not* apply the changes to the user immediately. To apply your changes, you must wait for the next maintenance window or reboot the broker.

For information about users and groups, see the following in the Apache ActiveMQ documentation:

- Authorization
- Authorization Example

For information about creating, editing, and deleting ActiveMQ users, see the following:

- Creating an ActiveMQ broker user
- Users

User attributes

For a full list of user attributes, see the following in the *Amazon MQ REST API Reference*:

- REST Operation ID: User
- REST Operation ID: Users

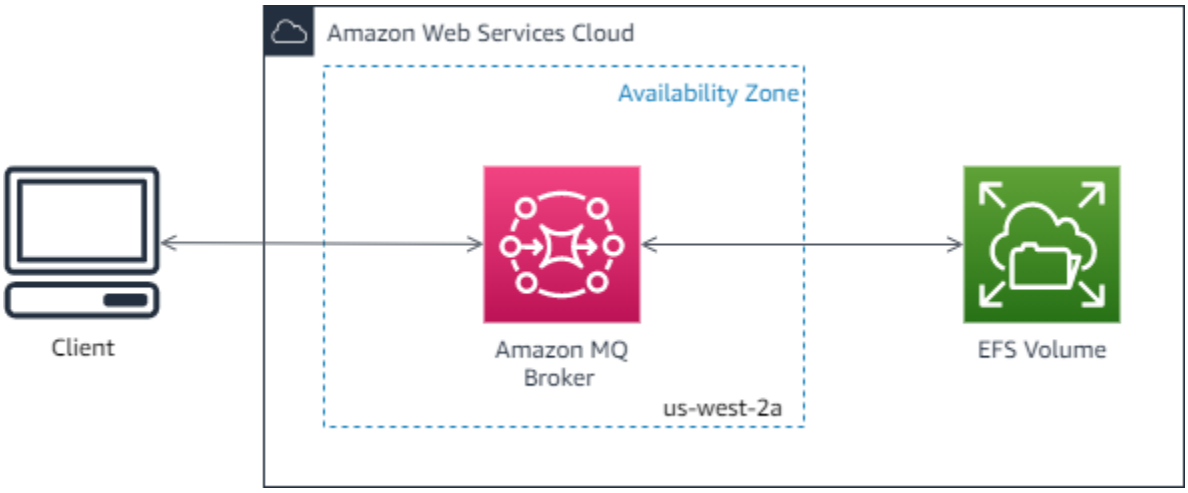
Deployment options for Amazon MQ for ActiveMQ brokers

Amazon MQ offers single instance and cluster deployment options for brokers.

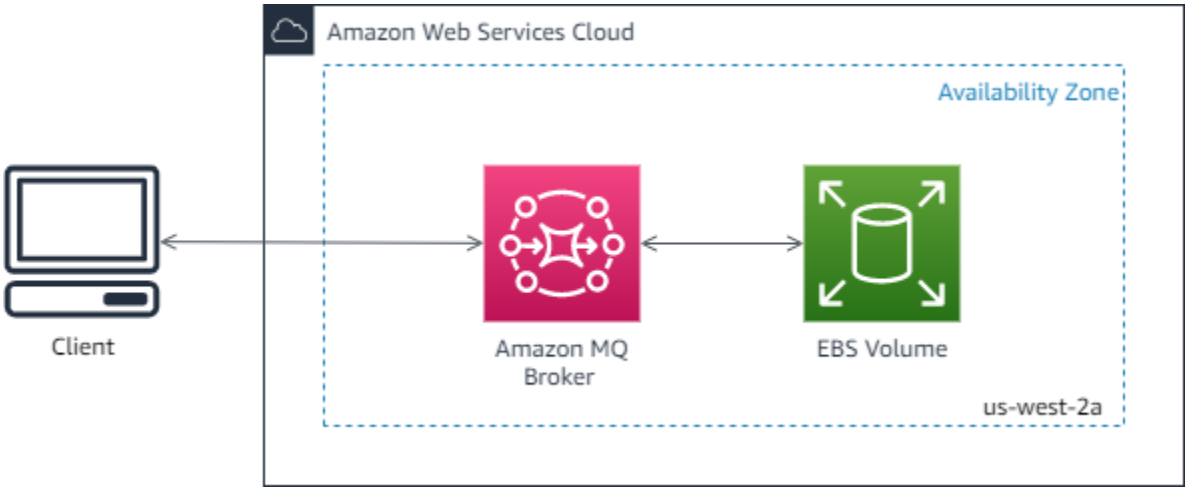
Option 1: Amazon MQ single-instance brokers

A *single-instance broker* is comprised of one broker in one Availability Zone. The broker communicates with your application and with an Amazon EBS or Amazon EFS storage volume. Amazon EFS storage volumes are designed to provide the highest level of durability and availability by storing data redundantly across multiple Availability Zones (AZs). Amazon EBS provides block level storage optimized for low-latency and high throughput. For more information about storage options, see Storage.

The following diagram illustrates a single-instance broker with Amazon EFS storage replicated across multiple AZs.



The following diagram illustrates a single-instance broker with Amazon EBS storage replicated across multiple servers within a single AZ.



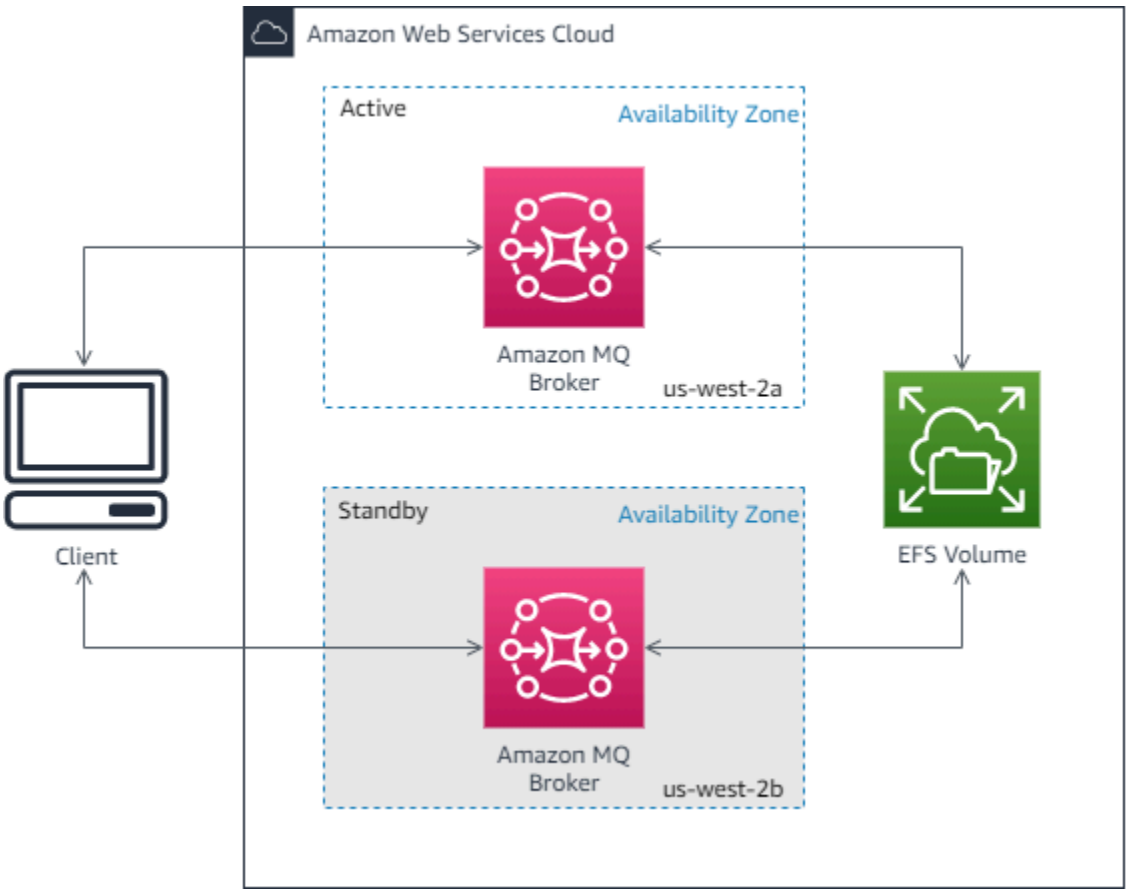
Option 2: Amazon MQ active/standby brokers for high availability

An *active/standby broker* is comprised of two brokers in two different Availability Zones, configured in a *redundant pair*. These brokers communicate synchronously with your application, and with Amazon EFS. Amazon EFS storage volumes are designed to provide the highest level of durability, and availability by storing data redundantly across multiple Availability Zones (AZs). For more information, see Storage.

Usually, only one of the broker instances is active at any time, while the other broker instance is on standby. If one of the broker instances malfunctions or undergoes maintenance, it takes Amazon MQ a short while to take the inactive instance out of service. This allows the healthy standby instance to become active and to begin accepting incoming communications. Maintencance windows and broker reboots you initate will cause a failover to happen. When you reboot a broker, the failover takes only a few seconds.

For an active/standby broker, Amazon MQ provides two ActiveMQ Web Console URLs, but only one URL is active at a time. Likewise, Amazon MQ provides two endpoints for each wire-level protocol, but only one endpoint is active in each pair at a time. The `-1` and `-2` suffixes denote a redundant pair. For wire-level protocol endpoints, you should allow your application to connect to either endpoint by using the Failover Transport.

The following diagram illustrates an active/standby broker with Amazon EFS storage replicated across multiple AZs.



Amazon MQ network of brokers

Amazon MQ supports ActiveMQ's network of brokers feature.

A network of brokers is comprised of multiple simultaneously active single-instance brokers or active/standby brokers. Creating a network of brokers can increase availability, fault tolerance, and load balancing with multiple broker instances.

How does a Network of Brokers work?

A network of brokers is established by connecting one broker to another using *network connectors*. A network connector provides on-demand message from one broker to another. Network connectors are configured in the broker configuration as either *non-duplex* or *duplex* connections. For non-duplex connections, messages are forwarded only from one broker to the other. For duplex connections, messages are forwarded both ways between both brokers.

If the network connector is configured as duplex, messages are also forwarded from *Broker2* to *Broker1*.

You can use both non-duplex and duplex connections in a network of brokers. You may want to introduce a duplex connection to another broker to improve traffic, or to avoid a limit increase. Duplex connections are also useful for partial migration from on-premises to Amazon MQ managed brokers.

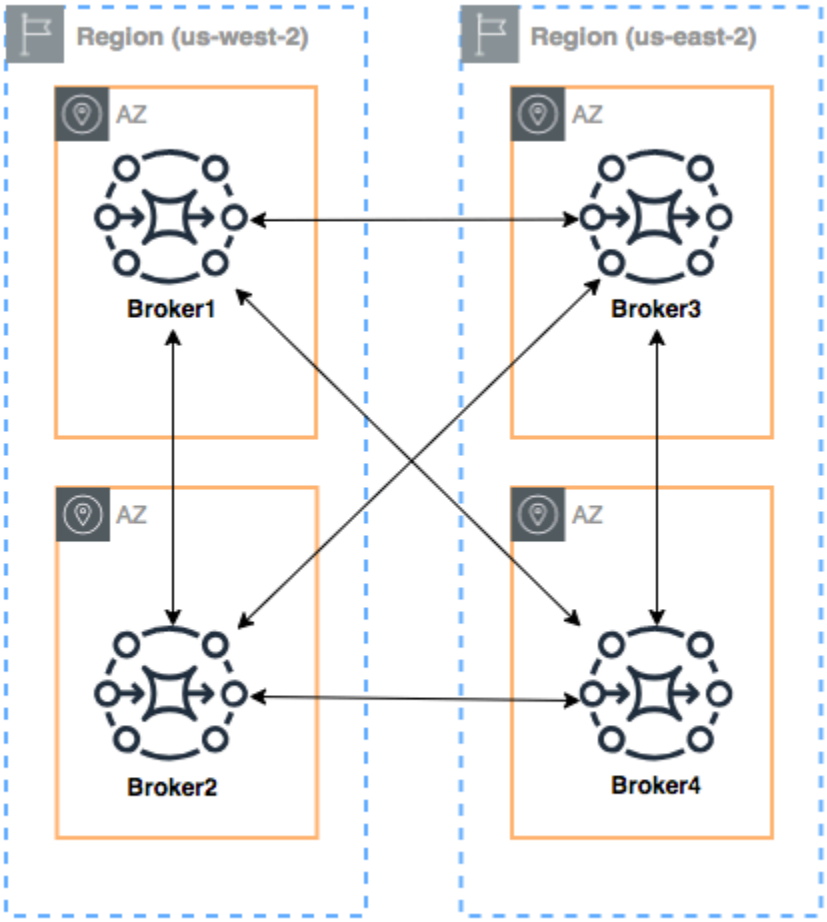
How Does a Network of Brokers Handle Credentials?

For broker A to connect to broker B in a network, broker A must use valid credentials, like any other producer or consumer. Instead of providing a password in broker A's `<networkConnector>` configuration, you must first create a user on broker A with the same values as another user on broker B (these are *separate, unique* users that share the same username and password values). When you specify the `userName` attribute in the `<networkConnector>` configuration, Amazon MQ will add the password automatically at runtime.

> **Important**
>
> Don't specify the `password` attribute for the `<networkConnector>`. We don't recommend storing plaintext passwords in broker configuration files, because this makes the passwords visible in the Amazon MQ console. For more information, see Configure Network Connectors for Your Broker.

Cross region

To configure a network of brokers that spans AWS regions, deploy brokers in those regions, and configure network connectors to the endpoints of those brokers.



To configure a network of brokers like this example, you could add `networkConnectors` entries to the configurations of *Broker1* and *Broker4* that reference the wire-level endpoints of those brokers.

Network connectors for Broker1:

```
<networkConnectors>
    <networkConnector name="1_to_2" userName="myCommonUser" duplex="true"
        uri="static:(ssl://b-9876l5k4-32ji-109h-8gfe-7d65c4b132a1-2.mq.us-
west-2.amazonaws.com:61617)"/>
    <networkConnector name="1_to_3" userName="myCommonUser" duplex="true"
        uri="static:(ssl://b-743c885d-2244-4c95-af67-a85017ff234e-3.mq.us-
east-2.amazonaws.com:61617)"/>
    <networkConnector name="1_to_4" userName="myCommonUser" duplex="true"
        uri="static:(ssl://b-62a7fb31-d51c-466a-a873-905cd660b553-4.mq.us-
east-2.amazonaws.com:61617)"/>
```

```
</networkConnectors>
```

Network connector for Broker2:

```
<networkConnectors>
    <networkConnector name="2_to_3" userName="myCommonUser" duplex="true"
        uri="static:(ssl://b-743c885d-2244-4c95-af67-a85017ff234e-3.mq.us-
east-2.amazonaws.com:61617)"/>
</networkConnectors>
```

Network connectors for Broker4:

```
<networkConnectors>
    <networkConnector name="4_to_3" userName="myCommonUser" duplex="true"
        uri="static:(ssl://b-743c885d-2244-4c95-af67-a85017ff234e-3.mq.us-
east-2.amazonaws.com:61617)"/>
    <networkConnector name="4_to_2" userName="myCommonUser" duplex="true"
        uri="static:(ssl://b-9876l5k4-32ji-109h-8gfe-7d65c4b132a1-2.mq.us-
west-2.amazonaws.com:61617)"/>
</networkConnectors>
```

Dynamic Failover With Transport Connectors

In addition to configuring `networkConnector` elements, you can configure your broker `transportConnector` options to enable dynamic failover, and to rebalance connections when brokers are added or removed from the network.

```
<transportConnectors>
  <transportConnector name="openwire" updateClusterClients="true"
 rebalanceClusterClients="true" updateClusterClientsOnRemove="true"/>
</transportConnectors>
```

In this example both `updateClusterClients` and `rebalanceClusterClients` are set to `true`. In this case clients will be provided a list of brokers in the network, and will request them to rebalance if a new broker joins.

Available options:

- `updateClusterClients`: Passes information to clients about changes in the network of broker topology.

- `rebalanceClusterClients`: Causes clients to re-balance across brokers when a new broker is added to a network of brokers.
- `updateClusterClientsOnRemove`: Updates clients with topology information when a broker leaves a network of brokers.

When `updateClusterClients` is set to true, clients can be configured to connect to a single broker in a network of brokers.

```
failover:(ssl://b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-
east-2.amazonaws.com:61617)
```

When a new broker connects, it will receive a list of URIs of all brokers in the network. If the connection to the broker fails, it can dynamically switch to one of the brokers provided when it connected.

For more information on failover, see Broker-side Options for Failover in the Active MQ documentation.

Amazon MQ for ActiveMQ broker instance types

The combined description of the broker instance *class* (`m5`) and *size* (`large`, `medium`) is called the *broker instance type* (for example, `mq.m5.large`). The following table lists the available Amazon MQ broker instance types for ActiveMQ brokers.

Amazon MQ provides at least a 90 day notice before an instance type reaches end of support. We recommend upgrading your broker to a new instance type before the end-of-support date to prevent any disruptions.

> **Important**
>
> You cannot create brokers on `t2.micro` or `mq.m4.large` after March 17, 2025.

Instance Type	vCPU	Memory (GiB)	Recommended Use	Storage	End of support on Amazon MQ
mq.t3.micro	2	1	Evaluation	EFS	
mq.m5.large	2	8	Production	EFS or EBS	
mq.m5.xlarge	4	16	Production	EFS or EBS	
mq.m5.2xlarge	8	32	Production	EFS or EBS	
mq.m5.4xlarge	16	64	Production	EFS or EBS	

For more information about throughput considerations, see Choose the Correct Broker Instance Type for the Best Throughput.

Amazon MQ for ActiveMQ broker configurations

A configuration contains all of the settings for your ActiveMQ broker in XML format (similar to ActiveMQ's `activemq.xml` file). You can create a configuration before creating any brokers. You can then apply the configuration to one or more brokers.

Important

Making changes to a configuration does *not* apply the changes to the broker immediately. To apply your changes, you must wait for the next maintenance window or reboot the broker.

You can only **delete** a configuration using the `DeleteConfiguration` API. For more information, see Configurations in the *Amazon MQ API Reference*.

Attributes

A broker configuration has several attributes, for example:

- A name (`MyConfiguration`)
- An ID (`c-1234a5b6-78cd-901e-2fgh-3i45j6k178l9`)
- An Amazon Resource Name (ARN) (`arn:aws:mq:us-east-2:123456789012:configuration:c-1234a5b6-78cd-901e-2fgh-3i45j6k178l9`)

For a full list of configuration attributes, see the following in the *Amazon MQ REST API Reference*:

- REST Operation ID: Configuration
- REST Operation ID: Configurations

For a full list of configuration revision attributes, see the following:

- REST Operation ID: Configuration Revision
- REST Operation ID: Configuration Revisions

Using Spring XML configuration files

ActiveMQ brokers are configured using Spring XML files. You can configure many aspects of your ActiveMQ broker, such as predefined destinations, destination policies, authorization policies, and plugins. Amazon MQ controls some of these configuration elements, such as network transports and storage. Other configuration options, such as creating networks of brokers, aren't currently supported.

The full set of supported configuration options is specified in the Amazon MQ XML schemas. Download zip files of the supported schemas using the following links.

- `amazon-mq-active-mq-5.19.1.xsd.zip`
- `amazon-mq-active-mq-5.18.4.xsd.zip`
- `amazon-mq-active-mq-5.17.6.xsd.zip`
- `amazon-mq-active-mq-5.16.7.xsd.zip`
- `amazon-mq-active-mq-5.15.16.xsd.zip`

You can use these schemas to validate and sanitize your configuration files. Amazon MQ also lets you provide configurations by uploading XML files. When you upload an XML file, Amazon MQ automatically sanitizes and removes invalid and prohibited configuration parameters according to the schema.

> **Note**
>
> You can use only static values for attributes. Amazon MQ sanitizes elements and attributes that contain Spring expressions, variables, and element references from your configuration.

Creating an Amazon MQ for ActiveMQ broker configuration

A *configuration* contains all of the settings for your ActiveMQ broker, in XML format (similar to ActiveMQ's `activemq.xml` file). You can create a configuration before creating any brokers. You can then apply the configuration to one or more brokers. You can apply a configuration immediately or during a *maintenance window*.

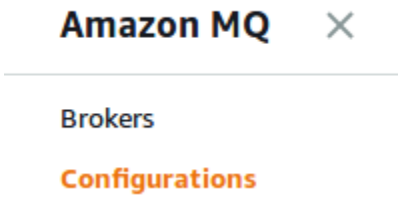
The following example shows how you can create and apply an Amazon MQ broker configuration using the AWS Management Console.

> **Important**
>
> You can only **delete** a configuration using the `DeleteConfiguration` API. For more information, see Configurations in the *Amazon MQ API Reference*.

Create a New Configuration

To create a new broker configuration, first create the new configuration.

1. Sign in to the Amazon MQ console.
2. On the left, expand the navigation panel and choose **Configurations**.



3. On the **Configurations** page, choose **Create configuration**.
4. On the **Create configuration** page, in the **Details** section, type the **Configuration name** (for example, `MyConfiguration`) and select a **Broker engine** version.

   > **Note**
   >
   > To learn more about ActiveMQ engine versions supported by Amazon MQ for ActiveMQ, see the section called “Version management”.

5. Choose **Create configuration**.

Create a New Configuration Revision

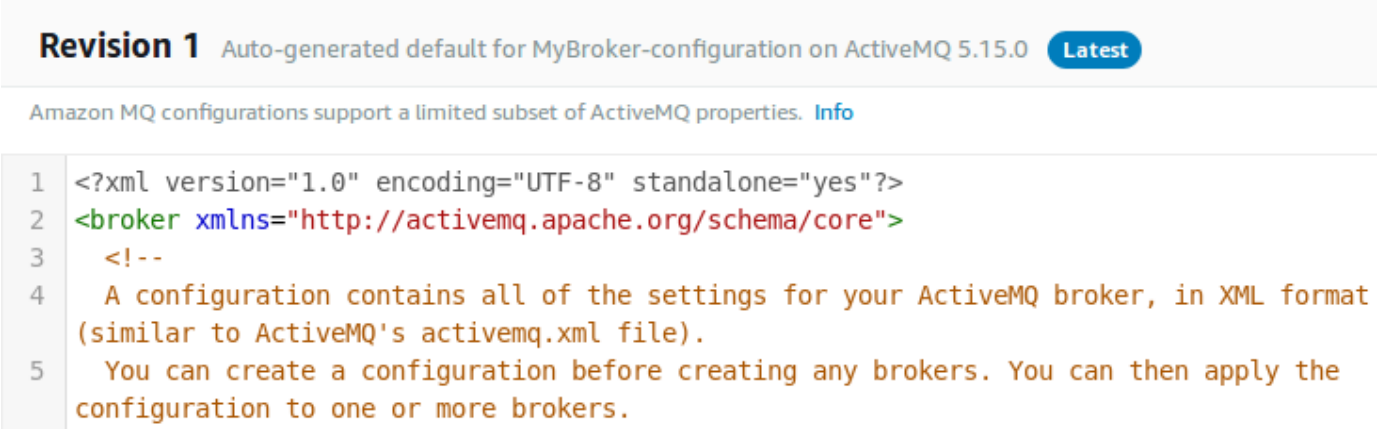
After you create a broker configuration, you will need to edit the configuration using a configuration revision.

1. From the configuration list, choose ***`MyConfiguration`***.

   > **Note**
   >
   > The first configuration revision is always created for you when Amazon MQ creates the configuration.

   On the ***`MyConfiguration`*** page, the broker engine type and version that your new configuration revision uses (for example, **Apache ActiveMQ 5.15.16**) are displayed.
2. On the **Configuration details** tab, the configuration revision number, description, and broker configuration in XML format are displayed.

   > **Note**
   >
   > Editing the current configuration creates a new configuration revision.



3. Choose **Edit configuration** and make changes to the XML configuration.
4. Choose **Save**.

   The **Save revision** dialog box is displayed.
5. (Optional) Type `A description of the changes in this revision`.
6. Choose **Save**.

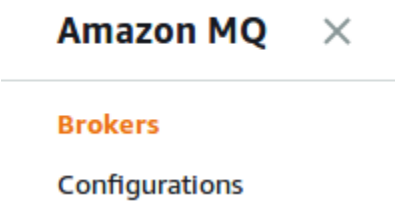
   The new revision of the configuration is saved.

> **Important**
>
> The Amazon MQ console automatically sanitizes invalid and prohibited configuration parameters according to a schema. For more information and a full list of permitted XML parameters, see Amazon MQ Broker Configuration Parameters.

Apply a Configuration Revision to Your Broker

After revising the configuration, you can apply the configuration revision to your broker.

1. On the left, expand the navigation panel and choose **Brokers**.



2. From the broker list, select your broker (for example, **MyBroker**) and then choose **Edit**.
3. On the **Edit *MyBroker*** page, in the **Configuration** section, select a **Configuration** and a **Revision** and then choose **Schedule Modifications**.
4. In the **Schedule broker modifications** section, choose whether to apply modifications **During the next scheduled maintenance window** or **Immediately**.

   > **Important**
   >
   > Single instance brokers are offline while being rebooted. For cluster brokers, only one node is down at a time while the broker reboots.

5. Choose **Apply**.

   Your configuration revision is applied to your broker at the specified time.

Edit an Amazon MQ for ActiveMQ configuration revision

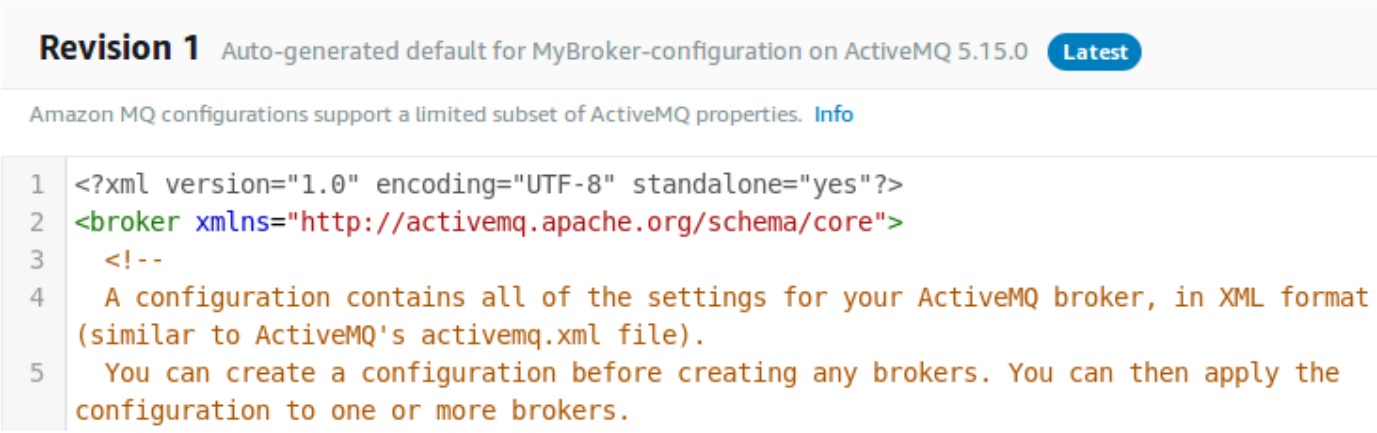
You may want to edit a configuration revision after applying it to your broker. Use the following instructions to edit a configuration revision.

1. Sign in to the Amazon MQ console.
2. From the broker list, select your broker (for example, **MyBroker**) and then choose **Edit**.
3. On the ***MyBroker*** page, choose **Edit**.
4. On the **Edit *MyBroker*** page, in the **Configuration** section, select a **Configuration** and a **Revision** and then choose **Edit**.

   > **Note**
   >
   > Unless you select a configuration when you create a broker, the first configuration revision is always created for you when Amazon MQ creates the broker.

   On the ***MyBroker*** page, the broker engine type and version that the configuration uses (for example, **Apache ActiveMQ 5.15.8**) are displayed.
5. On the **Configuration details** tab, the configuration revision number, description, and broker configuration in XML format are displayed.

> **Note**
>
> Editing the current configuration creates a new configuration revision.



6. Choose **Edit configuration** and make changes to the XML configuration.
7. Choose **Save**.

   The **Save revision** dialog box is displayed.
8. (Optional) Type `A description of the changes in this revision`.
9. Choose **Save**.

   The new revision of the configuration is saved.

> **Important**
>
> The Amazon MQ console automatically sanitizes invalid and prohibited configuration parameters according to a schema. For more information and a full list of permitted XML parameters, see Amazon MQ Broker Configuration Parameters.

Elements permitted in Amazon MQ configurations

The following is a detailed listing of the elements permitted in Amazon MQ configurations. For more information, see XML Configuration in the Apache ActiveMQ documentation.

Element
abortSlowAckConsumerStrategy (attributes)
abortSlowConsumerStrategy (attributes)
authorizationEntry (attributes)
authorizationMap (child collection elements)
authorizationPlugin (child collection elements)
broker (attributes | child collection elements)
cachedMessageGroupMapFactory (attributes)
compositeQueue (attributes | child collection elements)
compositeTopic (attributes | child collection elements)
constantPendingMessageLimitStrategy (attributes)
discarding (attributes)
discardingDLQBrokerPlugin (attributes)
fileCursor
fileDurableSubscriberCursor
fileQueueCursor
filteredDestination (attributes)
fixedCountSubscriptionRecoveryPolicy (attributes)
fixedSizedSubscriptionRecoveryPolicy (attributes)
forcePersistencyModeBrokerPlugin (attributes)
individualDeadLetterStrategy (attributes)

Element
lastImageSubscriptionRecoveryPolicy
messageGroupHashBucketFactory (attributes)
mirroredQueue (attributes)
noSubscriptionRecoveryPolicy
oldestMessageEvictionStrategy (attributes)
oldestMessageWithLowestPriorityEvictionStrategy (attributes)
policyEntry (attributes | child collection elements)
policyMap (child collection elements)
prefetchRatePendingMessageLimitStrategy (attributes)
priorityDispatchPolicy
priorityNetworkDispatchPolicy
queryBasedSubscriptionRecoveryPolicy (attributes)
queue (attributes)
redeliveryPlugin (attributes | child collection elements)
redeliveryPolicy (attributes)
redeliveryPolicyMap (child collection elements)
retainedMessageSubscriptionRecoveryPolicy (child collection elements)
roundRobinDispatchPolicy
sharedDeadLetterStrategy (attributes | child collection elements)
simpleDispatchPolicy

Element
simpleMessageGroupMapFactory
statisticsBrokerPlugin
storeCursor
storeDurableSubscriberCursor (attributes)
strictOrderDispatchPolicy
tempDestinationAuthorizationEntry (attributes)
tempQueue (attributes)
tempTopic (attributes)
timedSubscriptionRecoveryPolicy (attributes)
timeStampingBrokerPlugin (attributes)
topic (attributes)
transportConnector (attributes)
uniquePropertyMessageEvictionStrategy (attributes)
virtualDestinationInterceptor (child collection elements)
virtualTopic (attributes)
vmCursor
vmDurableCursor
vmQueueCursor

Elements and Their Attributes Permitted in Amazon MQ Configurations

The following is a detailed listing of the elements and their attributes permitted in Amazon MQ configurations. For more information, see XML Configuration in the Apache ActiveMQ documentation.

Element	Attribute
abortSlowAckConsumerStrategy	abortConnection
	checkPeriod
	ignoreIdleConsumers
	ignoreNetworkConsumers
	maxSlowCount
	maxSlowDuration
	maxTimeSinceLastAck
	name
abortSlowConsumerStrategy	abortConnection
	checkPeriod
	ignoreNetworkConsumers
	maxSlowCount
	maxSlowDuration
	name
authorizationEntry	admin
	queue
	read

Element	Attribute
	tempQueue
	tempTopic
	topic
	write
broker	advisorySupport
	allowTempAutoCreationOnSend
	cacheTempDestinations
	consumerSystemUsagePortion
	dedicatedTaskRunner
	deleteAllMessagesOnStartup
	keepDurableSubsActive
	enableMessageExpirationOnActiveDurableSubs
	maxPurgedDestinationsPerSweep
	maxSchedulerRepeatAllowed
	monitorConnectionSplits
	networkConnectorStartAsync
	offlineDurableSubscriberTaskSchedule
	offlineDurableSubscriberTimeout
	persistenceThreadPriority

Element	Attribute
	persistent
	populateJMSXUserID
	producerSystemUsagePortion
	rejectDurableConsumers
	rollbackOnlyOnAsyncException
	schedulePeriodForDestinatio nPurge
	schedulerSupport
	splitSystemUsageForProducer sConsumers
	taskRunnerPriority
	timeBeforePurgeTempDestinations
	useAuthenticatedPrincipalFo rJMSXUserID
	useMirroredQueues
	useTempMirroredQueues
	useVirtualDestSubs
	useVirtualDestSubsOnCreation
	useVirtualTopics
cachedMessageGroupMapFactory	cacheSize
compositeQueue	concurrentSend

Element	Attribute
	copyMessage
	forwardOnly
	name
	sendWhenNotMatched
compositeTopic	concurrentSend
	copyMessage
	forwardOnly
	name
	sendWhenNotMatched
conditionalNetworkBridgeFilterFactory	rateDuration
	rateLimit
	replayDelay
	replayWhenNoConsumers
	selectorAware **Supported in** Apache ActiveMQ 5.16.x
constantPendingMessageLimitStrategy	limit
discarding	deadLetterQueue
	enableAudit

Element	Attribute
	expiration
	maxAuditDepth
	maxProducersToAudit
	processExpired
	processNonPersistent
discardingDLQBrokerPlugin	dropAll
	dropOnly
	dropTemporaryQueues
	dropTemporaryTopics
	reportInterval
filteredDestination	queue
	selector
	topic
fixedCountSubscriptionRecoveryPolicy	maximumSize
fixedSizedSubscriptionRecoveryPolicy	maximumSize
	useSharedBuffer
forcePersistencyModeBrokerPlugin	persistenceFlag
individualDeadLetterStrategy	destinationPerDurableSubscriber
	enableAudit
	expiration

Element	Attribute
	maxAuditDepth
	maxProducersToAudit
	processExpired
	processNonPersistent
	queuePrefix
	queueSuffix
	topicPrefix
	topicSuffix
	useQueueForQueueMessages
	useQueueForTopicMessages
messageGroupHashBucketFactory	bucketCount
	cacheSize
mirroredQueue	copyMessage
	postfix
	prefix
oldestMessageEvictionStrategy	evictExpiredMessagesHighWat ermark
oldestMessageWithLowestPrio rityEvictionStrategy	evictExpiredMessagesHighWat ermark
policyEntry	advisoryForConsumed
	advisoryForDelivery

Element	Attribute
	advisoryForDiscardingMessages
	advisoryForFastProducers
	advisoryForSlowConsumers
	advisoryWhenFull
	allConsumersExclusiveByDefault
	alwaysRetroactive
	blockedProducerWarningInterval
	consumersBeforeDispatchStarts
	cursorMemoryHighWaterMark
	doOptimzeMessageStorage
	durableTopicPrefetch
	enableAudit
	expireMessagesPeriod
	gcInactiveDestinations
	gcWithNetworkConsumers
	inactiveTimeoutBeforeGC
	inactiveTimoutBeforeGC
	includeBodyForAdvisory
	lazyDispatch
	maxAuditDepth

Element	Attribute
	maxBrowsePageSize
	maxDestinations
	maxExpirePageSize
	maxPageSize
	maxProducersToAudit
	maxQueueAuditDepth
	memoryLimit
	messageGroupMapFactoryType
	minimumMessageSize
	optimizedDispatch
	optimizeMessageStoreInFligh tLimit
	persistJMSRedelivered
	prioritizedMessages
	producerFlowControl
	queue
	queueBrowserPrefetch
	queuePrefetch
	reduceMemoryFootprint
	sendAdvisoryIfNoConsumers
	sendFailIfNoSpace

Element	Attribute
	sendFailIfNoSpaceAfterTimeout **Supported in** Apache ActiveMQ 5.16.4 and above
	sendDuplicateFromStoreToDLQ
	storeUsageHighWaterMark
	strictOrderDispatch
	tempQueue
	tempTopic
	timeBeforeDispatchStarts
	topic
	topicPrefetch
	useCache
	useConsumerPriority
usePrefetchExtension	
prefetchRatePendingMessageLimitStrategy	multiplier
queryBasedSubscriptionRecoveryPolicy	query
queue	DLQ
	physicalName

Element	Attribute
redeliveryPlugin	fallbackToDeadLetter
	sendToDlqIfMaxRetriesExceeded
redeliveryPolicy	backOffMultiplier
	collisionAvoidancePercent
	initialRedeliveryDelay
	maximumRedeliveries
	maximumRedeliveryDelay
	preDispatchCheck
	queue
	redeliveryDelay
	tempQueue
	tempTopic
	topic
	useCollisionAvoidance
	useExponentialBackOff
sharedDeadLetterStrategy	enableAudit
	expiration
	maxAuditDepth
	maxProducersToAudit
	processExpired

Element	Attribute
	processNonPersistent
storeDurableSubscriberCursor	immediatePriorityDispatch
	useCache
tempDestinationAuthorizatio nEntry	admin
	queue
	read
	tempQueue
	tempTopic
	topic
	write
tempQueue	DLQ
	physicalName
tempTopic	DLQ
	physicalName
timedSubscriptionRecoveryPolicy	zeroExpirationOverride
timeStampingBrokerPlugin	recoverDuration
	futureOnly
	processNetworkMessages
	ttlCeiling
topic	DLQ

Element	Attribute
	physicalName
transportConnector	name
	updateClusterClients
	rebalanceClusterClients
	updateClusterClientsOnRemove
uniquePropertyMessageEvictionStrategy	evictExpiredMessagesHighWatermark
	propertyName
virtualTopic	concurrentSend
	local
	dropOnResourceLimit
	name
	postfix
	prefix
	selectorAware
	setOriginalDestination
	transactedSend

Amazon MQ Parent Element Attributes

The following is a detailed explanation of parent element attributes. For more information, see XML Configuration in the Apache ActiveMQ documentation.

Topics

- broker

broker

`broker` is a parent collection element.

Attributes

networkConnectionStartAsync

To mitigate network latency and to allow other networks to start in a timely manner, use the `<networkConnectionStartAsync>` tag. The tag instructs the broker to use an executor to start network connections in parallel, asynchronous to a broker start.

Default: `false`

Example Configuration

```
<broker networkConnectorStartAsync="false"/>
```

Elements, Child Collection Elements, and Their Child Elements Permitted in Amazon MQ Configurations

The following is a detailed listing of the elements, child collection elements, and their child elements permitted in Amazon MQ configurations. For more information, see XML Configuration in the Apache ActiveMQ documentation.

Element	Child Collection Element	Child Element
`authorizationMap`	`authorizationEntries`	`authorizationEntry`
		`tempDestinationAuthorizationEntry`
	`defaultEntry`	`authorizationEntry`
		`tempDestinationAuthorizationEntry`

Element	Child Collection Element	Child Element
	tempDestinationAuthorizationEntry	tempDestinationAuthorizationEntry
authorizationPlugin	map	authorizationMap
broker	destinationInterceptors	mirroredQueue
		virtualDestinationInterceptor
	destinationPolicy	policyMap
	destinations	queue
		tempQueue
		tempTopic
		topic
	networkConnectors	networkConnector
	persistenceAdapter	kahaDB
	plugins	authorizationPlugin
		discardingDLQBrokerPlugin
		forcePersistencyModeBrokerPlugin
		redeliveryPlugin
		statisticsBrokerPlugin
		timeStampingBrokerPlugin

Element	Child Collection Element	Child Element
	systemUsage	systemUsage
	transportConnector	name
		updateClusterClients
		rebalanceClusterClients
		updateClusterClientsOnRemove
compositeQueue	forwardTo	queue
		tempQueue
		tempTopic
		topic
		filteredDestination
compositeTopic	forwardTo	queue
		tempQueue
		tempTopic
		topic
		filteredDestination
policyEntry	deadLetterStrategy	discarding
		individualDeadLetterStrategy
		sharedDeadLetterStrategy

Element	Child Collection Element	Child Element
	destination	queue
		tempQueue
		tempTopic
		topic
	dispatchPolicy	priorityDispatchPolicy
		priorityNetworkDispatchPolicy
		roundRobinDispatchPolicy
		simpleDispatchPolicy
		strictOrderDispatchPolicy
		clientIdFilterDispatchPolicy
	messageEvictionStrategy	oldestMessageEvictionStrategy
		oldestMessageWithLowestPriorityEvictionStrategy
		uniquePropertyMessageEvictionStrategy
	messageGroupMapFactory	cachedMessageGroupMapFactory

Element	Child Collection Element	Child Element
		messageGroupHashBucketFactory
		simpleMessageGroupMapFactory
	pendingDurableSubscriberPolicy	fileDurableSubscriberCursor
		storeDurableSubscriberCursor
		vmDurableCursor
	pendingMessageLimitStrategy	constantPendingMessageLimitStrategy
		prefetchRatePendingMessageLimitStrategy
	pendingQueuePolicy	fileQueueCursor
		storeCursor
		vmQueueCursor
	pendingSubscriberPolicy	fileCursor
		vmCursor
	slowConsumerStrategy	abortSlowAckConsumerStrategy
		abortSlowConsumerStrategy

Element	Child Collection Element	Child Element
	subscriptionRecoveryPolicy	fixedCountSubscriptionRecoveryPolicy
		fixedSizedSubscriptionRecoveryPolicy
		lastImageSubscriptionRecoveryPolicy
		noSubscriptionRecoveryPolicy
		queryBasedSubscriptionRecoveryPolicy
		retainedMessageSubscriptionRecoveryPolicy
timedSubscriptionRecoveryPolicy		
policyMap	defaultEntry	policyEntry
	policyEntries	policyEntry
redeliveryPlugin	redeliveryPolicyMap	redeliveryPolicyMap
redeliveryPolicyMap	defaultEntry	redeliveryPolicy
	redeliveryPolicyEntries	redeliveryPolicy
retainedMessageSubscriptionRecoveryPolicy	wrapped	fixedCountSubscriptionRecoveryPolicy
		fixedSizedSubscriptionRecoveryPolicy

Element	Child Collection Element	Child Element
		lastImageSubscriptionRecoveryPolicy
		noSubscriptionRecoveryPolicy
		queryBasedSubscriptionRecoveryPolicy
		retainedMessageSubscriptionRecoveryPolicy
		timedSubscriptionRecoveryPolicy
sharedDeadLetterStrategy	deadLetterQueue	queue
		tempQueue
		tempTopic
		topic
virtualDestinationInterceptor	virtualDestinations	compositeQueue
		compositeTopic
		virtualTopic

Amazon MQ Child Element Attributes

The following is a detailed explanation of child element attributes. For more information, see XML Configuration in the Apache ActiveMQ documentation.

Topics

- authorizationEntry

- networkConnector
- kahaDB
- systemUsage

authorizationEntry

`authorizationEntry` is a child of the `authorizationEntries` child collection element.

Attributes

admin|read|write

The permissions granted to a group of users. For more information, see Always configure an authorization map.

If you specify an authorization map which doesn't include the `activemq-webconsole` group, you can't use the ActiveMQ Web Console because the group isn't authorized to send messages to, or receive messages from, the Amazon MQ broker.

Default: `null`

Example Configuration

```
<authorizationPlugin>
                                <map>
                                    <authorizationMap>
                                        <authorizationEntries>
                                            <authorizationEntry admin="admins,activemq-
webconsole" read="admins,users,activemq-webconsole" write="admins,activemq-webconsole"
 queue=">"/>
                                            <authorizationEntry admin="admins,activemq-
webconsole" read="admins,users,activemq-webconsole" write="admins,activemq-webconsole"
 topic=">"/>
                                        </authorizationEntries>
                                    </authorizationMap>
                                </map>
                                </authorizationPlugin>
```

> **Note**
>
> The `activemq-webconsole` group in ActiveMQ on Amazon MQ has admin permissions on all queues and topics. All users in this group will have admin access.

networkConnector

`networkConnector` is a child of the `networkConnectors` child collection element.

Topics

- Attributes
- Example Configurations

Attributes

conduitSubscriptions

Specifies whether a network connection in a network of brokers treats multiple consumers subscribed to the same destination as one consumer. For example, if `conduitSubscriptions` is set to `true` and two consumers connect to broker B and consume from a destination, broker B combines the subscriptions into a single logical subscription over the network connection to broker A, so that only a single copy of a message is forwarded from broker A to broker B.

> **Note**
>
> Setting `conduitSubscriptions` to `true` can reduce redundant network traffic. However, using this attribute can have implications for the load-balancing of messages across consumers and might cause incorrect behavior in certain scenarios (for example, with JMS message selectors or with durable topics).

Default: `true`

duplex

Specifies whether the connection in the network of brokers is used to produce *and* consume messages. For example, if broker A creates a connection to broker B in non-duplex mode, messages

can be forwarded only from broker A to broker B. However, if broker A creates a duplex connection to broker B, then broker B can forward messages to broker A without having to configure a `<networkConnector>`.

Default: `false`

name

The name of the bridge in the network of brokers.

Default: `bridge`

uri

The wire-level protocol endpoint for one of two brokers (or for multiple brokers) in a network of brokers.

Default: `null`

username

The username common to the brokers in a network of brokers.

Default: `null`

Example Configurations

> **Note**
>
> When using a `networkConnector` to define a network of brokers, don't include the password for the user common to your brokers.

A Network of Brokers with Two Brokers

In this configuration, two brokers are connected in a network of brokers. The name of the network connector is `connector_1_to_2`, the username common to the brokers is `myCommonUser`, the connection is `duplex`, and the OpenWire endpoint URI is prefixed by `static:`, indicating a one-to-one connection between the brokers.

```
<networkConnectors>
```

```
                                    <networkConnector name="connector_1_to_2"
 userName="myCommonUser" duplex="true"
                                        uri="static:(ssl://
b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-east-2.amazonaws.com:61617)"/>
                                    </networkConnectors>
```

For more information, see Configure Network Connectors for Your Broker.

A Network of Brokers with Multiple Brokers

In this configuration, multiple brokers are connected in a network of brokers. The name of the network connector is `connector_1_to_2`, the username common to the brokers is `myCommonUser`, the connection is `duplex`, and the comma-separated list of OpenWire endpoint URIs is prefixed by `masterslave:`, indicating a failover connection between the brokers. The failover from broker to broker isn't randomized and reconnection attempts continue indefinitely.

```
<networkConnectors>
                                    <networkConnector name="connector_1_to_2"
 userName="myCommonUser" duplex="true"
                                        uri="masterslave:(ssl://
b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-east-2.amazonaws.com:61617,
                                        ssl://
b-9876l5k4-32ji-109h-8gfe-7d65c4b132a1-2.mq.us-west-2.amazonaws.com:61617)"/>
                                    </networkConnectors>
```

> **Note**
>
> We recommend using the `masterslave:` prefix for networks of brokers. The prefix is identical to the more explicit `static:failover:()?randomize=false&maxReconnectAttempts=0` syntax.

> **Note**
>
> This XML configuration does not allow spaces.

kahaDB

kahaDB is a child of the `persistenceAdapter` child collection element.

Attributes

concurrentStoreAndDispatchQueues

Specifies whether to use concurrent store and dispatch for queues. For more information, see Disable Concurrent Store and Dispatch for Queues with Slow Consumers.

Default: `true`

cleanupOnStop

> **Supported in**
>
> Apache ActiveMQ 15.16.x and above

If deactivated, garbage collection and cleanup does not take place when the broker is stopped, which speeds up the shutdown process. The increased speed is useful in cases with large databases or scheduler databases.

Default: `true`

journalDiskSyncInterval

Interval (ms) for when to perform a disk sync if `journalDiskSyncStrategy=periodic`. For more information, see the Apache ActiveMQ kahaDB documentation.

Default: `1000`

journalDiskSyncStrategy

> **Supported in**
>
> Apache ActiveMQ 15.14.x and above

Configures the disk sync policy. For more information, see the Apache ActiveMQ kahaDB documentation.

Default: `always`

> **Note**
>
> The ActiveMQ documentation states that the data loss is limited to the duration of `journalDiskSyncInterval`, which has a default of 1s. The data loss can be longer than the interval, but it is difficult to be precise. Use caution.

preallocationStrategy

Configures how the broker will try to preallocate the journal files when a new journal file is needed. For more information, see the Apache ActiveMQ kahaDB documentation.

Default: `sparse_file`

Example Configuration

Example

```
<broker xmlns="http://activemq.apache.org/schema/core">
                                <persistenceAdapter>
                                <kahaDB preallocationStrategy="zeros"
 concurrentStoreAndDispatchQueues="false" journalDiskSyncInterval="10000"
 journalDiskSyncStrategy="periodic"/>
                            </persistenceAdapter>
                            </broker>
```

systemUsage

`systemUsage` is a child of the `systemUsage` child collection element. It controls the maximum amount of space the broker will use before slowing down producers. For more information, see Producer Flow Control in the Apache ActiveMQ documentation.

Child Element

memoryUsage

`memoryUsage` is a child of the `systemUsage` child element. It manages memory usage. Use `memoryUsage` to keep track of how much of something is being used so that you can control working set usage productively. For more information, see the schema in the Apache ActiveMQ documentation.

Child Element

`memoryUsage` is a child of the `memoryUsage` child element.

Attribute

percentOfJvmHeap

Integer between 0 (inclusive) and 70 (inclusive).

Default: `70`

Attributes

sendFailIfNoSpace

Sets whether a `send()` method should fail if there is no space free. The default value is false, which blocks the `send()` method until space becomes available. For more information, see the schema in the Apache Active MQ documentation.

Default: `false`

sendFailIfNoSpaceAfterTimeout

Default: `null`

Example Configuration

Example

```
<broker xmlns="http://activemq.apache.org/schema/core">
                                <systemUsage>
                                <systemUsage sendFailIfNoSpace="true"
 sendFailIfNoSpaceAfterTimeout="2000">
                                    <memoryUsage>
                                        <memoryUsage  percentOfJvmHeap="60" />
                                    </memoryUsage>>
                                </systemUsage>
                                </systemUsage>
                            </broker>
                            </persistenceAdapter>
```

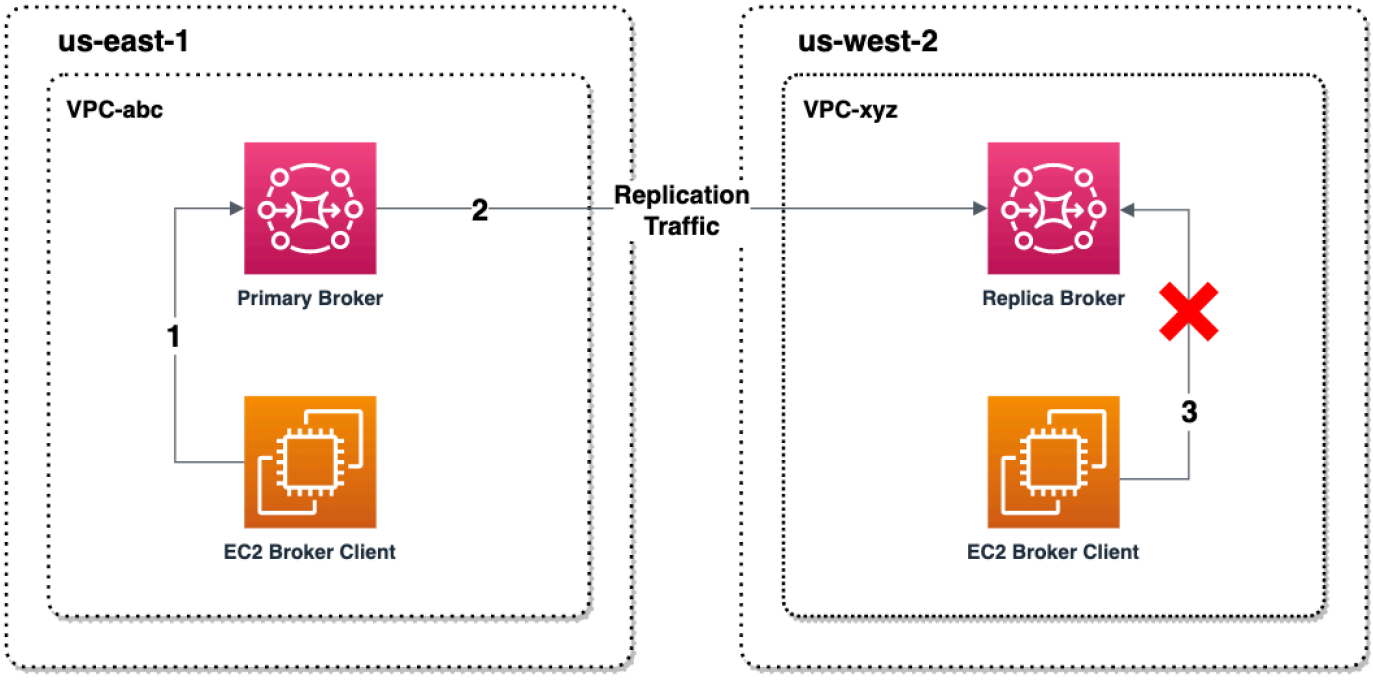
Cross-Region data replication for Amazon MQ for ActiveMQ

Amazon MQ for ActiveMQ offers a Cross-Region data replication (CRDR) feature that allows for asynchronous message replication from the primary broker in a primary AWS Region to the replica broker in a replica Region. By issuing a failover request to the Amazon MQ API, the current replica broker is promoted to the primary broker role, and the current primary broker is demoted to the replica role.

Primary and replica brokers for cross-Region data replication

You can create primary and replica brokers for asynchronous data replication from the primary broker in a primary AWS Region to the replica broker in a replica Region. The *primary Region* consists of a redundant pair of active/standby brokers referred to as the *primary broker*. The *secondary Region* consists of a redundant pair of active/standby brokers referred to as the *replica broker*.

The following diagram illustrates a replica broker in a secondary Region receiving asynchronous replicated data from the primary broker in the primary Region.



Primary and replica brokers act as a cross-Region data recovery solution. If the primary broker in the primary Region fails, you can promote the replica broker in the secondary Region to primary by initiating a switchover or failover. The former primary broker then becomes the replica broker, and

the former replica broker is promoted to primary broker. For instructions on creating a primary and replica broker, see Creating an Amazon MQ cross-Region data replication broker.

> **Note**
>
> Only available for active/standby brokers.
> Not available for mirrored queues.

Creating an Amazon MQ cross-Region data replication broker

With Cross-Region data replication (CRDR), you can switch between Amazon MQ for ActiveMQ message brokers in two AWS Regions as needed. You can designate an existing broker as a primary broker and create a replica for this broker, or create a new primary and replica broker together. You can then promote the replica broker to the primary broker role using the Amazon MQ `Promote` API operation. For more information about primary and replica brokers, see Primary and replica brokers for cross-Region data replication.

The following instructions describe how you can create and configure a replica broker using the Amazon MQ Management Console.

Topics

Prerequisites

To use the cross-Region data replication feature, you must review and comply with the following prerequisites:

- **Version**: The cross-Region data replication feature is only available for Amazon MQ for ActiveMQ brokers on versions 5.17.6 and above.
- **Region**: Cross-Region data replication is supported in the following regions: US East (Ohio), US East (N. Virginia), US West (Oregon), and US West (N. California).
- **Instance type**: Cross-Region data replication is only available for broker instance sizes `mq.m5.large` and above.

- **Deployment type**: Cross-Region data replication is only available for active/standby brokers with multi-availability zone deployment.
- **Broker status**: You can only create a replica broker for a primary broker with the broker status `Running`.

Step 1 (Optional): Create a new primary broker

Create a new primary broker

1. Sign in to the Amazon MQ console.
2. On the Brokers page of the Amazon MQ console, choose **Create brokers**.
3. On the **Select broker engine** page, choose **Apache ActiveMQ**.
4. On the **Select deployment and storage** page, in the **Deployment mode and storage type** section, do the following:
   - For the **Deployment mode**, choose **Active/standby broker**. An **Active/standby broker** is comprised of two brokers in two different Availability Zones configured in a redundant pair. These brokers communicate synchronously with your application and with Amazon EFS. For more information, see Deployment options for Amazon MQ for ActiveMQ brokers.
5. Choose **Next**.
6. On the **Configure settings** page, in the **Details** section, do the following:
   a. Enter the **Broker name**.

      > **Important**
      >
      > Do not add personally identifiable information (PII) or other confidential or sensitive information in broker names. Broker names are accessible to other AWS services, including CloudWatch Logs. Broker names are not intended to be used for private or sensitive data.

   b. Choose the **Broker instance type** (for example, **mq.m5.large**). For more information, see Broker instance types.
7. In the **ActiveMQ Web Console access** section, provide a **Username** and **Password**. The following restrictions apply to broker usernames and passwords:

- Your username can contain only alphanumeric characters, dashes, periods, underscores, and tildes (- . _ ~).
- Your password must be at least 12 characters long, contain at least 4 unique characters and must not contain commas, colons, or equal signs (,:=).

> **Important**
>
> Do not add personally identifiable information (PII) or other confidential or sensitive information in broker usernames. Broker usernames are accessible to other AWS services, including CloudWatch Logs. Broker usernames are not intended to be used for private or sensitive data.

The green flash bar at the top of the page confirms that Amazon MQ is creating the replica broker in the recovery Region. You can also see the CRDR role and RPO status for your brokers. To turn off the CRDR Role and RPO Status columns, choose the gear icon in the top right corner of the **Brokers** table. Then, on the **Preferences** page, turn off CRDR Role or RPO Status.

Step 2: Create a replica of an existing broker

1. On the Brokers page of the Amazon MQ console, choose **Create replica broker**.
2. On the **Choose primary broker page**, select an existing broker to use as a CRDR primary broker. Then, choose **Next**.
3. On the **Configure replica broker** page, use the drop down menu to choose the replica Region.
4. In the **ActiveMQ console user for replica broker** section, provide a **Username** and **Password** for the replica broker console user. The following restrictions apply to broker usernames and passwords:

   - Your username can contain only alphanumeric characters, dashes, periods, underscores, and tildes (- . _ ~).
   - Your password must be at least 12 characters long, contain at least 4 unique characters and must not contain commas, colons, or equal signs (,:=).

> **Important**
>
> Do not add personally identifiable information (PII) or other confidential or sensitive information in broker usernames. Broker usernames are accessible to other AWS services, including CloudWatch Logs. Broker usernames are not intended to be used for private or sensitive data.

5. In the **Data replication user to bridge access between brokers** section, provide a **Username** and **Password** for the user that will access both the primary and replica broker. The following restrictions apply to broker usernames and passwords:

   - Your username can contain only alphanumeric characters, dashes, periods, underscores, and tildes (- . _ ~).
   - Your password must be at least 12 characters long, contain at least 4 unique characters and must not contain commas, colons, or equal signs (,:=).

   > **Important**
   >
   > Do not add personally identifiable information (PII) or other confidential or sensitive information in broker usernames. Broker usernames are accessible to other AWS services, including CloudWatch Logs. Broker usernames are not intended to be used for private or sensitive data.

   Configure any additional settings. Then, choose **Next**.

6. On the **Review and create** page, review the replica broker details. Then, choose **Create replica broker**.
7. Next, reboot the primary broker. This will also reboot the replica broker. For instructions on rebooting your broker, see Rebooting a Broker.

For more information on configuring additional settings for your ActiveMQ broker, see Getting started: Creating and connecting to an ActiveMQ broker

Deleting an Amazon MQ cross-Region data replication broker

To delete a primary or replica cross-Region data replication (CRDR) broker, you must first unpair then reboot the brokers. The following instructions show how you can unpair and reboot the brokers using the AWS Management Console.

1. On the **Brokers** page, select the CRDR broker you want to unpair, then choose **Edit**.
2. On the broker **Edit** page in the **Data replication** section, choose **Unpair brokers**.
3. Enter "confirm" in the pop-up window to confirm your choice. Then choose **Unpair brokers**.
4. Next, reboot the unpaired primary broker. This will also reboot the replica broker. For instructions on rebooting your broker, see Rebooting a Broker. After the primary broker is rebooted, both brokers are unpaired and can be individually deleted. To delete your broker, see Deleting a broker.

Initiating switchover or failover to promote an Amazon MQ replica broker to primary broker role

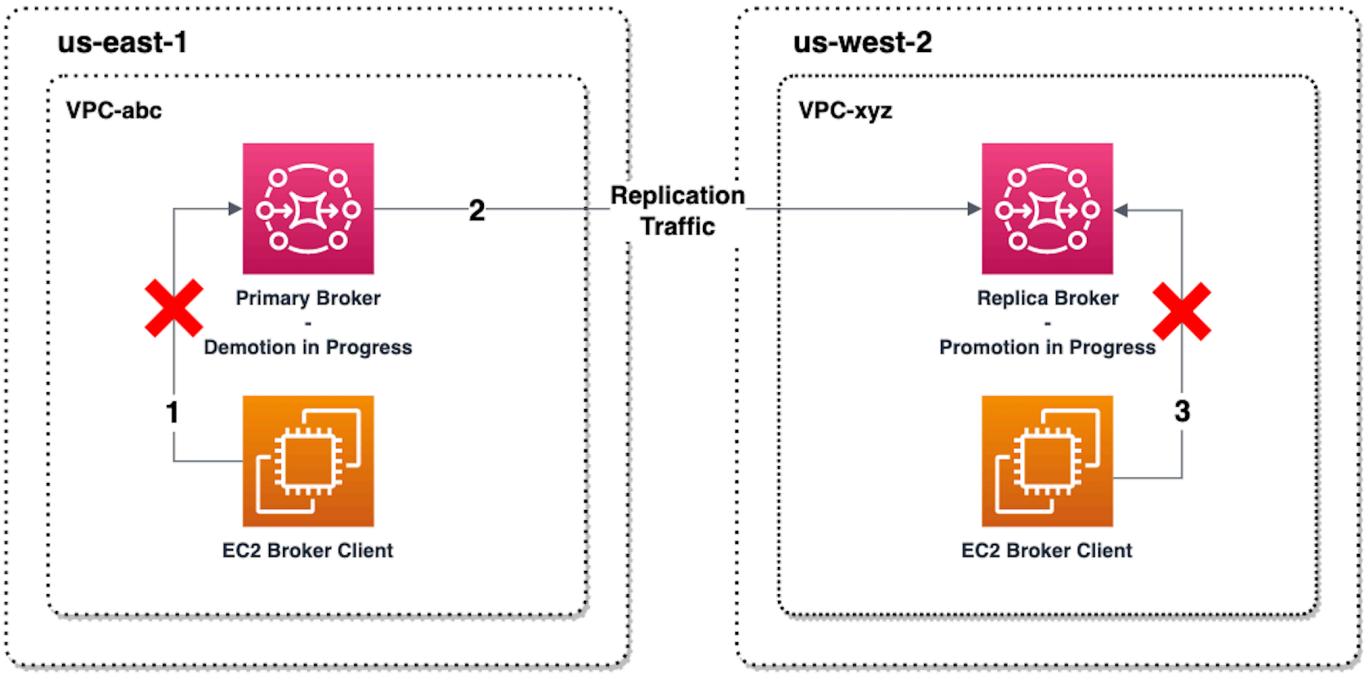
You can initiate a switchover or failover when you want to promote the replica broker to the primary broker role. When you promote the replica broker, the primary broker is demoted to the replica broker role.

A **switchover** prioritizes consistency over availability. Brokers are guaranteed to have identical state when this failover operation completes. With a switchover, there may be a period where neither broker is available for client connections while inter-broker consistency is established. Both brokers will have the same state at the instant when the replica is promoted. Switchover success depends on the health of both regions and the inter-region network to succeed.

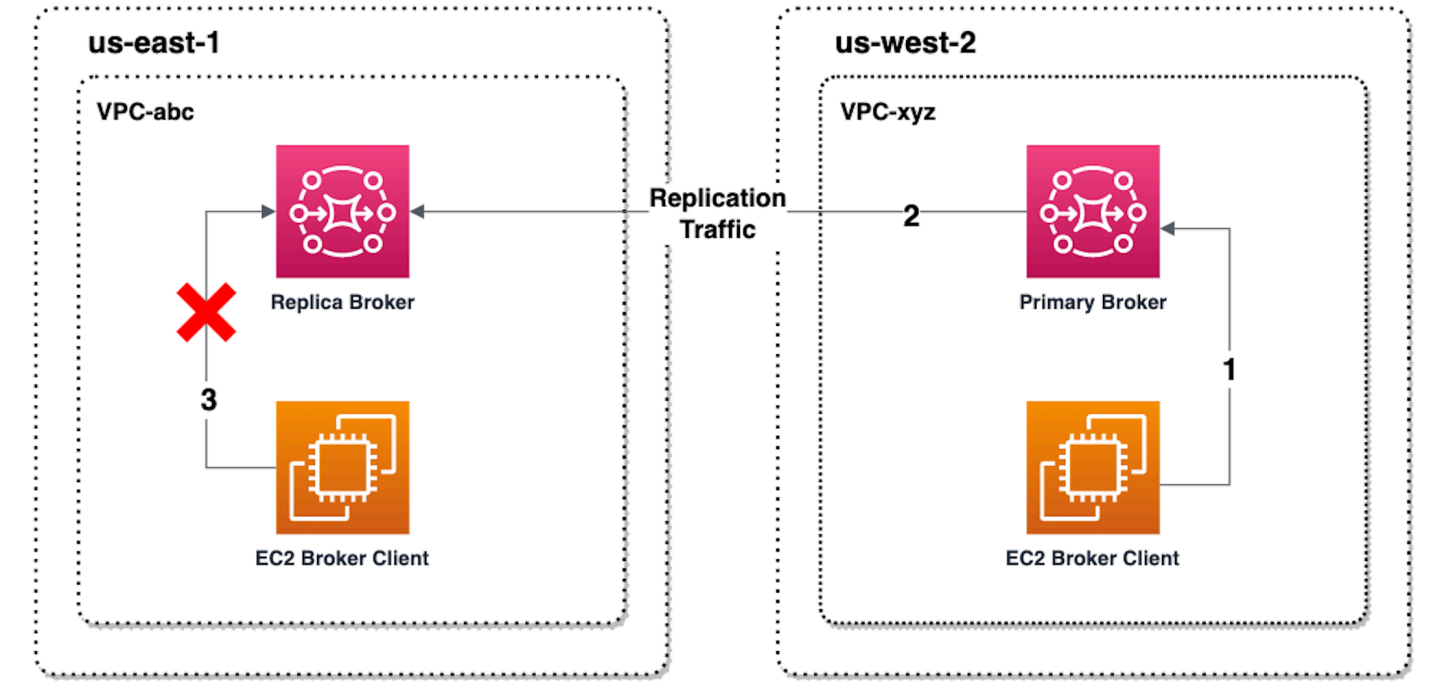
A **failover** prioritizes availability over consistency. Brokers are not guaranteed to have identical states when this operation completes. With a failover, the replica broker is guaranteed to become immediately available to serve client traffic, without waiting for any replication data to be synchronized, or for the primary to receive the shutdown signal. Failover depends on neither the health of the original primary region nor the inter-region network to succeed.

The following diagram illustrates a switchover in which neither broker accepts client connections while the replication queue is being drained and broker states are synchronized. In this process, the client in the primary broker's VPC is unable to produce further state changes while the operation is in progress, and the primary broker is being demoted to a replica. When the replication queue is

drained and the two brokers achieve identical state, the client in the replica broker's VPC is unable to connect to the replica broker until the failover operation completes, and the replica broker is promoted to primary.



The following diagram illustrates the broker status after the switchover process is complete. The original replica broker has now been promoted to the primary broker role and is accepting client connections. The client can produce and consume data from the broker.



Promote the replica broker using the console

To promote the replica broker using switchover or failover, follow these steps in the Amazon MQ console.

> **Note**
>
> You cannot initiate switchover or failover on a primary broker.

1. Switch to the region for your replica broker. From your Brokers table, select the existing replica broker you will promote to primary.
2. On the **Broker details page**, do the following:
    1. Select **Promote replica**.
    2. In the pop up window, chose *Switchover* or *Failover*.
    3. Type “confirm” in the text box to confirm your choice.
    4. Choose **Confirm**.

After initiating failover, the broker status changes to *Failover in progress*. The blue progress bar at the top of the Brokers page becomes green when failover is complete.

> **Note**
>
> The configuration is only replicated at the time the replicat broker is created. Any update afterwards is not replicated.

Cross-Region data replication metrics in Amazon CloudWatch

The Amazon MQ for ActiveMQ cross-Region data replication feature offers metrics for maintaining the reliability, availability, and performance of your primary and replica brokers. During the replication process, a replica broker in a secondary Region receives asynchronously replicated data from the primary broker in the primary Region. If the primary broker in the primary Region fails, you can promote the replica broker in the secondary Region to primary by initiating a switchover or failover. For instructions on viewing metrics in Amazon CloudWatch, see Accessing CloudWatch metrics for Amazon MQ.

CRDR timestamps

The following timestamps describe how the metrics found in Amazon CloudWatch are calculated. There are five timestamps in the data replication process:

- Time of current observation (TCO): The current instant in time.
- Time of creation (TC): The instant in time an event was created on the replication queue by the primary broker. Available on both primary and replica brokers.
- Time of delivery (TD): The instant in time an event was successfully delivered to the replica broker. Only available on replica brokers.
- Time of processing (TP): The instant in time an event was successfully processed by the replica broker. Only available on replica brokers.
- Time of acknowledgement (TA): The instant in time an event was successfully acknowledged by the primary broker. Only available on primary brokers.

Estimate switchover/failover performance with CRDR CloudWatch metrics

Amazon MQ enables metrics for your broker by default. You can view your broker metrics by accessing the Amazon CloudWatch console, or by using the CloudWatch API. The following metrics are useful for understanding the replication and switchover/failover performance of your CRDR brokers:

Amazon MQ CloudWatch metric	Reason for CRDR use
`TotalReplicationLag`	The estimated time between TA and TC of the last unacknowledged event on the primary broker.
`ReplicationLag`	The estimated time between TP and TC of the last unacknowledged event on the replica broker.
`PrimaryWaitTime`	The estimated time between TCO and TC of the last processed event on the primary broker.
`ReplicaWaitTime`	The estimated time between TCO and TP of the last processed event on the replica broker.
`QueueSize`	The total number of unacknowledged events in the replication queue on the primary broker.

`TotalReplicationLag` and `ReplicationLag` describe the delayed replication between the primary and replica brokers. The two metrics can also be used to estimate the time until the ongoing switchover or failover operation complete.

`PrimaryWaitTime` and `ReplicaWaitTime` can be used to identify any ongoing issues with the replication process. If the value of the metric is constantly growing, this can indicate the replication process is degraded or paused. Slow replication may happen due issues like to network partitioning, broker starts, and long recovery.

ActiveMQ tutorials

The following tutorials show how you can create and connect to your ActiveMQ brokers. To use the ActiveMQ Java example code, you must install the Java Standard Edition Development Kit and make some changes to the code

Topics

- Creating and configuring an Amazon MQ network of brokers
- Connecting a Java application to your Amazon MQ broker
- Integrating ActiveMQ brokers with LDAP
- Step 3: (Optional) Connect to an AWS Lambda function
- Creating an ActiveMQ broker user
- Edit an ActiveMQ broker user
- Delete an ActiveMQ broker user
- Working examples of using Java Message Service (JMS) with ActiveMQ

Creating and configuring an Amazon MQ network of brokers

A *network of brokers* is comprised of multiple simultaneously active single-instance brokers or active/standby brokers. In this tutorial, you learn how to create a two-broker network of brokers with a *source and sink* topology.

For a conceptual overview and detailed configuration information, see the following:

- Amazon MQ network of brokers
- Configure Your Network of Brokers Correctly

- `networkConnector`
- `networkConnectionStartAsync`
- Networks of Brokers in the ActiveMQ documentation

You can use the Amazon MQ console to create an Amazon MQ network of brokers. Because you can start the creation of the two brokers in parallel, this process takes approximately 15 minutes.

Topics

Prerequisites

To create a network of brokers, you must have the following:

- Two or more simultaneously active brokers (named `MyBroker1` and `MyBroker2` in this tutorial). For more information about creating brokers, see Getting started: Creating and connecting to an ActiveMQ broker.
- The two brokers must be in the same VPC or in peered VPCs. For more information about VPCs, see What is Amazon VPC? in the *Amazon VPC User Guide* and What is VPC Peering? in the *Amazon VPC Peering Guide*.

  > **Important**
  >
  > If you don't have a default VPC, subnet(s), or security group, you must create them first. For more information, see the following in the *Amazon VPC User Guide*:
  >
  > - Creating a Default VPC
  > - Creating a Default Subnet
  > - Creating a Security Group

- Two users with identical sign-in credentials for both brokers. For more information about creating users, see Creating an ActiveMQ broker user.

> **Note**
>
> When integrating LDAP authentication with a network of brokers, make sure that the user exists both as an ActiveMQ brokers, as well as an LDAP user.

The following example uses two single-instance brokers. However, you can create networks of brokers using active/standby brokers or a combination of broker deployment modes.

Step 1: Allow Traffic between Brokers

After you create your brokers, you must allow traffic between them.

1. On the Amazon MQ console, on the **MyBroker2** page, in the **Details** section, under **Security and network**, choose the name of your security group or

   The **Security Groups** page of the EC2 Dashboard is displayed.
2. From the security group list, choose your security group.
3. At the bottom of the page, choose **Inbound**, and then choose **Edit**.
4. In the **Edit inbound rules** dialog box, add a rule for the OpenWire endpoint.
   a. Choose **Add Rule**.
   b. For **Type**, select **Custom TCP**.
   c. For **Port Range**, type the OpenWire port (61617).
   d. Do one of the following:
      - If you want to restrict access to a particular IP address, for **Source**, leave **Custom** selected, and then enter the IP address of `MyBroker1`, followed by `/32`. (This converts the IP address to a valid CIDR record). For more information see Elastic Network Interfaces.

        > **Tip**
        >
        > To retrieve the IP address of `MyBroker1`, on the Amazon MQ console, choose the name of the broker and navigate to the **Details** section.

- If all the brokers are private and belong to the same VPC, for **Source**, leave **Custom** selected and then type the ID of the security group you are editing.

> **Note**
>
> For public brokers, you must restrict access using IP addresses.

e. Choose **Save**.

Your broker can now accept inbound connections.

Step 2: Configure Network Connectors for Your Broker

After you allow traffic between your brokers, you must configure network connectors for one of them.

1. Edit the configuration revision for broker `MyBroker1`.

   a. On the **MyBroker1** page, choose **Edit**.

   b. On the **Edit MyBroker1** page, in the **Configuration** section, choose **View**.

   The broker engine type and version that the configuration uses (for example, **Apache ActiveMQ 5.15.0**) are displayed.

   c. On the **Configuration details** tab, the configuration revision number, description, and broker configuration in XML format are displayed.

   d. Choose **Edit configuration**.

   e. At the bottom of the configuration file, uncomment the `<networkConnectors>` section and include the following information:

   - The `name` for the network connector.
   - The ActiveMQ Web Console `username` that is common to both brokers.
   - Enable `duplex` connections.
   - Do one of the following:
     - If you are connecting the broker to a single-instance broker, use the `static:` prefix and the OpenWire endpoint `uri` for `MyBroker2`. For example:

```
<networkConnectors>
```

```
  <networkConnector name="connector_1_to_2" userName="myCommonUser"
 duplex="true"
    uri="static:(ssl://b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-
east-2.amazonaws.com:61617)"/>
</networkConnectors>
```

- If you are connecting the broker to an active/standby broker, use the `static +failover` transport and the OpenWire endpoint `uri` for both brokers with the following query parameters `?randomize=false&maxReconnectAttempts=0`. For example:

```
<networkConnectors>
  <networkConnector name="connector_1_to_2" userName="myCommonUser"
 duplex="true"
    uri="static:(failover:(ssl://
b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-east-2.amazonaws.com:61617,
    ssl://b-9876l5k4-32ji-109h-8gfe-7d65c4b132a1-2.mq.us-
west-2.amazonaws.com:61617)?randomize=false&amp;maxReconnectAttempts=0)"/>
</networkConnectors>
```

> **Note**
>
> Don't include the sign-in credentials for the ActiveMQ user.

f. Choose **Save**.

g. In the **Save revision** dialog box, type `Add network of brokers connector for MyBroker2`.

h. Choose **Save** to save the new revision of the configuration.

2. Edit `MyBroker1` to set the latest configuration revision to apply immediately.

a. On the **MyBroker1** page, choose **Edit**.

b. On the **Edit MyBroker1** page, in the **Configuration** section, choose **Schedule Modifications**.

c. In the **Schedule broker modifications** section, choose to apply modifications **Immediately**.

d. Choose **Apply**.

`MyBroker1` is rebooted and your configuration revision is applied.

The network of brokers is created.

Next Steps

After you configure your network of brokers, you can test it by producing and consuming messages.

> **Important**
>
> Make sure that you enable inbound connections *from your local machine* for broker `MyBroker1` on port 8162 (for the ActiveMQ Web Console) and port 61617 (for the OpenWire endpoint).
> You might also need to adjust your security group(s) settings to allow the producer and consumer to connect to the network of brokers.

1. On the Amazon MQ console, navigate to the **Connections** section and note the ActiveMQ Web Console endpoint for broker `MyBroker1`.
2. Navigate to the ActiveMQ Web Console for broker `MyBroker1`.
3. To verify that the network bridge is connected, choose **Network**.

   In the **Network Bridges** section, the name and the address of `MyBroker2` are listed in the **Remote Broker** and **Remote Address** columns.
4. From any machine that has access to broker `MyBroker2`, create a consumer. For example:

   ```
   activemq consumer --brokerUrl "ssl://
   b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-east-2.amazonaws.com:61617" \
    --user commonUser \
    --password myPassword456 \
    --destination queue://MyQueue
   ```

   The consumer connects to the OpenWire endpoint of `MyBroker2` and begins to consume messages from queue `MyQueue`.
5. From any machine that has access to broker `MyBroker1`, create a producer and send some messages. For example:

```
activemq producer --brokerUrl "ssl://
b-9876l5k4-32ji-109h-8gfe-7d65c4b132a1-1.mq.us-east-2.amazonaws.com:61617" \
 --user commonUser \
 --password myPassword456 \
 --destination queue://MyQueue \
 --persistent true \
 --messageSize 1000 \
 --messageCount 10000
```

The producer connects to the OpenWire endpoint of `MyBroker1` and begins to produce persistent messages to queue `MyQueue`.

Connecting a Java application to your Amazon MQ broker

After you create an Amazon MQ ActiveMQ broker, you can connect your application to it. The following examples show how you can use the Java Message Service (JMS) to create a connection to the broker, create a queue, and send a message. For a complete, working Java example, see Working Java Example.

You can connect to ActiveMQ brokers using various ActiveMQ clients. We recommend using the ActiveMQ Client.

Topics

Prerequisites

Enable VPC Attributes

To ensure that your broker is accessible within your VPC, you must enable the `enableDnsHostnames` and `enableDnsSupport` VPC attributes. For more information, see DNS Support in your VPC in the *Amazon VPC User Guide*.

Enable Inbound Connections

Next, enable inbound connections for your application.

1. Sign in to the Amazon MQ console.
2. From the broker list, choose the name of your broker (for example, **MyBroker**).
3. On the ***MyBroker*** page, in the **Connections** section, note the addresses and ports of the broker's web console URL and wire-level protocols.
4. In the **Details** section, under **Security and network**, choose the name of your security group or .

   The **Security Groups** page of the EC2 Dashboard is displayed.
5. From the security group list, choose your security group.
6. At the bottom of the page, choose **Inbound**, and then choose **Edit**.
7. In the **Edit inbound rules** dialog box, add a rule for every URL or endpoint that you want to be publicly accessible (the following example shows how to do this for a broker web console).

   a. Choose **Add Rule**.

   b. For **Type**, select **Custom TCP**.

   c. For **Port Range**, type the web console port (8162).

   d. For **Source**, leave **Custom** selected and then type the IP address of the system that you want to be able to access the web console (for example, `192.0.2.1`).

   e. Choose **Save**.

   Your broker can now accept inbound connections.

Add Java Dependencies

Add the `activemq-client.jar` and `activemq-pool.jar` packages to your Java class path. The following example shows these dependencies in a Maven project `pom.xml` file.

```
<dependencies>
    <dependency>
        <groupId>org.apache.activemq</groupId>
        <artifactId>activemq-client</artifactId>
        <version>5.15.16</version>
    </dependency>
    <dependency>
        <groupId>org.apache.activemq</groupId>
        <artifactId>activemq-pool</artifactId>
        <version>5.15.16</version>
```

```
    </dependency>
</dependencies>
```

For more information about `activemq-client.jar`, see Initial Configuration in the Apache ActiveMQ documentation.

> **Important**
>
> In the following example code, producers and consumers run in a single thread. For production systems (or to test broker instance failover), make sure that your producers and consumers run on separate hosts or threads.

To Create a Message Producer and Send a Message

Use the following instruction to create a message producer and recieve a message.

1. Create a JMS pooled connection factory for the message producer using your broker's endpoint and then call the `createConnection` method against the factory.

   > **Note**
   >
   > For an active/standby broker, Amazon MQ provides two ActiveMQ Web Console URLs, but only one URL is active at a time. Likewise, Amazon MQ provides two endpoints for each wire-level protocol, but only one endpoint is active in each pair at a time. The `-1` and `-2` suffixes denote a redundant pair. For more information, see Deployment options for Amazon MQ for ActiveMQ brokers).
   > For wire-level protocol endpoints, you should allow your application to connect to either endpoint by using the Failover Transport.

   ```
   // Create a connection factory.
   final ActiveMQConnectionFactory connectionFactory = new
    ActiveMQConnectionFactory(wireLevelEndpoint);

   // Pass the sign-in credentials.
   connectionFactory.setUserName(activeMqUsername);
   connectionFactory.setPassword(activeMqPassword);
   ```

```
// Create a pooled connection factory.
final PooledConnectionFactory pooledConnectionFactory = new
 PooledConnectionFactory();
pooledConnectionFactory.setConnectionFactory(connectionFactory);
pooledConnectionFactory.setMaxConnections(10);

// Establish a connection for the producer.
final Connection producerConnection = pooledConnectionFactory.createConnection();
producerConnection.start();

// Close all connections in the pool.
pooledConnectionFactory.clear();
```

> **Note**
>
> Message producers should always use the `PooledConnectionFactory` class. For more information, see Always Use Connection Pooling.

2. Create a session, a queue named `MyQueue`, and a message producer.

```
// Create a session.
final Session producerSession = producerConnection.createSession(false,
 Session.AUTO_ACKNOWLEDGE);

// Create a queue named "MyQueue".
final Destination producerDestination = producerSession.createQueue("MyQueue");

// Create a producer from the session to the queue.
final MessageProducer producer =
 producerSession.createProducer(producerDestination);
producer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);
```

3. Create the message string `"Hello from Amazon MQ!"` and then send the message.

```
// Create a message.
final String text = "Hello from Amazon MQ!";
TextMessage producerMessage = producerSession.createTextMessage(text);

// Send the message.
producer.send(producerMessage);
System.out.println("Message sent.");
```

4. Clean up the producer.

```
producer.close();
producerSession.close();
producerConnection.close();
```

To Create a Message Consumer and Receive the Message

Use the following instruction to create a message producer and recieve a message.

1. Create a JMS connection factory for the message producer using your broker's endpoint and then call the `createConnection` method against the factory.

```
// Create a connection factory.
final ActiveMQConnectionFactory connectionFactory = new
 ActiveMQConnectionFactory(wireLevelEndpoint);

// Pass the sign-in credentials.
connectionFactory.setUserName(activeMqUsername);
connectionFactory.setPassword(activeMqPassword);

// Establish a connection for the consumer.
final Connection consumerConnection = connectionFactory.createConnection();
consumerConnection.start();
```

> **Note**
>
> Message consumers should *never* use the `PooledConnectionFactory` class. For more information, see Always Use Connection Pooling.

2. Create a session, a queue named `MyQueue`, and a message consumer.

```
// Create a session.
final Session consumerSession = consumerConnection.createSession(false,
 Session.AUTO_ACKNOWLEDGE);

// Create a queue named "MyQueue".
final Destination consumerDestination = consumerSession.createQueue("MyQueue");

// Create a message consumer from the session to the queue.
```

```
final MessageConsumer consumer =
 consumerSession.createConsumer(consumerDestination);
```

3. Begin to wait for messages and receive the message when it arrives.

```
// Begin to wait for messages.
final Message consumerMessage = consumer.receive(1000);

// Receive the message when it arrives.
final TextMessage consumerTextMessage = (TextMessage) consumerMessage;
System.out.println("Message received: " + consumerTextMessage.getText());
```

> **Note**
>
> Unlike AWS messaging services (such as Amazon SQS), the consumer is constantly connected to the broker.

4. Close the consumer, session, and connection.

```
consumer.close();
consumerSession.close();
consumerConnection.close();
```

Integrating ActiveMQ brokers with LDAP

> **Important**
>
> Amazon MQ does not support server certificate issued by a private CA.

You can access your ActiveMQ brokers using the following protocols with TLS enabled:

- AMQP
- MQTT
- MQTT over WebSocket
- OpenWire
- STOMP

- STOMP over WebSocket

Amazon MQ offers a choice between native ActiveMQ authentication and LDAP authentication and authorization to manage user permissions. For information about restrictions related to ActiveMQ usernames and passwords, see Users.

To authorize ActiveMQ users and groups to works with queues and topics, you must edit your broker's configuration. Amazon MQ uses ActiveMQ's Simple Authentication Plugin to restrict reading and writing to destinations. For more information and examples, see Always configure an authorization map and `authorizationEntry`.

> **Note**
>
> Currently, Amazon MQ doesn't support Client Certificate Authentication.

Topics

Integrate LDAP with ActiveMQ

You can authenticate Amazon MQ users through the credentials stored in your lightweight directory access protocol (LDAP) server. You can also add, delete, and modify Amazon MQ users and assign permissions to topics and queues through it. Management operations like creating, updating and deleting brokers still require IAM credentials and are not integrated with LDAP.

Customers who want to simplify and centralize their Amazon MQ broker authentication and authorization using an LDAP server can use this feature. Keeping all user credentials in the LDAP server saves time and effort by providing a central location for storing and managing these credentials.

Amazon MQ provides LDAP support using the Apache ActiveMQ JAAS plugin. Any LDAP server, such as Microsoft Active Directory or OpenLDAP that is supported by the plugin is also supported

by Amazon MQ. For more information about the plugin, see the Security section of the Active MQ documentation.

In addition to users, you can specify access to topics and queues for a specific group or a user through your LDAP server. You do this by creating entries that represent topics and queues in your LDAP server and then assigning permissions to a specific LDAP user or a group. You can then configure broker to retrieve authorization data from the LDAP server.

> **Important**
>
> When using LDAP, authentication is not case-sensitive, but authorization is case-sensitive for your username.

Prerequisites

Before you add LDAP support to a new or existing Amazon MQ broker, you must set up a service account. This service account is required to initiate a connection to an LDAP server and must have the correct permissions to make this connection. This service account will set up LDAP authentication for your broker. Any successive client connections will be authenticated through the same connection.

The service account is an account in your LDAP server, which has access to initiate a connection. It is a standard LDAP requirement and you have to provide the service account credentials only once. After the connection is setup, all the future client connections are authenticated through your LDAP server. Your service account credentials are stored securely in an encrypted form, which is accessible only to Amazon MQ.

To integrate with ActiveMQ, a specific Directory Information Tree (DIT) is required on the LDAP server. For an example `ldif` file that clearly shows this structure, see *Import the following LDIF file into the LDAP server* in the Security section of the ActiveMQ documentation.

Getting Started with LDAP

To get started, navigate to the Amazon MQ console and choose **LDAP authentication and authorization** when you create a new Amazon MQ or edit an existing broker instance.

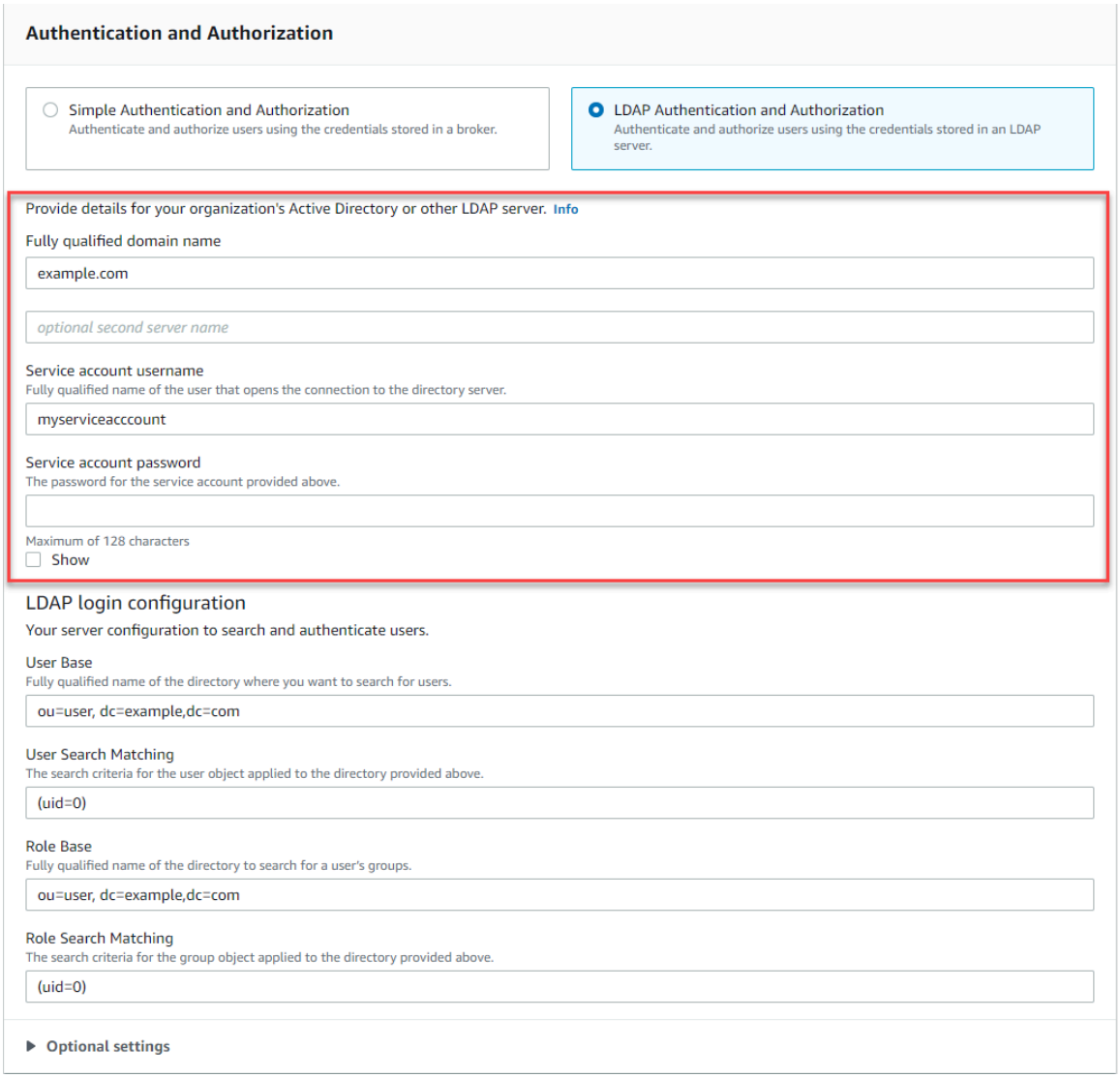
Provide the following information about the service account:

- **Fully qualified domain name** The location of the LDAP server to which authentication and authorization requests are to be issued.

  > **Note**
  >
  > The fully qualified domain name of the LDAP server you supply must not include the protocol or port number. Amazon MQ will prepend the fully qualified domain name with the protocol `ldaps`, and will append the port number `636`.
  > For example, if you provide the following fully qualified domain: `example.com`, Amazon MQ will access your LDAP server using the following URL: `ldaps://example.com:636`.
  > For the broker host to be able to successfully communicate with the LDAP server, the fully qualified domain name must be publicly resolvable. To keep the LDAP server private and secure, restrict inbound traffic in the server's inbound rules to only allow traffic originated from within the broker's VPC.

- **Service account username** The distinguished name of the user that will be used to perform the initial bind to the LDAP server.
- **Service account password** The password of the user performing the initial bind.

The following image highlights where to supply these details.



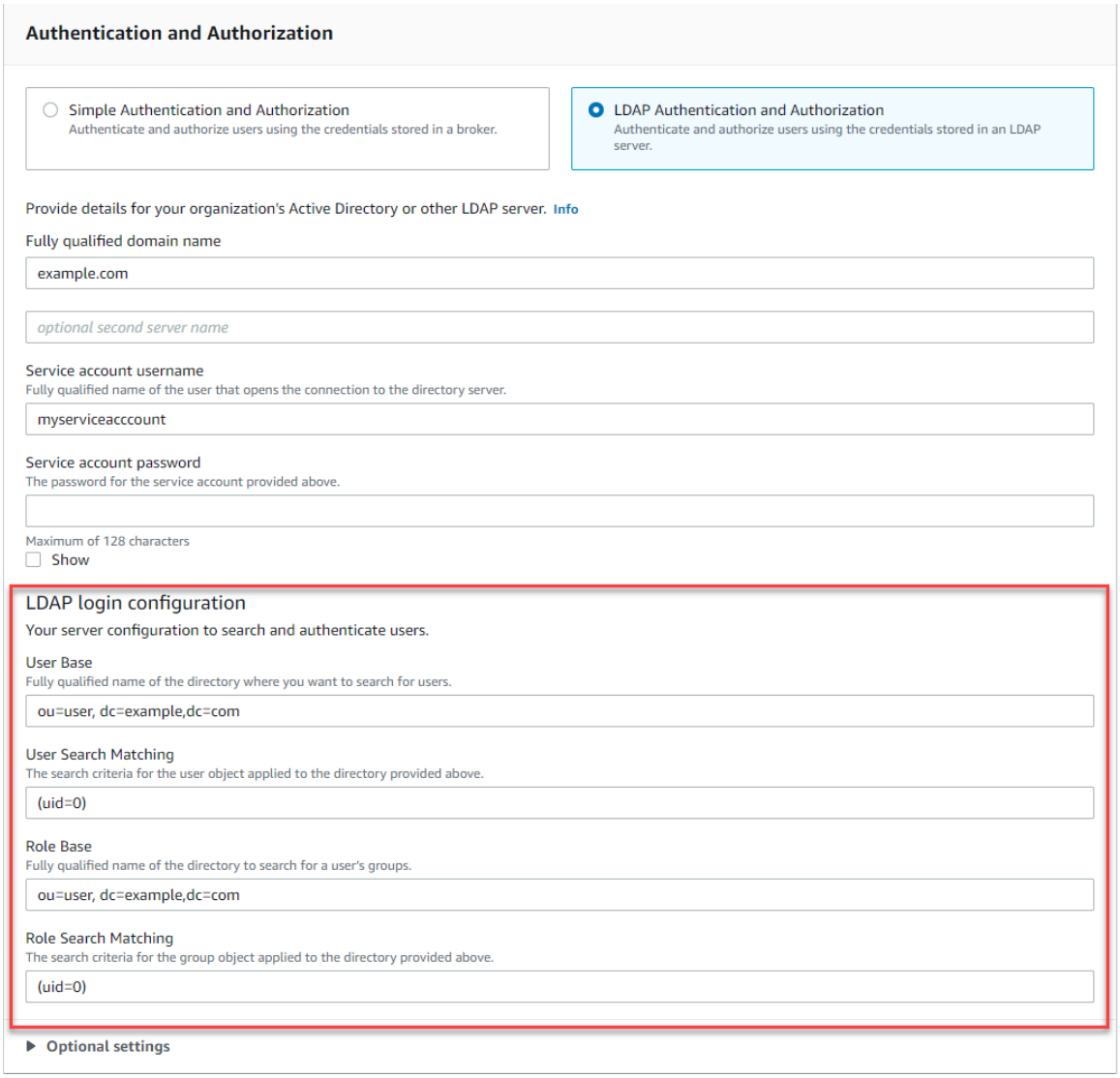
In the **LDAP login configuration** section, provide the following required information:

- **User Base** The distinguished name of the node in the directory information tree (DIT) that will be searched for users.
- **User Search Matching** The LDAP search filter that will be used to find users within the `userBase`. The client's username will be substituted into the `{0}` placeholder in the search filter. For more information, see Authentication and Authorization.
- **Role Base** The distinguished name of the node in the DIT that will be searched for roles. Roles can be configured as explicit LDAP group entries in your directory. A typical role entry may consist of one attribute for the name of the role, such as **common name (CN)**, and another attribute, such as `member`, with values representing the distinguished names or usernames of the

users belonging to the role group. For example, given the organizational unit, `group`, you can provide the following distinguished name: `ou=group,dc=example,dc=com`.

- **Role Search Matching** The LDAP search filter that will be used to find roles within the `roleBase`. The distinguished name of the user matched by `userSearchMatching` will be substituted into the `{0}` placeholder in the search filter. The client's username will be substituted in place of the `{1}` placeholder. For example, if role entries in your directory include an attribute named `member`, containing the usernames for all users in that role, you can provide the following search filter: `(member:=uid={1})`.

The following image highlights where to specify these details.



In the **Optional settings** section, you can provide the following optional information:

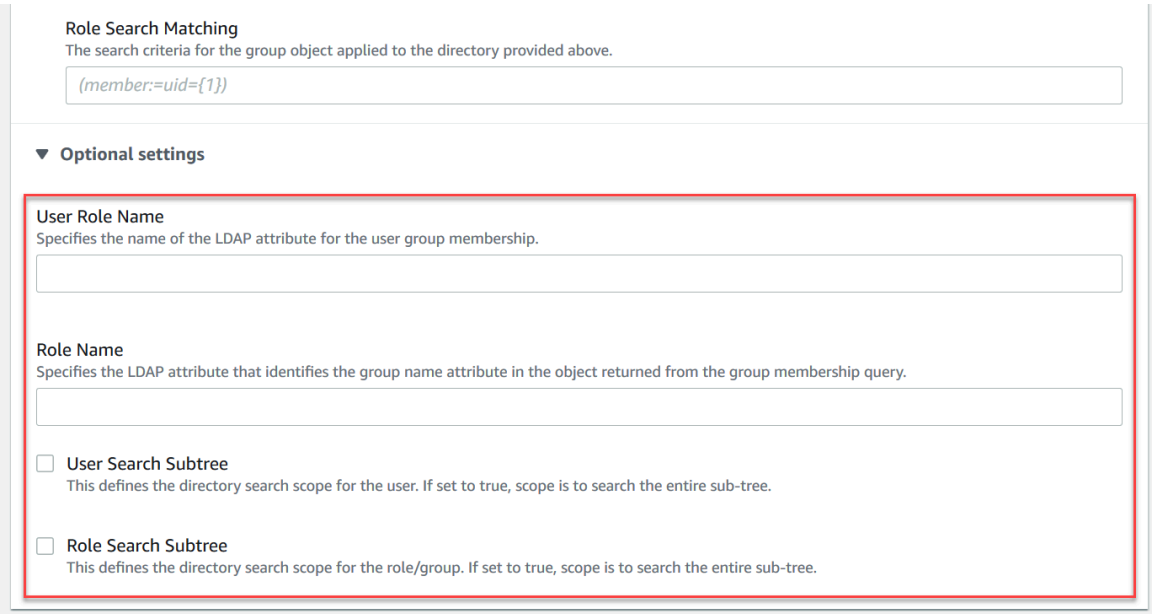
- **User Role Name** The name of the LDAP attribute in the user's directory entry for the user's group membership. In some cases, user roles may be identified by the value of an attribute in the user's directory entry. The `userRoleName` option allows you to provide the name of this attribute. For example, let's consider the following user entry:

  ```
  dn: uid=jdoe,ou=user,dc=example,dc=com
  objectClass: user
  uid: jdoe
  sn: jane
  cn: Jane Doe
  mail: j.doe@somecompany.com
  memberOf: role1
  userPassword: password
  ```

  To provide the correct `userRoleName` for the example above, you would specify the `memberOf` attribute. If authentication is successful, the user is assigned the role `role1`.

- **Role Name** The group name attribute in a role entry whose value is the name of that role. For example, you can specify `cn` for a group entry's **common name**. If authentication succeeds, the user is assigned the the value of the `cn` attribute for each role entry that they are a member of.
- **User Search Subtree** Defines the scope for the LDAP user search query. If true, the scope is set to search the entire subtree under the node defined by `userBase`.
- **Role Search Subtree** Defines the scope for the LDAP role search query. If true, the scope is set to search the entire subtree under the node defined by `roleBase`.

The following image highlights where to specify these optional settings.



How LDAP integration works

You can think of integration in two main categories: the structure for authentication, and the structure for authorization.

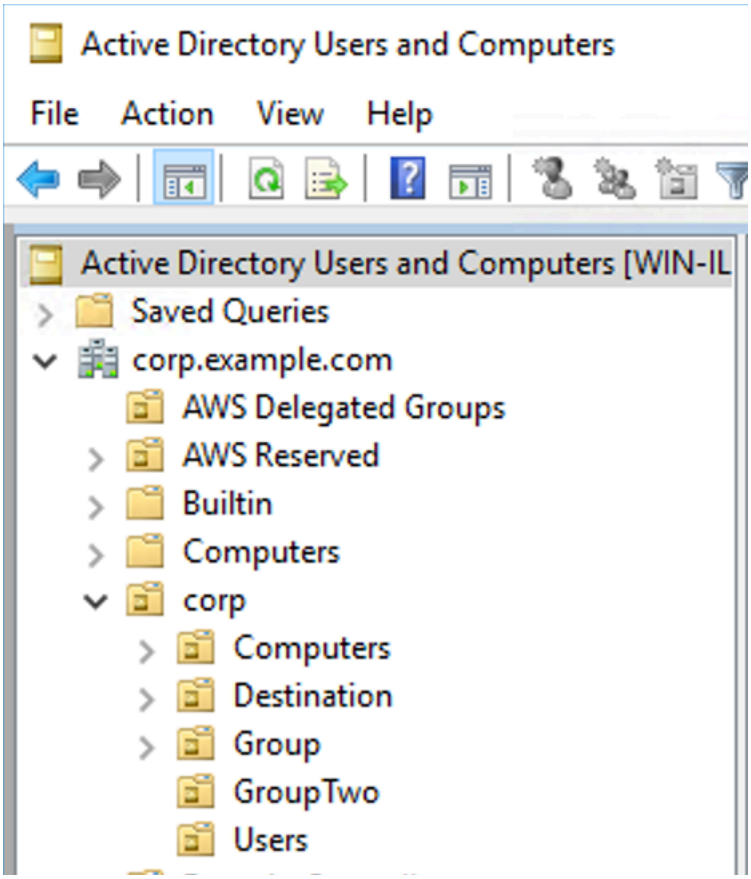
Authentication

For authentication, client credentials must be valid. These credentials are validated against users in the user base in the LDAP server.

The user base supplied to the ActiveMQ broker must point to the node in the DIT where users are stored in the LDAP server. For example, if you are using AWS Managed Microsoft AD, and you have the domain components `corp`, `example`, and `com`, and within those you have organizational units `corp` and `Users`, you would use the following as your user base:

```
OU=Users,OU=corp,DC=corp,DC=example,DC=com
```

The ActiveMQ broker would search at this location in the DIT for users in order to authenticate client connection requests to the broker.



Because the ActiveMQ source code hardcodes the attribute name for users to `uid`, you must make sure that each user has this attribute set. For simplicity, you can use the user's connection username. For more information, see the activemq source code and Configuring ID mappings in Active Directory Users and Computers for Windows Server 2016 (and subsequent) versions.

To enable ActiveMQ console access for specific users, make sure they belong to the `amazonmq-console-admins` group.

Authorization

For authorization, permissions search bases are specified in the broker configuration. Authorization is done on a per-destination basis (or wildcard, destination set) via the `cachedLdapAuthorizationMap` element, found in the broker's `activemq.xml` configuration file. For more information, see Cached LDAP Authorization Module.

> **Note**
>
> To be able to use the `cachedLDAPAuthorizationMap` element in your broker's `activemq.xml` configuration file, you must choose the **LDAP Authentication and**

Authorization option when creating a configuration via the AWS Management Console, or set the creating a configuration via the AWS Management Console, or set the `authenticationStrategy` property to LDAP when creating a new configuration using the Amazon MQ API.

You must provide the following three attributes as part of the `cachedLDAPAuthorizationMap` element:

- `queueSearchBase`
- `topicSearchBase`
- `tempSearchBase`

Important

To prevent sensitive information from being directly placed in the broker's configuration file, Amazon MQ blocks the following attributes from being used in `cachedLdapAuthorizationMap`:

- `connectionURL`
- `connectionUsername`
- `connectionPassword`

When you create a broker, Amazon MQ substitutes the values you provide via the AWS Management Console, or in the `ldapServerMetadata` property of your API request, for the above attributes.

The following demonstrates a working example of the `cachedLdapAuthorizationMap`.

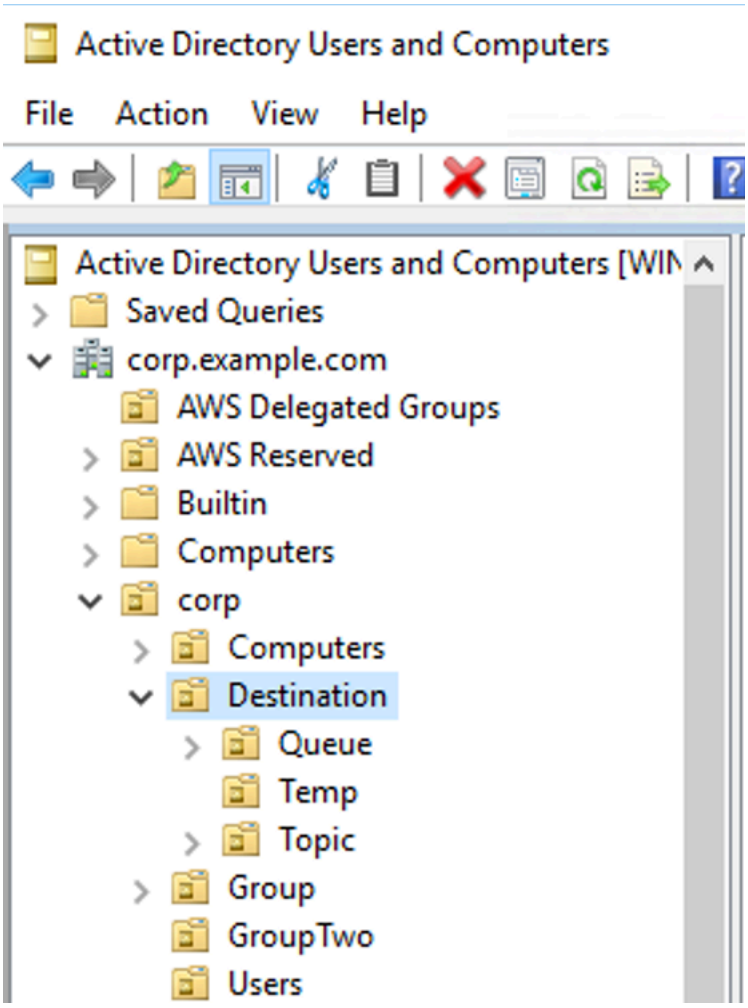
```
<authorizationPlugin>
    <map>
        <cachedLDAPAuthorizationMap
            queueSearchBase="ou=Queue,ou=Destination,ou=corp,dc=corp,dc=example,dc=com"
            topicSearchBase="ou=Topic,ou=Destination,ou=corp,dc=corp,dc=example,dc=com"
            tempSearchBase="ou=Temp,ou=Destination,ou=corp,dc=corp,dc=example,dc=com"
            refreshInterval="300000"
```

```
            legacyGroupMapping="false"
        />
    </map>
</authorizationPlugin>
```

These values identify the locations within the DIT where permissions for each type of destination are specified. So for the above example with AWS Managed Microsoft AD, using the same domain components of `corp`, `example`, and `com`, you would specify an organizational unit named `destination` to contain all your destination types. Within that OU, you would create one for `queues`, one for `topics`, and one for `temp` destinations.

This would mean that your queue search base, which provides authorization information for destinations of type queue, would have the following location in your DIT:

```
OU=Queue,OU=Destination,OU=corp,DC=corp,DC=example,DC=com
```



Similarly, permissions rules for topics and temp destinations would be located at the same level in the DIT:

```
OU=Topic,OU=Destination,OU=corp,DC=corp,DC=example,DC=com
OU=Temp,OU=Destination,OU=corp,DC=corp,DC=example,DC=com
```

Within the OU for each type of destination (queue, topic, temp), either a wildcard or specific destination name can be provided. For example, to provide an authorization rule for all queues that start with the prefix DEMO.EVENTS.$., you could create the following OU:

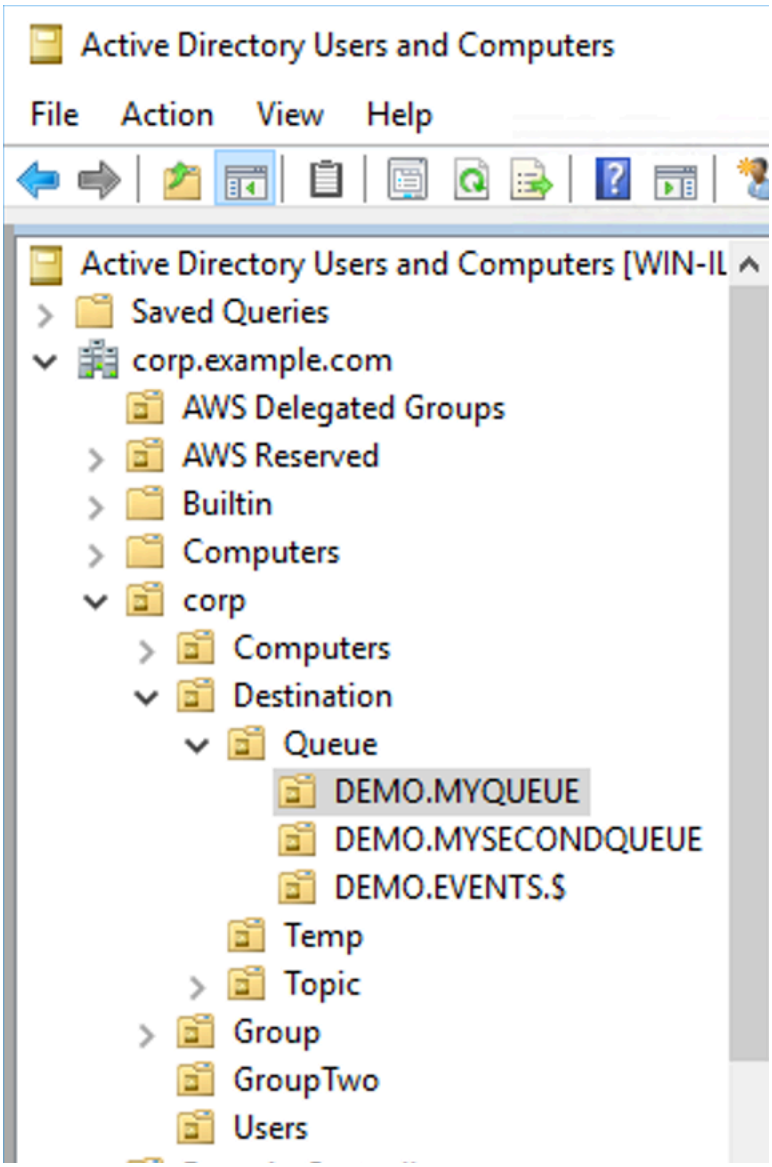
```
OU=DEMO.EVENTS.$,OU=Queue,OU=Destination,OU=corp,DC=corp,DC=example,DC=com
```

> **Note**
>
> The `DEMO.EVENTS.$` OU is within the `Queue` OU.

For more info on wildcards in ActiveMQ, see Wildcards

To provide authorization rules for specific queues, such as DEMO.MYQUEUE, specify something like the following:

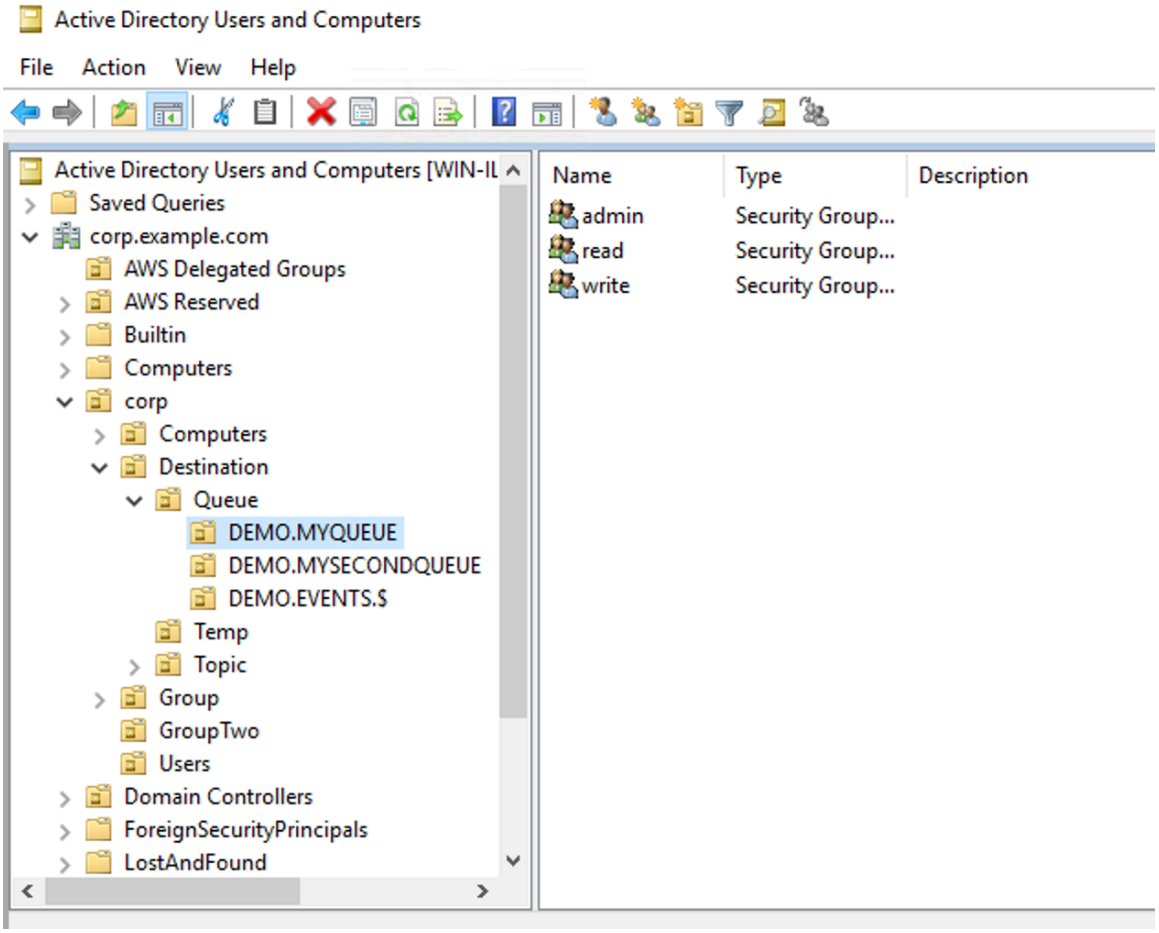
```
OU=DEMO.MYQUEUE,OU=Queue,OU=Destination,OU=corp,DC=corp,DC=example,DC=com
```



Security Groups

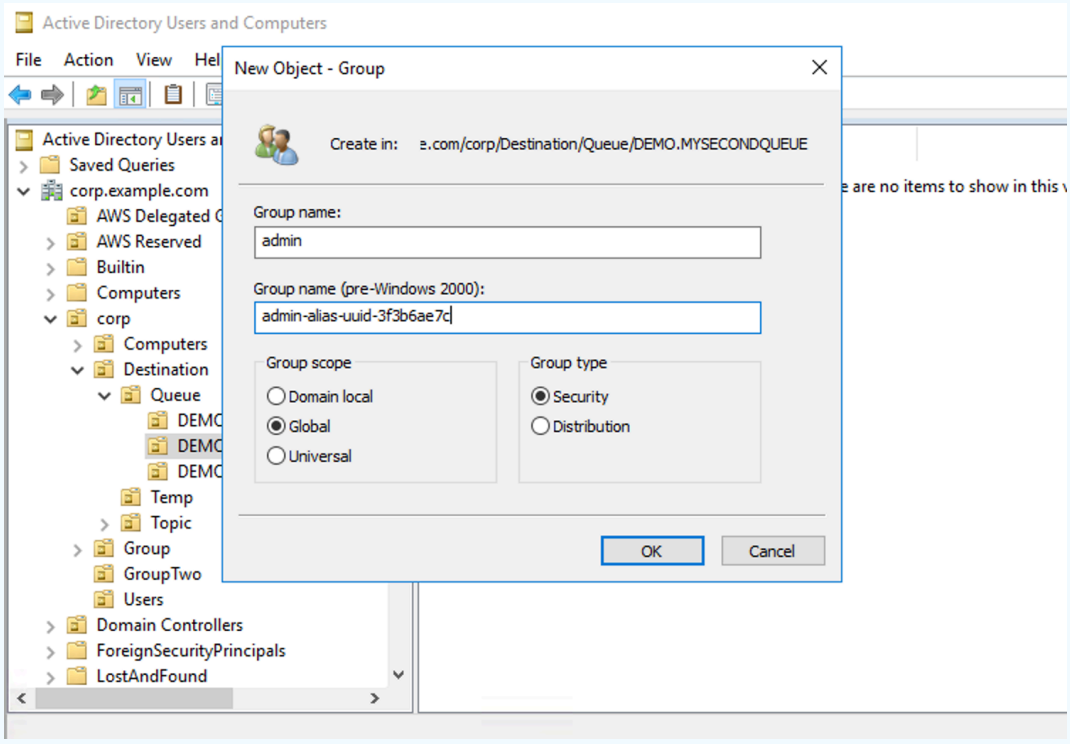
Within each OU that represents a destination or a wildcard, you must create three security groups. As with all permissions in ActiveMQ, these are read/write/admin permissions. For more information on what each of these permissions allows a user to do, see Security in the ActiveMQ documentation.

You must name these security groups `read`, `write`, and `admin`. Within each of these security groups, you can add users or groups, who will then have permission to perform the associated actions. You'll need these security groups for each wildcard destination set or individual destination.



> **Note**
>
> When you create the admin group, a conflict will arise with the group name. This conflict happens because the legacy pre-Windows 2000 rules do not allow groups to share the same name, even if the groups are in different locations of the DIT. The value in the **pre-**

Windows 2000 text box has no impact on the setup, but it must be globally unique. To avoid this conflict, you can append a `uuid` suffix to each `admin` group.



Adding a user to the `admin` security group for a particular destination will enable the user to create and delete that topic. Adding them to the `read` security group will enable them to read from the destination, and adding them to the `write` group will enable them to write to the destination.

In addition to adding individual users to security group permissions, you can also add entire groups. However, because ActiveMQ again hardcodes attribute names for groups, you must ensure the group you want to add has the object class `groupOfNames`, as shown in the activemq source code.

To do this, follow the same process as with the `uid` for users. See Configuring ID mappings in Active Directory Users and Computers for Windows Server 2016 (and subsequent) versions.

Step 3: (Optional) Connect to an AWS Lambda function

AWS Lambda can connect to and consume messages from your Amazon MQ broker. When you connect a broker to Lambda, you create an event source mapping that reads messages from a queue and invokes the function synchronously. The event source mapping you create reads messages from your broker in batches and converts them into a Lambda payload in the form of a JSON object.

To connect your broker to a Lambda function

1. Add the following IAM role permissions to your Lambda function execution role.

   - mq:DescribeBroker
   - ec2:CreateNetworkInterface
   - ec2:DeleteNetworkInterface
   - ec2:DescribeNetworkInterfaces
   - ec2:DescribeSecurityGroups
   - ec2:DescribeSubnets
   - ec2:DescribeVpcs
   - logs:CreateLogGroup
   - logs:CreateLogStream
   - logs:PutLogEvents
   - secretsmanager:GetSecretValue

   > **Note**
   >
   > Without the necessary IAM permissions, your function will not be able to successfully read records from Amazon MQ resources.

2. (Optional) If you have created a broker without public accessibility, you must do one of the following to allow Lambda to connect to your broker:

   - Configure one NAT gateway per public subnet. For more information, see Internet and service access for VPC-connected functions in the *AWS Lambda Developer Guide*.
   - Create a connection between your Amazon Virtual Private Cloud (Amazon VPC) and Lambda using a VPC endpoint. Your Amazon VPC must also connect to AWS Security Token Service (AWS STS) and Secrets Manager endpoints. For more information, see Configuring interface VPC endpoints for Lambda in the *AWS Lambda Developer Guide*.

3. Configure your broker as an event source for a Lambda function using the AWS Management Console. You can also use the `create-event-source-mapping` AWS Command Line Interface command.
4. Write some code for your Lambda function to process the messages consumed from your broker. The Lambda payload that retrieved by your event source mapping depends on the

engine type of the broker. The following is an example of a Lambda payload for an Amazon MQ for ActiveMQ queue.

> **Note**
>
> In the example, `testQueue` is the name of the queue.

```
{
  "eventSource": "aws:amq",
  "eventSourceArn": "arn:aws:mq:us-
west-2:112556298976:broker:test:b-9bcfa592-423a-4942-879d-eb284b418fc8",
  "messages": {
    [
      {
        "messageID": "ID:b-9bcfa592-423a-4942-879d-eb284b418fc8-1.mq.us-
west-2.amazonaws.com-37557-1234520418293-4:1:1:1:1",
        "messageType": "jms/text-message",
        "data": "QUJDOkFBQUE=",
        "connectionId": "myJMSCoID",
        "redelivered": false,
        "destination": {
          "physicalname": "testQueue"
        },
        "timestamp": 1598827811958,
        "brokerInTime": 1598827811958,
        "brokerOutTime": 1598827811959
      },
      {
        "messageID": "ID:b-9bcfa592-423a-4942-879d-eb284b418fc8-1.mq.us-
west-2.amazonaws.com-37557-1234520418293-4:1:1:1:1",
        "messageType":"jms/bytes-message",
        "data": "3DTOOW7crj51prgVLQaGQ82S48k=",
        "connectionId": "myJMSCoID1",
        "persistent": false,
        "destination": {
          "physicalname": "testQueue"
        },
        "timestamp": 1598827811958,
        "brokerInTime": 1598827811958,
        "brokerOutTime": 1598827811959
      }
```

```
    ]
  }
}
```

For more information about connecting Amazon MQ to Lambda, the options Lambda supports for an Amazon MQ event source, and event source mapping errors, see Using Lambda with Amazon MQ in the *AWS Lambda Developer Guide*.

Creating an ActiveMQ broker user

An ActiveMQ *user* is a person or an application that can access the queues and topics of an ActiveMQ broker. You can configure users to have specific permissions. For example, you can allow some users to access the ActiveMQ Web Console.

A *group* is a semantic label. You can assign a group to a user and configure permissions for groups to send to, receive from, and administer specific queues and topics.

> **Note**
>
> You can't configure groups independently of users. A group label is created when you add at least one user to it and deleted when you remove all users from it.

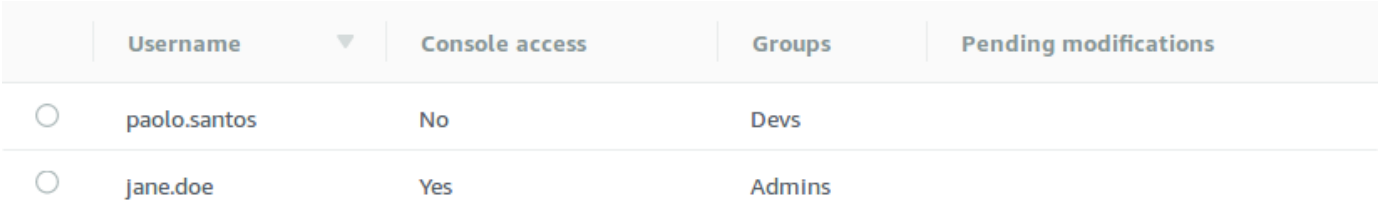
> **Note**
>
> The `activemq-webconsole` group in ActiveMQ on Amazon MQ has admin permissions on all queues and topics. All users in this group will have admin access.

The following examples show how you can create, edit, and delete Amazon MQ broker users using the AWS Management Console.

Create a new ActiveMQ broker user

1. Sign in to the Amazon MQ console.
2. From the broker list, choose the name of your broker (for example, **MyBroker**) and then choose **View details**.

   On the ***MyBroker*** page, in the **Users** section, all the users for this broker are listed.



3. Choose **Create user**.
4. In the **Create user** dialog box, type a **Username** and **Password**.
5. (Optional) Type the names of groups to which the user belongs, separated by commas (for example: `Devs, Admins`).
6. (Optional) To enable the user to access the ActiveMQ Web Console, choose **ActiveMQ Web Console**.
7. Choose **Create user**.

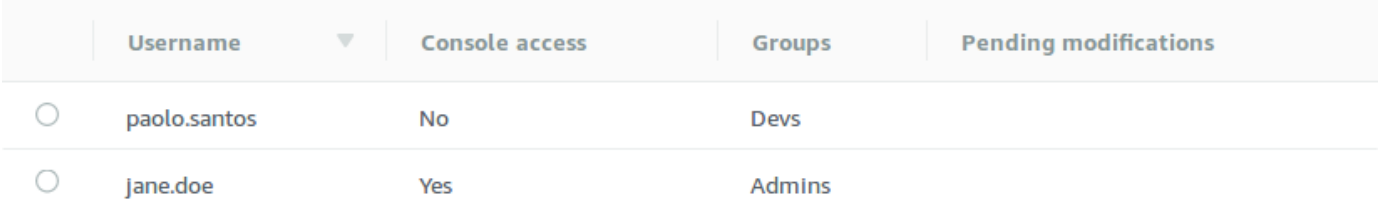
> **Important**
>
> Making changes to a user does *not* apply the changes to the user immediately. To apply your changes, you must wait for the next maintenance window or reboot the broker.

Edit an ActiveMQ broker user

To edit an existing user, do the following:

1. Sign in to the Amazon MQ console.
2. From the broker list, choose the name of your broker (for example, **MyBroker**) and then choose **View details**.

   On the ***MyBroker*** page, in the **Users** section, all the users for this broker are listed.



3. Select your sign-in credentials and choose **Edit**.

   The **Edit user** dialog box is displayed.

4. (Optional) Type a new **Password**.
5. (Optional) Add or remove the names of groups to which the user belongs, separated by commas (for example: `Managers, Admins`).
6. (Optional) To enable the user to access the ActiveMQ Web Console, choose **ActiveMQ Web Console**.
7. To save the changes to the user, choose **Done**.

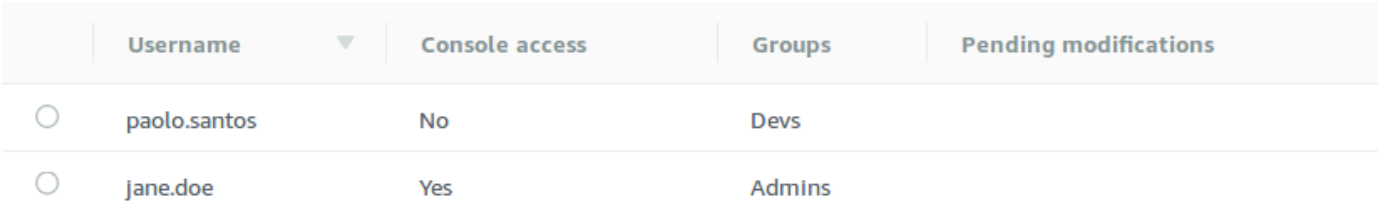
> **Important**
>
> Making changes to a user does *not* apply the changes to the user immediately. To apply your changes, you must wait for the next maintenance window or reboot the broker.

Delete an ActiveMQ broker user

When you no longer need a user, you can delete the user.

1. Sign in to the Amazon MQ console.
2. From the broker list, choose the name of your broker (for example, **MyBroker**) and then choose **View details**.

   On the ***MyBroker*** page, in the **Users** section, all the users for this broker are listed.



3. Select a your sign-in credentials (for example, ***MyUser***) and then choose **Delete**.
4. To confirm deleting the user, in the **Delete *MyUser*?** dialog box, choose **Delete**.

> **Important**
>
> Making changes to a user does *not* apply the changes to the user immediately. To apply your changes, you must wait for the next maintenance window or reboot the broker.

Working examples of using Java Message Service (JMS) with ActiveMQ

The following examples show how you can work with ActiveMQ programmatically:

- The OpenWire example Java code connects to a broker, creates a queue, and sends and receives a message. For a detailed breakdown and explanation, see Connecting a Java application to your broker.
- The MQTT example Java code connects to a broker, creates a topic, and publishes and receives a message.
- The STOMP+WSS example Java code connects to a broker, creates a queue, and publishes and receives a message.

Prerequisites

Enable VPC Attributes

To ensure that your broker is accessible within your VPC, you must enable the `enableDnsHostnames` and `enableDnsSupport` VPC attributes. For more information, see DNS Support in your VPC in the *Amazon VPC User Guide*.

Enable inbound Connections

To work with Amazon MQ programmatically, you must use inbound connections.

1. Sign in to the Amazon MQ console.
2. From the broker list, choose the name of your broker (for example, **MyBroker**).
3. On the ***MyBroker*** page, in the **Connections** section, note the addresses and ports of the broker's web console URL and wire-level protocols.
4. In the **Details** section, under **Security and network**, choose the name of your security group or .

   The **Security Groups** page of the EC2 Dashboard is displayed.
5. From the security group list, choose your security group.
6. At the bottom of the page, choose **Inbound**, and then choose **Edit**.
7. In the **Edit inbound rules** dialog box, add a rule for every URL or endpoint that you want to be publicly accessible (the following example shows how to do this for a broker web console).

   a. Choose **Add Rule**.

b. For **Type**, select **Custom TCP**.

c. For **Port Range**, type the web console port (8162).

d. For **Source**, leave **Custom** selected and then type the IP address of the system that you want to be able to access the web console (for example, `192.0.2.1`).

e. Choose **Save**.

Your broker can now accept inbound connections.

Add Java dependencies

OpenWire

Add the `activemq-client.jar` and `activemq-pool.jar` packages to your Java class path. The following example shows these dependencies in a Maven project `pom.xml` file.

```
<dependencies>
    <dependency>
        <groupId>org.apache.activemq</groupId>
        <artifactId>activemq-client</artifactId>
        <version>5.15.16</version>
    </dependency>
    <dependency>
        <groupId>org.apache.activemq</groupId>
        <artifactId>activemq-pool</artifactId>
        <version>5.15.16</version>
    </dependency>
</dependencies>
```

For more information about `activemq-client.jar`, see Initial Configuration in the Apache ActiveMQ documentation.

MQTT

Add the `org.eclipse.paho.client.mqttv3.jar` package to your Java class path. The following example shows this dependency in a Maven project `pom.xml` file.

```
<dependencies>
                    <dependency>
                        <groupId>org.eclipse.paho</groupId>
                        <artifactId>org.eclipse.paho.client.mqttv3</artifactId>
                        <version>1.2.0</version>
```

```
                    </dependency>
                    </dependencies>
```

For more information about `org.eclipse.paho.client.mqttv3.jar`, see Eclipse Paho Java Client.

STOMP+WSS

Add the following packages to your Java class path:

- `spring-messaging.jar`
- `spring-websocket.jar`
- `javax.websocket-api.jar`
- `jetty-all.jar`
- `slf4j-simple.jar`
- `jackson-databind.jar`

The following example shows these dependencies in a Maven project `pom.xml` file.

```
<dependencies>
                    <dependency>
                        <groupId>org.springframework</groupId>
                        <artifactId>spring-messaging</artifactId>
                        <version>5.0.5.RELEASE</version>
                    </dependency>
                    <dependency>
                        <groupId>org.springframework</groupId>
                        <artifactId>spring-websocket</artifactId>
                        <version>5.0.5.RELEASE</version>
                    </dependency>
                    <dependency>
                        <groupId>javax.websocket</groupId>
                        <artifactId>javax.websocket-api</artifactId>
                        <version>1.1</version>
                    </dependency>
                    <dependency>
                        <groupId>org.eclipse.jetty.aggregate</groupId>
                        <artifactId>jetty-all</artifactId>
                        <type>pom</type>
                        <version>9.3.3.v20150827</version>
                    </dependency>
```

```
                    <dependency>
                        <groupId>org.slf4j</groupId>
                        <artifactId>slf4j-simple</artifactId>
                        <version>1.6.6</version>
                    </dependency>
                    <dependency>
                        <groupId>com.fasterxml.jackson.core</groupId>
                        <artifactId>jackson-databind</artifactId>
                        <version>2.5.0</version>
                    </dependency>
                    </dependencies>
```

For more information, see STOMP Support in the Spring Framework documentation.

AmazonMQExample.java

Important

In the following example code, producers and consumers run in a single thread. For production systems (or to test broker instance failover), make sure that your producers and consumers run on separate hosts or threads.

OpenWire

```
/*
 * Copyright 2010-2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 *  https://aws.amazon.com/apache2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 *
 */
```

```
                    import org.apache.activemq.ActiveMQConnectionFactory;
                    import org.apache.activemq.jms.pool.PooledConnectionFactory;

                    import javax.jms.*;

                    public class AmazonMQExample {

                    // Specify the connection parameters.
                    private final static String WIRE_LEVEL_ENDPOINT
                            = "ssl://b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-
east-2.amazonaws.com:61617";
                    private final static String ACTIVE_MQ_USERNAME =
 "MyUsername123";
                    private final static String ACTIVE_MQ_PASSWORD =
 "MyPassword456";

                    public static void main(String[] args) throws JMSException {
                        final ActiveMQConnectionFactory connectionFactory =
                                createActiveMQConnectionFactory();
                        final PooledConnectionFactory pooledConnectionFactory =
                                createPooledConnectionFactory(connectionFactory);

                        sendMessage(pooledConnectionFactory);
                        receiveMessage(connectionFactory);

                        pooledConnectionFactory.stop();
                    }

                    private static void
                    sendMessage(PooledConnectionFactory pooledConnectionFactory)
 throws JMSException {
                        // Establish a connection for the producer.
                        final Connection producerConnection =
 pooledConnectionFactory
                                .createConnection();
                        producerConnection.start();

                        // Create a session.
                        final Session producerSession = producerConnection
                                .createSession(false, Session.AUTO_ACKNOWLEDGE);

                        // Create a queue named "MyQueue".
                        final Destination producerDestination = producerSession
                                .createQueue("MyQueue");
```

```
                        // Create a producer from the session to the queue.
                        final MessageProducer producer = producerSession
                                .createProducer(producerDestination);
                        producer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);

                        // Create a message.
                        final String text = "Hello from Amazon MQ!";
                        final TextMessage producerMessage = producerSession
                                .createTextMessage(text);

                        // Send the message.
                        producer.send(producerMessage);
                        System.out.println("Message sent.");

                        // Clean up the producer.
                        producer.close();
                        producerSession.close();
                        producerConnection.close();
                    }

                    private static void
                    receiveMessage(ActiveMQConnectionFactory connectionFactory)
throws JMSException {
                        // Establish a connection for the consumer.
                        // Note: Consumers should not use PooledConnectionFactory.
                        final Connection consumerConnection =
connectionFactory.createConnection();
                        consumerConnection.start();

                        // Create a session.
                        final Session consumerSession = consumerConnection
                                .createSession(false, Session.AUTO_ACKNOWLEDGE);

                        // Create a queue named "MyQueue".
                        final Destination consumerDestination = consumerSession
                                .createQueue("MyQueue");

                        // Create a message consumer from the session to the queue.
                        final MessageConsumer consumer = consumerSession
                                .createConsumer(consumerDestination);

                        // Begin to wait for messages.
                        final Message consumerMessage = consumer.receive(1000);
```

```
                        // Receive the message when it arrives.
                        final TextMessage consumerTextMessage = (TextMessage)
consumerMessage;
                        System.out.println("Message received: " +
consumerTextMessage.getText());

                        // Clean up the consumer.
                        consumer.close();
                        consumerSession.close();
                        consumerConnection.close();
                    }

                    private static PooledConnectionFactory
                    createPooledConnectionFactory(ActiveMQConnectionFactory
connectionFactory) {
                        // Create a pooled connection factory.
                        final PooledConnectionFactory pooledConnectionFactory =
                                new PooledConnectionFactory();

pooledConnectionFactory.setConnectionFactory(connectionFactory);
                        pooledConnectionFactory.setMaxConnections(10);
                        return pooledConnectionFactory;
                    }

                    private static ActiveMQConnectionFactory
createActiveMQConnectionFactory() {
                        // Create a connection factory.
                        final ActiveMQConnectionFactory connectionFactory =
                                new ActiveMQConnectionFactory(WIRE_LEVEL_ENDPOINT);

                        // Pass the sign-in credentials.
                        connectionFactory.setUserName(ACTIVE_MQ_USERNAME);
                        connectionFactory.setPassword(ACTIVE_MQ_PASSWORD);
                        return connectionFactory;
                    }
                    }
```

MQTT

```
/*
 * Copyright 2010-2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
```

```
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 *  https://aws.amazon.com/apache2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 *
 */

                    import org.eclipse.paho.client.mqttv3.*;

                    public class AmazonMQExampleMqtt implements MqttCallback {

                    // Specify the connection parameters.
                    private final static String WIRE_LEVEL_ENDPOINT =
                            "ssl://b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-
east-2.amazonaws.com:8883";
                    private final static String ACTIVE_MQ_USERNAME =
 "MyUsername123";
                    private final static String ACTIVE_MQ_PASSWORD =
 "MyPassword456";

                    public static void main(String[] args) throws Exception {
                        new AmazonMQExampleMqtt().run();
                    }

                    private void run() throws MqttException, InterruptedException {

                        // Specify the topic name and the message text.
                        final String topic = "myTopic";
                        final String text = "Hello from Amazon MQ!";

                        // Create the MQTT client and specify the connection
 options.
                        final String clientId = "abc123";
                        final MqttClient client = new
 MqttClient(WIRE_LEVEL_ENDPOINT, clientId);
                        final MqttConnectOptions connOpts = new
 MqttConnectOptions();
```

```
                        // Pass the sign-in credentials.
                        connOpts.setUserName(ACTIVE_MQ_USERNAME);
                        connOpts.setPassword(ACTIVE_MQ_PASSWORD.toCharArray());

                        // Create a session and subscribe to a topic filter.
                        client.connect(connOpts);
                        client.setCallback(this);
                        client.subscribe("+");

                        // Create a message.
                        final MqttMessage message = new
MqttMessage(text.getBytes());

                        // Publish the message to a topic.
                        client.publish(topic, message);
                        System.out.println("Published message.");

                        // Wait for the message to be received.
                        Thread.sleep(3000L);

                        // Clean up the connection.
                        client.disconnect();
                    }

                    @Override
                    public void connectionLost(Throwable cause) {
                        System.out.println("Lost connection.");
                    }

                    @Override
                    public void messageArrived(String topic, MqttMessage message)
throws MqttException {
                        System.out.println("Received message from topic " + topic +
": " + message);
                    }

                    @Override
                    public void deliveryComplete(IMqttDeliveryToken token) {
                        System.out.println("Delivered message.");
                    }
                    }
```

STOMP+WSS

```
/*
 * Copyright 2010-2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 *  https://aws.amazon.com/apache2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 *
 */

                    import
 org.springframework.messaging.converter.StringMessageConverter;
                    import org.springframework.messaging.simp.stomp.*;
                    import org.springframework.web.socket.WebSocketHttpHeaders;
                    import org.springframework.web.socket.client.WebSocketClient;
                    import
 org.springframework.web.socket.client.standard.StandardWebSocketClient;
                    import
 org.springframework.web.socket.messaging.WebSocketStompClient;

                    import java.lang.reflect.Type;

                    public class AmazonMQExampleStompWss {

                    // Specify the connection parameters.
                    private final static String DESTINATION = "/queue";
                    private final static String WIRE_LEVEL_ENDPOINT =
                            "wss://b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-
east-2.amazonaws.com:61619";
                    private final static String ACTIVE_MQ_USERNAME =
 "MyUsername123";
                    private final static String ACTIVE_MQ_PASSWORD =
 "MyPassword456";

                    public static void main(String[] args) throws Exception {
```

```
                        final AmazonMQExampleStompWss example = new
AmazonMQExampleStompWss();

                        final StompSession stompSession = example.connect();
                        System.out.println("Subscribed to a destination using
session.");
                        example.subscribeToDestination(stompSession);

                        System.out.println("Sent message to session.");
                        example.sendMessage(stompSession);
                        Thread.sleep(60000);
                    }

                    private StompSession connect() throws Exception {
                        // Create a client.
                        final WebSocketClient client = new
StandardWebSocketClient();
                        final WebSocketStompClient stompClient = new
WebSocketStompClient(client);
                        stompClient.setMessageConverter(new
StringMessageConverter());

                        final WebSocketHttpHeaders headers = new
WebSocketHttpHeaders();

                        // Create headers with authentication parameters.
                        final StompHeaders head = new StompHeaders();
                        head.add(StompHeaders.LOGIN, ACTIVE_MQ_USERNAME);
                        head.add(StompHeaders.PASSCODE, ACTIVE_MQ_PASSWORD);

                        final StompSessionHandler sessionHandler = new
MySessionHandler();

                        // Create a connection.
                        return stompClient.connect(WIRE_LEVEL_ENDPOINT, headers,
head,
                                sessionHandler).get();
                    }

                    private void subscribeToDestination(final StompSession
stompSession) {
                        stompSession.subscribe(DESTINATION, new MyFrameHandler());
                    }
```

```
                    private void sendMessage(final StompSession stompSession) {
                        stompSession.send(DESTINATION, "Hello from Amazon
MQ!".getBytes());
                    }

                    private static class MySessionHandler extends
StompSessionHandlerAdapter {
                        public void afterConnected(final StompSession stompSession,
                                                   final StompHeaders stompHeaders) {
                            System.out.println("Connected to broker.");
                        }
                    }

                    private static class MyFrameHandler implements StompFrameHandler
{
                        public Type getPayloadType(final StompHeaders headers) {
                            return String.class;
                        }

                        public void handleFrame(final StompHeaders stompHeaders,
                                                final Object message) {
                            System.out.print("Received message from topic: " +
message);
                        }
                    }
                    }
```

Managing Amazon MQ for ActiveMQ engine versions

Apache ActiveMQ organizes version numbers according to semantic versioning specification as `X.Y.Z`. In Amazon MQ for ActiveMQ implementations, `X` denotes the major version, `Y` represents the minor version, and `Z` denotes the patch version number. Amazon MQ considers a version change to be major if the major version numbers change. For example, upgrading from version **5**.17 to **6**.0 is considered a *major version upgrade*. A version change is considered minor if only the minor or patch version number changes. For example, upgrading from version 5.**18** to 5.**19** is considered a *minor version upgrade*. When `autoMinorVersionUpgrade` is turned on, Amazon MQ upgrades your broker to the newest available patch version.

Amazon MQ for ActiveMQ recommends all brokers use the latest supported minor version. For instructions on how to upgrade your broker engine version, see Upgrading an Amazon MQ broker engine version.

Supported engine versions on Amazon MQ for ActiveMQ

The Amazon MQ version support calendar indicates when a broker engine version will reach end of support. When a version reaches end of support, Amazon MQ upgrades all brokers on this version to the next supported version automatically. This upgrade takes place during your broker's scheduled maintenance windows, within the 45 days following the end-of-support date.

Amazon MQ provides at least a 90 day notice before a version reaches end of support. We recommend upgrading your broker before the end-of-support date to prevent any disruptions. Additionally, you cannot create new brokers on versions scheduled for end of support within 30 days of the end of support date.

Apache ActiveMQ version	End of support on Amazon MQ
ActiveMQ 5.19 (recommended)	
ActiveMQ 5.18	
ActiveMQ 5.17	June 16, 2025
ActiveMQ 5.16	November 15, 2024
ActiveMQ 5.15	September 16, 2024

When you create a new Amazon MQ for ActiveMQ broker, you can specify any supported ActiveMQ engine version. If you do not specify the engine version number when creating a broker, Amazon MQ automatically defaults to the latest engine version number.

Engine version upgrades

You can manually upgrade your broker at any time to the next supported major or minor version. When you turn on automatic minor version upgrades, Amazon MQ will upgrade your broker to the latest supported patch version during the maintenance window.

For more information about manually upgrading your broker, see the section called “Upgrading the engine version”.

Listing supported engine versions

You can list all supported minor and major engine versions by using the `describe-broker-instance-options` AWS CLI command.

```
aws mq describe-broker-instance-options
```

To filter the results by engine and instance type use the `--engine-type` and `--host-instance-type` options as shown in the following.

```
aws mq describe-broker-instance-options --engine-type engine-type --host-instance-
type instance-type
```

For example, to filter the results for ActiveMQ, and `mq.m5.large` instance type, replace *`engine-type`* with `ACTIVEMQ` and *`instance-type`* with `mq.m5.large`.

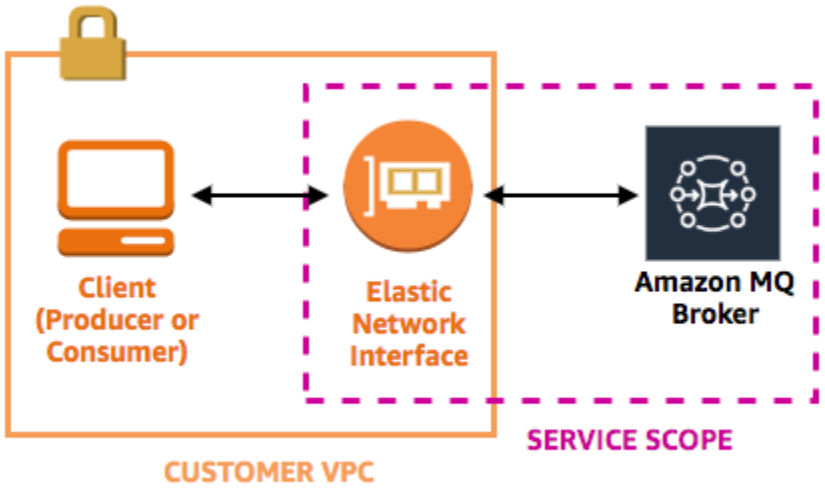
Amazon MQ for ActiveMQ best practices

Use this as a reference to quickly find recommendations for maximizing performance and minimizing throughput costs when working with ActiveMQ brokers on Amazon MQ.

Never Modify or Delete the Amazon MQ Elastic Network Interface

When you first create an Amazon MQ broker, Amazon MQ provisions an elastic network interface in the Virtual Private Cloud (VPC) under your account and, thus, requires a number of EC2 permissions. The network interface allows your client (producer or consumer) to communicate with the Amazon MQ broker. The network interface is considered to be within the *service scope* of Amazon MQ, despite being part of your account's VPC.

> **Warning**
>
> You must not modify or delete this network interface. Modifying or deleting the network interface can cause a permanent loss of connection between your VPC and your broker.



Always Use Connection Pooling

In a scenario with a single producer and single consumer (such as the Getting started: Creating and connecting to an ActiveMQ broker tutorial), you can use a single `ActiveMQConnectionFactory` class for every producer and consumer. For example:

```
// Create a connection factory.
final ActiveMQConnectionFactory connectionFactory = new
 ActiveMQConnectionFactory(wireLevelEndpoint);

// Pass the sign-in credentials.
connectionFactory.setUserName(activeMqUsername);
connectionFactory.setPassword(activeMqPassword);

// Establish a connection for the consumer.
final Connection consumerConnection = connectionFactory.createConnection();
consumerConnection.start();
```

However, in more realistic scenarios with multiple producers and consumers, it can be costly and inefficient to create a large number of connections for multiple producers. In these scenarios, you should group multiple producer requests using the `PooledConnectionFactory` class. For example:

Note

Message consumers should *never* use the `PooledConnectionFactory` class.

```
// Create a connection factory.
final ActiveMQConnectionFactory connectionFactory = new
 ActiveMQConnectionFactory(wireLevelEndpoint);

// Pass the sign-in credentials.
connectionFactory.setUserName(activeMqUsername);
connectionFactory.setPassword(activeMqPassword);

// Create a pooled connection factory.
final PooledConnectionFactory pooledConnectionFactory = new PooledConnectionFactory();
pooledConnectionFactory.setConnectionFactory(connectionFactory);
pooledConnectionFactory.setMaxConnections(10);

// Establish a connection for the producer.
final Connection producerConnection = pooledConnectionFactory.createConnection();
producerConnection.start();
```

Always Use the Failover Transport to Connect to Multiple Broker Endpoints

If you need your application to connect to multiple broker endpoints—for example, when you use an active/standby deployment mode or when you migrate from an on-premises message broker to Amazon MQ—use the Failover Transport to allow your consumers to randomly connect to either one. For example:

```
failover:(ssl://b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-
east-2.amazonaws.com:61617,ssl://b-9876l5k4-32ji-109h-8gfe-7d65c4b132a1-2.mq.us-
west-2.amazonaws.com:61617)?randomize=true
```

Important

Multi-availability zone brokers can experience failovers during maintenance windows and broker restarts. Use the Failover Transport to ensure your broker availability.

Avoid Using Message Selectors

It is possible to use JMS selectors to attach filters to topic subscriptions (to route messages to consumers based on their content). However, the use of JMS selectors fills up the Amazon MQ broker's filter buffer, preventing it from filtering messages.

In general, avoid letting consumers route messages because, for optimal decoupling of consumers and producers, both the consumer and the producer should be ephemeral.

Prefer Virtual Destinations to Durable Subscriptions

A durable subscription can help ensure that the consumer receives all messages published to a topic, for example, after a lost connection is restored. However, the use of durable subscriptions also precludes the use of competing consumers and might have performance issues at scale. Consider using virtual destinations instead.

If using Amazon VPC peering, avoid client IPs in CIDR range `10.0.0.0/16`

If you are setting up Amazon VPC peering between on-premise infrastructure and your Amazon MQ broker, you must not configure client connections with IPs in CIDR range `10.0.0.0/16`.

Disable Concurrent Store and Dispatch for Queues with Slow Consumers

By default, Amazon MQ optimizes for queues with fast consumers:

- Consumers are considered *fast* if they are able to keep up with the rate of messages generated by producers.
- Consumers are considered *slow* if a queue builds up a backlog of unacknowledged messages, potentially causing a decrease in producer throughput.

To instruct Amazon MQ to optimize for queues with slow consumers, set the `concurrentStoreAndDispatchQueues` attribute to `false`. For an example configuration, see `concurrentStoreAndDispatchQueues`.

Choose the Correct Broker Instance Type for the Best Throughput

The message throughput of a broker instance type depends on your application's use case and the following factors:

- Use of ActiveMQ in persistent mode
- Message size
- The number of producers and consumers
- The number of destinations

Understanding the relationship between message size, latency, and throughput

Depending on your use case, a larger broker instance type might not necessarily improve system throughput. When ActiveMQ writes messages to durable storage, the size of your messages determines your system's limiting factor:

- If your messages are smaller than 100 KB, persistent storage latency is the limiting factor.
- If your messages are larger than 100 KB, persistent storage throughput is the limiting factor.

When you use ActiveMQ in persistent mode, writing to storage normally occurs when there are either few consumers or when the consumers are slow. In non-persistent mode, writing to storage also occurs with slow consumers if the heap memory of the broker instance is full.

To determine the best broker instance type for your application, we recommend testing different broker instance types. For more information, see Broker instance types and also Measuring the Throughput for Amazon MQ using the JMS Benchmark.

Use cases for larger broker instance types

There are three common use cases when larger broker instance types improve throughput:

- **Non-persistent mode** – When your application is less sensitive to losing messages during broker instance failover (for example, when broadcasting sports scores), you can often use ActiveMQ's non-persistent mode. In this mode, ActiveMQ writes messages to persistent storage only if the heap memory of the broker instance is full. Systems that use non-persistent mode can benefit from the higher amount of memory, faster CPU, and faster network available on larger broker instance types.

- **Fast consumers** – When active consumers are available and the `concurrentStoreAndDispatchQueues` flag is enabled, ActiveMQ allows messages to flow directly from producer to consumer without sending messages to storage (even in persistent mode). If your application can consume messages quickly (or if you can design your consumers to do this), your application can benefit from a larger broker instance type. To let your application consume messages more quickly, add consumer threads to your application instances or scale up your application instances vertically or horizontally.
- **Batched transactions** – When you use persistent mode and send multiple messages per transaction, you can achieve an overall higher message throughput by using larger broker instance types. For more information, see Should I Use Transactions? in the ActiveMQ documentation.

Choose the correct broker storage type for the best throughput

To take advantage of high durability and replication across multiple Availability Zones, use Amazon EFS. To take advantage of low latency and high throughput, use Amazon EBS. For more information, see Storage.

Configure Your Network of Brokers Correctly

When you create a network of brokers, configure it correctly for your application:

- **Enable persistent mode** – Because (relative to its peers) each broker instance acts like a producer or a consumer, networks of brokers don't provide distributed replication of messages. The first broker that acts as a consumer receives a message and persists it to storage. This broker sends an acknowledgement to the producer and forwards the message to the next broker. When the second broker acknowledges the persistence of the message, the first broker deletes the message.

  If persistent mode is disabled, the first broker acknowledges the producer without persisting the message to storage. For more information, see Replicated Message Store and What is the difference between persistent and non-persistent delivery? in the Apache ActiveMQ documentation.
- **Don't disable advisory messages for broker instances** – For more information, see Advisory Message in the Apache ActiveMQ documentation.

- **Don't use multicast broker discovery** – Amazon MQ doesn't support broker discovery using multicast. For more information, see What is the difference between discovery, multicast, and zeroconf? in the Apache ActiveMQ documentation.

Avoid slow restarts by recovering prepared XA transactions

ActiveMQ supports distributed (XA) transactions. Knowing how ActiveMQ processes XA transactions can help avoid slow recovery times for broker restarts and failovers in Amazon MQ

Unresolved prepared XA transactions are replayed on every restart. If these remain unresolved, their number will grow over time, significantly increasing the time needed to start up the broker. This affects restart and failover time. You must resolve these transactions with a `commit()` or a `rollback()` so that performance doesn't degrade over time.

To monitor your unresolved prepared XA transactions, you can use the `JournalFilesForFastRecovery` metric in Amazon CloudWatch Logs. If this number is increasing, or is consistently higher than 1, you should recover your unresolved transactions with code similar to the following example. For more information, see Quotas in Amazon MQ.

The following example code walks through prepared XA transactions and closes them with a `rollback()`.

```
import org.apache.activemq.ActiveMQXAConnectionFactory;

import javax.jms.XAConnection;
import javax.jms.XASession;
import javax.transaction.xa.XAResource;
import javax.transaction.xa.Xid;

public class RecoverXaTransactions {
    private static final ActiveMQXAConnectionFactory ACTIVE_MQ_CONNECTION_FACTORY;
    final static String WIRE_LEVEL_ENDPOINT =
            "tcp://localhost:61616";;
    static {
        final String activeMqUsername = "MyUsername123";
        final String activeMqPassword = "MyPassword456";
        ACTIVE_MQ_CONNECTION_FACTORY = new
 ActiveMQXAConnectionFactory(activeMqUsername, activeMqPassword, WIRE_LEVEL_ENDPOINT);
        ACTIVE_MQ_CONNECTION_FACTORY.setUserName(activeMqUsername);
        ACTIVE_MQ_CONNECTION_FACTORY.setPassword(activeMqPassword);
    }
```

```
    public static void main(String[] args) {
        try {
            final XAConnection connection =
 ACTIVE_MQ_CONNECTION_FACTORY.createXAConnection();
            XASession xaSession = connection.createXASession();
            XAResource xaRes = xaSession.getXAResource();

            for (Xid id : xaRes.recover(XAResource.TMENDRSCAN)) {
                xaRes.rollback(id);
            }
            connection.close();

        } catch (Exception e) {
        }
    }
}
```

In a real-world scenario, you could check your prepared XA transactions against your XA Transaction Manager. Then you can decide whether to handle each prepared transaction with a `rollback()` or a `commit()`.

Using Amazon MQ for RabbitMQ

Amazon MQ makes it easy to create a message broker with the computing and storage resources that fit your needs. You can create, manage, and delete brokers using the AWS Management Console, Amazon MQ REST API, or the AWS Command Line Interface.

This section describes the basic elements of a message broker for ActiveMQ and RabbitMQ engine types, lists available Amazon MQ broker instance types and their statuses, and provides an overview of broker architecture and configuration options.

To learn about Amazon MQ REST APIs, see the *Amazon MQ REST API Reference*.

What is an Amazon MQ for RabbitMQ broker?

A *broker* is a message broker environment running on Amazon MQ. It is the basic building block of Amazon MQ. The combined description of the broker instance *class* (`m7g`) and *size* (`large`, `medium`) is called the *broker instance type* (for example, `mq.m7g.large`).

- A *single-instance broker* consists of one broker in one Availability Zone behind a Network Load Balancer (NLB). The broker communicates with your application and with an Amazon EBS storage volume.
- A *cluster deployment* is a logical grouping of three RabbitMQ broker nodes behind a Network Load Balancer, each sharing users, queues, and a distributed state across multiple Availability Zones (AZ).

For more information, see Deploying a RabbitMQ broker.

Listener ports

Amazon MQ managed RabbitMQ brokers support the following listener ports for application-level connectivity via `amqps`. You can also use these ports for client connections using the RabbitMQ web console and the management API. All connections use TLS encryption for security.

- Listener port 5671 - Used for secure AMQP connections made via the secure AMQP URL. This port supports both AMQP 0-9-1 and AMQP 1.0 protocols in RabbitMQ 4. For example, given a broker with broker ID `b-c8352341-ec91-4a78-ad9c-a43f23d325bb`, deployed in the `us-west-2` region, the following is the broker's full `amqps` URL: `b-c8352341-ec91-4a78-ad9c-a43f23d325bb.mq.us-west-2.amazonaws.com:5671`.

- Listener ports 443 and 15671 - You can use both listener ports interchangeably to access a broker via the RabbitMQ web console or the management API. Port 443 provides standard HTTPS access, while port 15671 is the traditional RabbitMQ management port with TLS encryption.

Attributes

A RabbitMQ broker has several attributes:

- A name. For example, `MyBroker`.
- An ID. For example, `b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9`.
- An Amazon Resource Name (ARN). For example, `arn:aws:mq:us-east-2:123456789012:broker:MyBroker:b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9`.
- A RabbitMQ web console URL. For example, `https://b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-east-2.amazonaws.com`.

  For more information, see RabbitMQ web console in the RabbitMQ documentation.
- A secure AMQP endpoint. For example, `amqps://b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-east-2.amazonaws.com`.

For a full list of broker attributes, see the following in the *Amazon MQ REST API Reference*:

- REST Operation ID: Broker
- REST Operation ID: Brokers
- REST Operation ID: Broker Reboot

Managing Amazon MQ for RabbitMQ engine versions

RabbitMQ organizes version numbers according to semantic versioning specification as `X.Y.Z`. In Amazon MQ for RabbitMQ implementations, `X` denotes the major version, `Y` represents the minor version, and `Z` denotes the patch version number. Amazon MQ considers a version change to be major if the major version numbers change. For example, upgrading from version **3**.13 to **4**.0 is considered a major version upgrade. A version change is considered minor if only the minor or patch version number changes. For example, upgrading from version 3.**11**.28 to 3.**12**.13 is considered a minor version upgrade.

Amazon MQ for RabbitMQ recommends all brokers use the latest supported version RabbitMQ 4.2. For instructions on how to upgrade your broker engine version, see Upgrading an Amazon MQ broker engine version.

When you create a new Amazon MQ for RabbitMQ broker, you only need to specify the major and minor version numbers. For example, RabbitMQ 4.2. If you do not specify the engine version when creating a broker, Amazon MQ automatically defaults to the latest engine version.

> **Important**
>
> Amazon MQ does not support streams. Creating a stream will result in data loss.
> Amazon MQ does not support using structured logging in JSON.

Amazon MQ supports two major version releases of RabbitMQ:

- **RabbitMQ 4**

  Amazon MQ supports RabbitMQ 4.2 in the RabbitMQ 4 release series only on the mq.m7g instance type across all supported instance sizes.

- **RabbitMQ 3**

  Amazon MQ supports RabbitMQ 3.13 in the RabbitMQ 3 release series on mq.t3, mq.m5, and mq.m7g instance types across all supported instance sizes.

Listing supported engine versions

You can list all supported minor and major engine versions by using the `describe-broker-instance-options` AWS CLI command.

```
aws mq describe-broker-instance-options
```

To filter the results by engine and instance type use the `--engine-type` and `--host-instance-type` options as shown in the following.

```
aws mq describe-broker-instance-options --engine-type engine-type --host-instance-
type instance-type
```

For example, to filter the results for RabbitMQ, and `mq.m7g.large` instance type, replace *`engine-type`* with `RABBITMQ` and *`instance-type`* with `mq.m7g.large`.

RabbitMQ 4

Amazon MQ supports RabbitMQ 4.2 in the RabbitMQ 4 release series only on the mq.m7g instance type across all supported instance sizes.

> **Important**
>
> You can only create new brokers on RabbitMQ 4.2. In place upgrades from RabbitMQ 3.13 are not currently supported.

> **Important**
>
> The default queue type on Amazon MQ for RabbitMQ 4.2 brokers will be “quorum”. If no queue type argument is specified during queue creation, a quorum queue will be created. We highly recommend using quorum queues on RabbitMQ 4 for durability needs, since classic queues are not guaranteed to be durable in all cases.

The following changes have been introduced in RabbitMQ 4 on Amazon MQ

- **AMQP 1.0 as a core protocol:** For more information, see Protocols.
- **Local shovels:** Shovels now support a new protocol called "local" in addition to AMQP 0-9-1 and AMQP 1.0. Local shovels are internally based on AMQP 1.0 but instead of using separate TCP connections, they use intra-cluster connections between cluster nodes and internal APIs for publishing and consuming messages. This can only be used for consuming and publishing within the same cluster and can offer higher throughput while using fewer resources than AMQP 0-9-1 and AMQP 1.0.
- **Quorum queues support message priorities:** Quorum queue message priorities are always active and do not require a policy to work. As soon as a quorum queue receives a message with a priority set it will enable prioritization. Quorum queues internally only support two priorities - high and normal. Messages without a priority set will be mapped to normal as will priorities 0 - 4. Messages with a priority higher than 4 will be mapped to high. High priority messages will be favoured over normal priority messages at a ratio of 2:1, i.e. for every 2 high priority message the queue will deliver 1 normal priority message (if available). Hence, quorum queues implement

a kind of non-strict, "fair share" priority processing. This ensures progress is always made on normal priority messages, but high priorities are favoured at a ratio of 2:1.

- **Khepri:** Khepri is used as the default metadata store for RabbitMQ 4 brokers
- **Mutual TLS (mTLS):** Amazon MQ supports mutual TLS (mTLS) for RabbitMQ brokers, allowing clients to authenticate using certificates. For more information, see mTLS configuration.
- **SSL certificate authentication plugin:** The SSL authentication plugin uses client certificates from mTLS connections to authenticate users, allowing authentication using X.509 client certificates instead of username and password credentials. For more information, see SSL certificate authentication.
- **HTTP authentication plugin:** The HTTP authentication backend plugin allows delegating authentication and authorization to an external HTTP service. For more information, see HTTP authentication and authorization.
- **JMS support:** The broker now supports JMS workloads with the JMS topic exchange plugin enabled, allowing JMS applications to connect using the RabbitMQ JMS client.

The following features have been deprecated from RabbitMQ 4 on Amazon MQ

- **Mirroring of classic queues:** Classic queues continue to be supported without any breaking changes for client libraries and applications, but they are now a non-replicated queue type. Clients will be able to connect to any node to publish to and consume from any non-replicated classic queues. Quorum queues are recommended for replication and data safety.
- **Removal of Global QoS:** Customers are recommended to set per-consumer QoS (non-global) instead of Global QoS, where a single shared prefetch is used for an entire channel.
- **Support for transient, non-exclusive queues:** Transient queues are queues whose lifetime is linked to the uptime of the node they are declared on. In a single instance broker, they are removed when the node is restarted. In a cluster deployment, they are removed when the node they are hosted on is restarted. We recommend using queue TTL for auto-deleting unused, idle queues after some time of inactivity. Exclusive queues continue to be supported and are deleted once all connections to the queue have been removed.

The following breaking changes may impact your applications when upgrading to RabbitMQ 4.2 on Amazon MQ

- **Default queue type:** The default queue type on a RabbitMQ 4 broker is set to quorum. If no queue type argument is specified during queue creation, a quorum queue will be created.

- **Default redelivery limit on quorum queues is set to 20:** Messages that are redelivered 20 times or more will be dead-lettered or dropped (removed). If 20 deliveries per message is a common scenario for a queue, a dead-lettering target or a higher limit must be configured for such queues to avoid data loss. The recommended way of doing that is via a policy.
- **amqplib:** Node JS client **amqplib versions older than 0.10.7** or any AMQP client library using **frame_max < 8192** will not be able to connect to RabbitMQ
- Default resource limits: Amazon MQ for RabbitMQ has introduced default resource usage limits for connections, channels, consumers per channel, queues, vhosts, shovels, exchanges, and maximum message size. These serve as guardrails to protect broker availability and can be customized using configurations to match your specific requirements.

The following features are not supported on RabbitMQ 4 on Amazon MQ

- **Local Random exchanges:** Local random exchanges are not supported on Amazon MQ since the Amazon MQ nodes are behind a network load balancer.
- **Message Interceptor:** RabbitMQ message interceptors are not supported on Amazon MQ.
- **Per queue metrics:** Amazon MQ will not vend RabbitMQ queue metrics for RabbitMQ 4 brokers through AWS CloudWatch. Amazon MQ will still provide broker level metrics through AWS CloudWatch. You can query queue metrics using the RabbitMQ management API. We recommend querying metrics for specific queues at a frequency of one minute or longer intervals.

RabbitMQ version support

The Amazon MQ version support calendar below indicates when a broker engine version will reach end of support. When a version reaches end of support, Amazon MQ upgrades all brokers on this version to the next supported version automatically. This upgrade takes place during your broker's scheduled maintenance windows, within 45 days following the end-of-support date.

Amazon MQ provides at least a 90 day notice before a version reaches end of support. We recommend upgrading your broker before the end-of-support date to prevent any disruptions. Additionally, you cannot create new brokers on versions scheduled for end of support within 30 days of the end of support date.

RabbitMQ version	End of support on Amazon MQ
4.2 (Recommended)	
3.13	
3.12	March 17, 2025

Version upgrades

You can manually upgrade your broker at any time to the next supported major or minor version. For more information about manually upgrading your broker, see Upgrading an Amazon MQ broker engine version.

Amazon MQ manages upgrades to the latest supported patch version for all RabbitMQ brokers using version 3.13 and above. Both manual and automatic version upgrades occur during the scheduled maintenance window or after you reboot your broker.

> **Important**
>
> RabbitMQ only allows incremental version updates (ex: 3.9.x to 3.10.x). You cannot skip minor versions when updating (ex: 3.8.x to 3.11.x).

Single instance brokers will be offline while being rebooted. For cluster brokers, the mirrored queues must be synced during reboot. With longer queues, the queue-sync process can take longer. During the queue-sync process, the queue is unavailable to consumers and producer. When the queue-sync process is complete, the broker becomes available again. To minimize the impact, we recommend upgrading during a low traffic time. For more information on best practices for version upgrades, see Amazon MQ for RabbitMQ best practices.

Deployment options for Amazon MQ for RabbitMQ brokers

RabbitMQ brokers can be created as *single-instance brokers* or in a *cluster deployment*. For both deployment modes, Amazon MQ provides high durability by storing its data redundantly.

You can access your RabbitMQ brokers by using any programming language that RabbitMQ supports and by enabling TLS for the following protocols:

- AMQP (0-9-1)

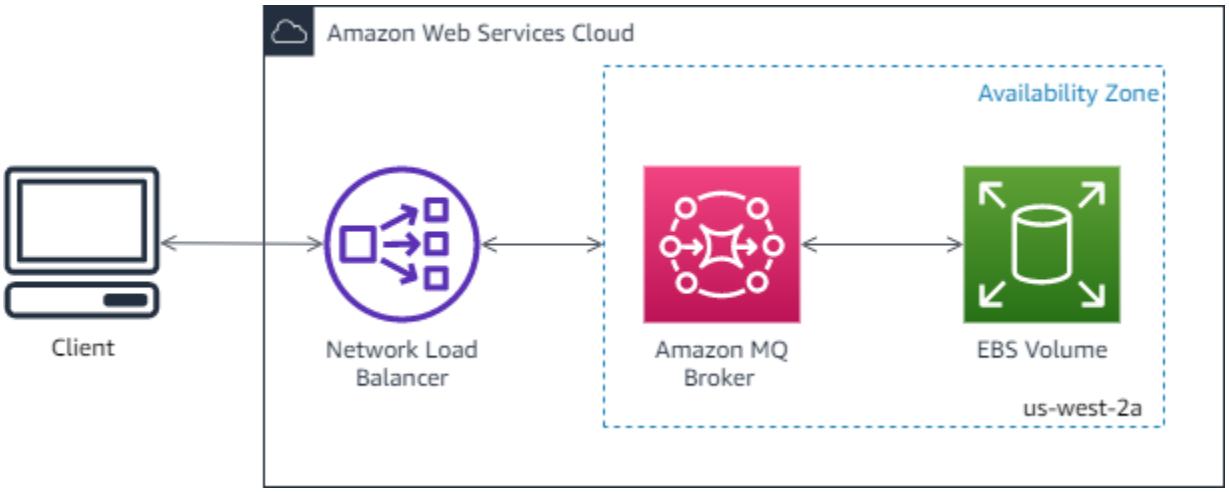
Topics

Option 1: Amazon MQ for RabbitMQ single-instance broker

A *single-instance broker* is comprised of one broker in one Availability Zone behind a Network Load Balancer (NLB). The broker communicates with your application and with an Amazon EBS storage volume. Amazon EBS provides block level storage optimized for low-latency and high throughput.

Using an Network Load Balancer ensures that your Amazon MQ for RabbitMQ broker endpoint remains unchanged if the broker instance is replaced during a maintenance window or because of underlying Amazon EC2 hardware failures. An Network Load Balancer allows your applications and users to continue to use the same endpoint to connect to the broker.

The following diagram illustrates an Amazon MQ for RabbitMQ single-instance broker.



Option 2: Amazon MQ for RabbitMQ cluster deployment

A *cluster deployment* is a logical grouping of three RabbitMQ broker nodes behind a Network Load Balancer, each sharing users, queues, and a distributed state across multiple Availability Zones (AZ).

In a cluster deployment, Amazon MQ automatically manages broker policies to enable classic mirroring across all nodes, ensuring high availability (HA). Each mirrored queue consists of one *main* node and one or more *mirrors*. Each queue has its own main node. All operations for a given

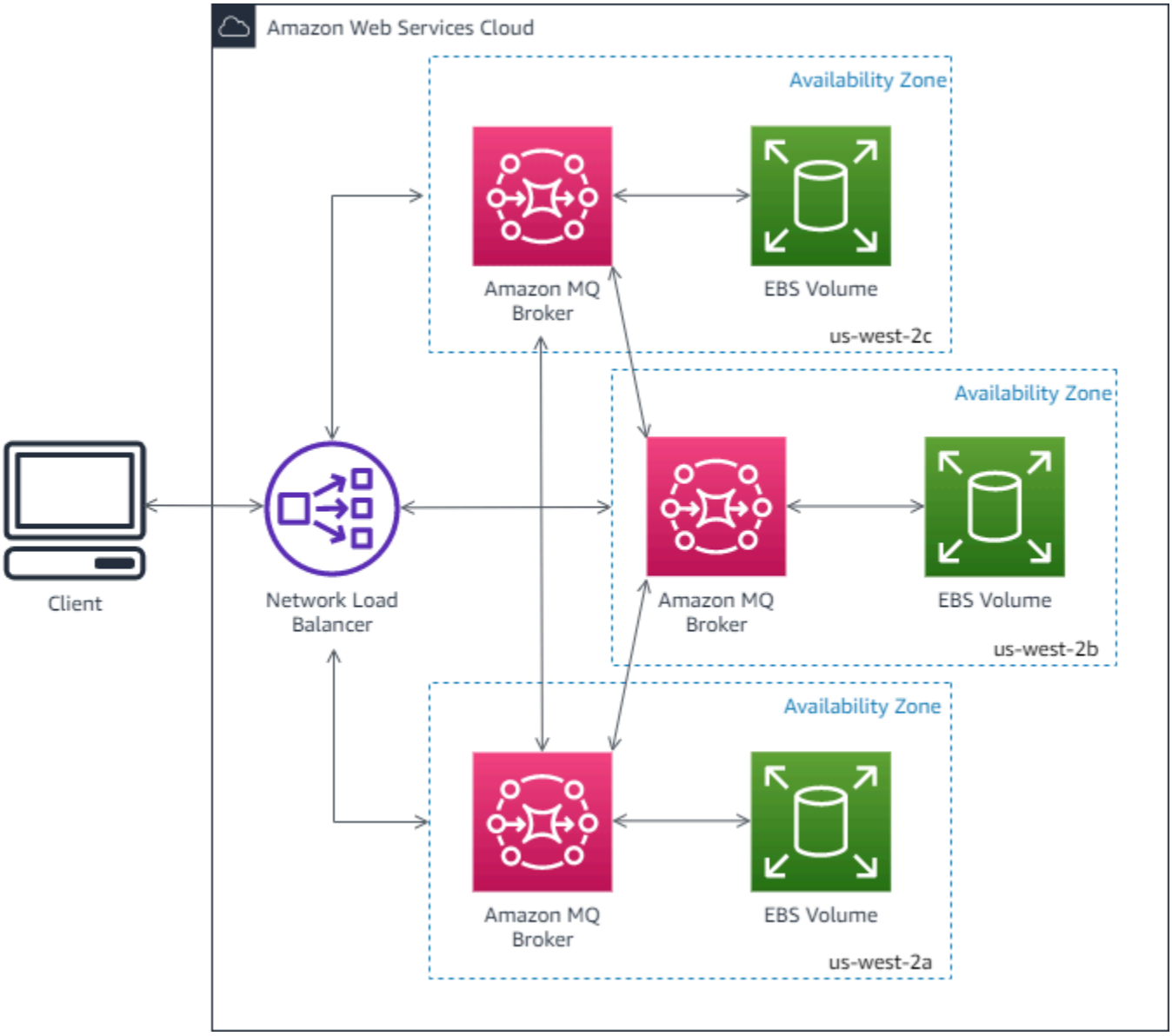
queue are first applied on the queue's main node and then propagated to mirrors. Amazon MQ creates a default system policy that sets the `ha-mode` to `all` and `ha-sync-mode` to `automatic`. This ensures that data is replicated to all nodes in the cluster across different Availability Zones for better durability.

> **Note**
>
> In a cluster deployment, if an Availability Zone outage occurs, Amazon MQ will automatically attempt to relocate the affected RabbitMQ nodes to a different AZ to maintain the cluster size. Once the outage resolves, the cluster will be automatically rebalanced across the AZs.

> **Note**
>
> During a *maintenance window*, all maintenance to a cluster is performed one node at a time, keeping at least two running nodes at all times. Each time a node is brought down, client connections to that node are severed and need to be re-established. You must ensure that your client code is designed to automatically reconnect to your cluster. For more information about connection recovery, see the section called “Step 1: Automatically recover from network failures”.
>
> Because Amazon MQ sets `ha-sync-mode: automatic`, during a maintenance window, queues will synchronize when each node re-joins the cluster. Queue synchronization blocks all other queue operations. You can mitigate the impact of queue synchronization during maintenance windows by keeping queues short.

The default policy should not be deleted. If you do delete this policy, Amazon MQ will automatically recreate it. Amazon MQ will also ensure that HA properties are applied to all other policies that you create on a clustered broker. If you add a policy without the HA properties, Amazon MQ will add them for you. If you add a policy with different high availability properties, Amazon MQ will replace them. For more information about classic mirroring, see Classic mirrored queues.

The following diagram illustrates a RabbitMQ cluster broker deployment with three nodes in three Availability Zones (AZ), each with its own Amazon EBS volume and a shared state. Amazon EBS provides block level storage optimized for low-latency and high throughput.



Amazon MQ for RabbitMQ broker instance types

The combined description of the broker instance class (m7g) and size (large, medium) is called the broker instance type (for example, mq.m7g.large).

We recommend using mq.m7g instance types for both cluster and single-instance deployments.

Amazon MQ provides at least a 90 day notice before an instance type reaches end of support. We recommend upgrading your broker to a new instance type before the end-of-support date to prevent any disruptions.

Important

You cannot downgrade a broker from an `mq.m7g` or `mq.m5` instance type to a `mq.t3.micro` instance type.
The `mq.t3.micro` instance type does not support cluster deployment.

Instance types for m7g cluster deployment

We recommending using `mq.m7g.x` instance types with cluster deployment. The following table shows the available `mq.m7g.x` instance types for cluster deployment.

Instance Type	vCPU	Memory (GiB)	Network Baseline / Burst bandwidth (Gbps)	Recommended use	Storage	Disk volume size per node(GB)
mq.m7g.medium	1	4	0.52 / 12.5	Evaluation	EBS	5
mq.m7g.large	2	8	0.937 / 12.5	Production	EBS	15
mq.m7g.xlarge	4	16	1.876 / 12.5	Production	EBS	25
mq.m7g.2xlarge	8	32	3.75 / 15.0	Production	EBS	45
mq.m7g.4xlarge	16	64	7.5 / 15.0	Production	EBS	90
mq.m7g.8xlarge	32	128	15 Gigabit	Production	EBS	175
mq.m7g.12xlarge	48	192	22.5 Gigabit	Production	EBS	260

Instance Type	vCPU	Memory (GiB)	Network Baseline / Burst bandwidth (Gbps)	Recommended use	Storage	Disk volume size per node(GB)
mq.m7g.16xlarge	64	256	30 Gigabit	Production	EBS	345

Instance types for m7g single instance deployment

The following table shows the available `mq.m7g.x` instance types for single instance deployment.

Instance Type	vCPU	Memory (GiB)	Network Baseline / Burst bandwidth (Gbps)	Recommended use	Storage	Disk volume size per node(GB)
mq.m7g.medium	1	4	0.52 / 12.5	Evaluation	EBS	200
mq.m7g.large	2	8	0.937 / 12.5	Production	EBS	200
mq.m7g.xlarge	4	16	1.876 / 12.5	Production	EBS	200
mq.m7g.2xlarge	8	32	3.75 / 15.0	Production	EBS	200
mq.m7g.4xlarge	16	64	7.5 / 15.0	Production	EBS	200
mq.m7g.8xlarge	32	128	15 Gigabit	Production	EBS	200

Instance Type	vCPU	Memory (GiB)	Network Baseline / Burst bandwidth (Gbps)	Recommended use	Storage	Disk volume size per node(GB)
mq.m7g.12xlarge	48	192	22.5 Gigabit	Production	EBS	200
mq.m7g.16xlarge	64	256	39 Gigabit	Production	EBS	200

Instance types for `mq.m5` single instance deployment

The following tables show the available `mq.m5.x` instance types for single instance deployment

Instance Type	vCPU	Memory (GiB)	Network Baseline / Burst bandwidth (Gbps)	Recommended use	Storage	Disk volume size per node(GB)
mq.t3.micro	2	1	0.064 / 5.0	Evaluation	EBS	20
mq.m5.large	2	8	0.75 / 10.0	Production	EBS	200
mq.m5.xlarge	4	16	1.25 / 10.0	Production	EBS	200
mq.m5.2xlarge	8	32	2.5 / 10.0	Production	EBS	200
mq.m5.4xlarge	16	64	5.0 / 10.0	Production	EBS	200

Instance types for `mq.m5` cluster deployment

The following tables show the available `mq.m5.x` instance types for cluster deployment

Instance Type	vCPU	Memory (GiB)	Network Baseline / Burst bandwidth (Gbps)	Recommended use	Storage	Disk volume size per node(GB)
mq.m5.large	2	8	0.75 / 10.0	Production	EBS	200
mq.m5.xlarge	4	16	1.25 / 10.0	Production	EBS	200
mq.m5.2xlarge	8	32	2.5 / 10.0	Production	EBS	200
mq.m5.4xlarge	16	64	5.0 / 10.0	Production	EBS	200

Amazon MQ for RabbitMQ sizing guidelines

You can choose the broker instance type that best supports your application. When choosing an instance type, consider factors that will affect broker performance:

- the number of clients and queues
- the volume of messages sent
- messages kept in memory
- redundant messages

Smaller broker instance types `m7g.medium` are recommended only for testing application performance. We recommend larger broker instance types `m7g.large` and above or production levels of clients and queues, high throughput, messages in memory, and redundant messages.

Important

You cannot downgrade a broker from an `mq.m5` or `mq.m7g` instance type to an `mq.t3.micro` instance type.

It is important to test your brokers to determine the appropriate instance type and size for your workload messaging requirements.

Always use the default resource limits on RabbitMQ 4 broker to determine the appropriate instance size for your application according to Amazon MQ best practices. These default resource limits are based on types `m7g` instance type and quorum queues.

- Default resource limits for m7g single-instance deployment
- Default resource limits for m7g cluster deployment

You can increase the value of any limit up to the maximum values as defined by instance type and deployment mode. However, we strongly recommend you test the broker performance with the increased values before using in production.

- Maximum resource limits for m7g single-instance deployment
- Maximum resource limits for m7g cluster deployment
- Maximum resource limits for m5 single-instance deployment
- Maximum resource limits for m5 cluster deployment
- Error messages

Note

RabbitMQ 3.13 brokers do not come with default resource limits, but we recommend you use the suggested defaults.

Default resource limits

Amazon MQ for RabbitMQ supports configuring the broker resource limits from RabbitMQ 4 onwards. When you create a broker, Amazon MQ automatically applies default values to

these resource limits. These defaults act as guardrails to protect your broker availability while accommodating common customer usage patterns. You can customize your broker behavior by changing the limit configuration values to better match your specific workload requirements.

Before making changes, please note:

Important

1. Configuration changes can impact broker performance and availability
2. Understand the impact before changing any default configuration options
3. Test configuration changes in non-production environments first
4. Monitor broker health after applying changes

Important

Default values and supported ranges for these configurations vary by RabbitMQ version, instance type, and broker deployment mode.

Important

Note: Associating or creating a broker with configuration values outside the supported range will result in an error response.

To learn how to customize these default resource limits for your broker, see the section called "Configuring Resource Limit".

The default resource limits applied for RabbitMQ 4.2 brokers are

- Default resource limits for m7g single-instance deployment
- Default resource limits for m7g cluster deployment

Default resource limits

> **Important**
>
> Amazon MQ for RabbitMQ 3 brokers, the default is configured with the maximum resource limit and Amazon MQ does not provide the ability to override resource limit configuration.

Default values for single instance brokers

Instance type	Connections per Node	Channels per Node	Consumer per channel	Queues	vhosts	Shovels	Exchange	Message size in Bytes
mq.m7g.medium	100	500	10	500	10	30	500	524288
mq.m7g.large	1,500	4,500	10	1,000	50	50	1,000	524288
mq.m7g.xlarge	3,000	9,000	10	2,000	100	100	2,000	524288
mq.m7g.2xlarge	6,000	18,000	10	4,000	150	200	4,000	524288
mq.m7g.4xlarge	12,000	36,000	10	8,000	200	400	8,000	524288
mq.m7g.8xlarge	24,000	72,000	10	16,000	250	800	16,000	524288
mq.m7g.12xlarge	36,000	108,000	10	24,000	300	1,200	24,000	524288
mq.m7g.16xlarge	48,000	144,000	10	32,000	350	1,600	32,000	524288

Default values for cluster brokers

Instance type	Connections per Node	Channels per Node	Consumers per channel	Queues	vhosts	Shovels	Exchanges	Message size in Bytes
mq.m7g.medium	100	300	10	100	10	10	100	524288
mq.m7g.large	500	1500	10	1,000	50	30	1,000	524288
mq.m7g.xlarge	1000	3000	10	2,000	100	60	2,000	524288
mq.m7g.2xlarge	2000	6000	10	4,000	150	120	4,000	524288
mq.m7g.4xlarge	4000	12,000	10	8,000	200	240	8,000	524288
mq.m7g.8xlarge	8000	24,000	10	16,000	250	480	16,000	524288
mq.m7g.12xlarge	12000	36,000	10	24,000	300	720	24000	524288
mq.m7g.16xlarge	16,000	48,000	10	32,000	350	960	32,000	524288

Amazon MQ for RabbitMQ maximum resource limit

You can configure resource limits up to the maximum values shown in the following tables. To learn how to update resource limits for your broker, see the section called “Configuring Resource Limit”.

Sizing guidelines for m7g with quorum queues for single instance deployment

The following table shows the **maximum** limit values for each instance type for single instance brokers.

Instance Type	Connections	Channels	Consumers per channel	Queues	Vhosts	Shovels	Exchanges	Message Size in Bytes
mq.m7g.medium	300	900	1,000	2,500	10	150	12500	134217728
mq.m7g.large	5,000	15,000	1,000	20,000	1500	250	100,000	134217728
mq.m7g.xlarge	10,000	30,000	1,000	30,000	1,500	500	150,000	134217728
mq.m7g.2xlarge	20,000	60,000	1,000	40,000	1,500	1,000	200,000	134217728
mq.m7g.4xlarge	40,000	120,000	1,000	60,000	1,500	2000	300,000	134217728
mq.m7g.8xlarge	80,000	240,000	1,000	80,000	1,500	4000	400,000	134217728
mq.m7g.12xlarge	120,000	360,000	1,000	100,000	1,500	6,000	500,000	134217728
mq.m7g.16xlarge	160,000	480,000	1,000	120,000	1,500	8,000	600,000	134217728

Sizing guidelines for m7g with quorum queues for cluster deployment

The following table shows the **maximum** limit values for each instance type for cluster brokers.

Instance Type	Connections per Node	Channels per Node	Consumers per channel	Queues	Vhosts	Shovels	Exchanges	Message Size in Bytes
mq.m7g.medium	300	900	1,000	500	10	50	500	134217728
mq.m7g.large	5,000	15,000	1,000	10,000	1,500	150	50,000	134217728
mq.m7g.xlarge	10,000	30,000	1,000	15,000	1,500	300	75,000	134217728
mq.m7g.2large	20,000	60,000	1,000	20,000	1,500	600	100,000	134217728
mq.m7g.4large	40,000	120,000	1,000	30,000	1,500	1200	150,000	134217728
mq.m7g.8large	80,000	240,000	1,000	40,000	1,500	2,400	200,000	134217728
mq.m7g.12xlarge	120,000	360,000	1,000	50,000	1,500	3,600	250,000	134217728
mq.m7g.16xlarge	160,000	480,000	1,000	60,000	1,500	4,800	300,000	134217728

Maximum resource limits for M5 single-instance deployment

The following table shows the **maximum** limit values for each instance type for single instance brokers.

Instance Type	Connections	Channels	Consumers per channel	Queues	Vhosts	Shovels
m5.large	5,000	15,000	1,000	30,000	1500	250

Instance Type	Connections	Channels	Consumers per channel	Queues	Vhosts	Shovels
m5.xlarge	10,000	30,000	1,000	60,000	1500	500
m5.2xlarge	20,000	60,000	1,000	120,000	1500	1,000
m5.4xlarge	40,000	120,000	1000	240,000	1,000	2,000

Maximum resource limits for m5 cluster deployment

The following table shows the **maximum** limit values for each instance type for cluster brokers.

Instance Type	Queues	Consumers per channel	Shovels
m5.large	10,000	1,000	150
m5.xlarge	15,000	1,000	300
m5.2xlarge	20,000	1,000	600
m5.4xlarge	30,000	1,000	1200

The following connection and channel limits are applied per node:

Instance Type	Connections	Channels
m5.large	5000	15,000
m5.xlarge	10,000	30,000
m5.2xlarge	20,000	60,000
m5.4xlarge	40,000	120,000

The exact limit values for a cluster broker may be lower than the indicated value depending on the number of available nodes and how RabbitMQ distributes resources among the available nodes. If you exceed the limit values, you can create a new connection to a different node and try again, or you can upgrade the instance size to increase the maximum limits

Error messages

The following error messages are returned when limits are exceeded. All values are based on the **`m7.large`** single instance limits.

> **Note**
>
> The error codes for the following messages may change based on the client library you are using.

Connection

```
ConnectionClosedByBroker 500 "NOT_ALLOWED - connection refused: node
connection limit (5000) is reached"
```

Channel

```
ConnectionClosedByBroker 1500 "NOT_ALLOWED - number of channels opened on
node 'rabbit@ip-10-0-23-173.us-west-2.compute.internal' has reached the
maximum allowed limit of (15,000)"
```

Consumer

```
ConnectionClosedByBroker: (530, 'NOT_ALLOWED - reached maximum (1,000) of
consumers per channel')
```

Maximum message size

```
(406, 'PRECONDITION_FAILED - message size 524289 is larger than configured
max size 524288')
```

Exchange

```
(406, "PRECONDITION_FAILED - cannot declare exchange 'limit_test_3' in
vhost '/': exchange limit of 10 is reached")
```

> **Note**
>
> The following error messages use the HTTP Management API format.

Queue

```
{"error":"bad_request","reason":"cannot declare queue 'my_queue': queue
limit in cluster (10,000) is reached"}]
```

Shovel

```
{"error":"bad_request","reason":"Validation failed\n\ncomponent shovel is
limited to 150 per node\n"}
```

Vhost

```
{"error":"bad_request","reason":"cannot create vhost 'my_vhost': vhost
limit of 1500 is reached"}
```

Amazon MQ for RabbitMQ broker defaults

When you create an Amazon MQ for RabbitMQ broker, Amazon MQ applies a default set of broker policies and vhost limits to optimize your broker's performance. Amazon MQ applies vhost limits only to the default (/) vhost. Amazon MQ will not apply default policies to newly created vhosts. We recommend keeping these defaults for all new and existing brokers. However, you can modify, override, or delete these defaults at any time.

Amazon MQ creates different broker policies and vhost limits for Amazon MQ for RabbitMQ 3 and RabbitMQ 4. The differences will be discussed in detail in the following subsections.

Amazon MQ creates policies and limits based on the instance type and broker deployment mode that you choose when you create your broker. The default policies are named according to the deployment mode, as follows:

Amazon MQ for RabbitMQ 3:

- **Single-instance** – `AWS-DEFAULT-POLICY-SINGLE-INSTANCE`
- **Cluster deployment** – `AWS-DEFAULT-POLICY-CLUSTER-MULTI-AZ` && `AWS-DEFAULT-QUORUM-QUEUES-POLICY-CLUSTER-MULTI-AZ`

Amazon MQ for RabbitMQ 4:

- **Single-instance** – `AWS-DEFAULT-POLICY-SINGLE-INSTANCE`
- **Cluster deployment** – `AWS-DEFAULT-POLICY-CLUSTER && AWS-DEFAULT-QUORUM-QUEUES-POLICY-CLUSTER-MULTI-AZ`

For single-instance brokers, Amazon MQ sets the policy priority value to `0`. To override the default priority value, you can create your own custom policies with higher priority values. For cluster deployments, Amazon MQ sets the priority value to 1 for broker defaults. To create your own custom policy for clusters, assign a priority value greater than 1.

> **Note**
>
> In cluster deployments, `ha-mode` and `ha-sync-mode` broker policies are required for classic mirroring and high availability (HA). These settings are only applicable for Amazon MQ for RabbitMQ 3 and are not configured for RabbitMQ 4.
> If you delete the default `AWS-DEFAULT-POLICY-CLUSTER-MULTI-AZ` policy, Amazon MQ uses the `ha-all-AWS-OWNED-DO-NOT-DELETE` policy with a priority value of `0`. This ensures that the required `ha-mode` and `ha-sync-mode` policies are still in effect. If you create your own custom policy, Amazon MQ automatically appends `ha-mode` and `ha-sync-mode` to your policy definitions.

Topics

Policy and limit descriptions

The following list describes the default policies and limits that Amazon MQ applies to a newly created broker. The values for `max-length`, `max-queues`, and `max-connections` vary based on your broker's instance type and deployment mode. These values are listed in the Recommended default values section.

Settings on both RabbitMQ 3 and RabbitMQ 4 brokers

- **`queue-mode: lazy`** (policy) – Enables lazy queues. By default, queues keep an in-memory cache of messages, enabling the broker to deliver messages to consumers as fast as possible. This can lead to the broker running out of memory and raising a high-memory alarm. Lazy queues attempt to move messages to disk as early as is practical. This means that fewer messages are kept in memory under normal operating conditions. Using lazy queues, Amazon MQ for RabbitMQ can support much larger messaging loads and longer queues. Note that for certain use cases, brokers with lazy queues might perform marginally slower. This is because messages are moved from disk to broker, as opposed to delivering messages from an in-memory cache.

  Deployment modes

  Single-instance, cluster

- **`max-length: number-of-messages`** (policy) – Sets a limit for the number of messages in a queue. In cluster deployments, the limit prevents paused queue synchronization in cases such as broker reboots, or following a maintenance window.

  Deployment modes

  Cluster

- **`overflow: reject-publish`** (policy) – Enforces queues with a `max-length` policy to reject new messages after the number of messages in the queue reaches the `max-length` value. To ensure that messages aren't lost if a queue is in an overflow state, client applications that publish messages to the broker must implement publisher confirms. For information about implementing publisher confirms, see Publisher Confirms on the RabbitMQ website.

  Deployment modes

  Cluster

Settings specific to RabbitMQ 3

- **`max-queues: number-of-queues-per-vhost`** (vhost limit) – Sets the limit for the number of queues in a broker. Similar to the `max-length` policy definition, limiting the number of queues in cluster deployments prevents paused queue synchronization following broker reboots

or maintenance windows. Limiting queues also prevents excessive amounts of CPU usage for maintaining queues.

> **Deployment modes**
>
> Single-instance, cluster

- **`max-connections: number-of-connections-per-vhost`** (vhost limit) – Sets the limit for the number of client connections to the broker. Limiting the number of connections according to the recommended values prevents excessive broker memory usage, which could result in the broker raising a high memory alarm and pausing operations.

> **Deployment modes**
>
> Single-instance, cluster

Recommended default values

> **Important**
>
> `max-queues` and `max-connections` are only applied to Amazon MQ for RabbitMQ 3.

> **Note**
>
> The `max-length` and `max-queue` default limits are tested and evaluated based on an average message size of 5 kB. If your messages are significantly larger than 5 kB, you will need to adjust and reduce the `max-length` and `max-queue` limits.

The following table lists the default limit values for a newly created broker. Amazon MQ applies these values according to the broker's instance type and deployment mode.

Instance type	Deployment mode	max-length	max-queues	max-connections
mq.m7g.medium	Single-instance	N/A	2,500	100
	Cluster	500,000	100	100
mq.m7g.large	Single-instance	N/A	20,000	5,000
	Cluster	8,000,000	10,000	5,000
mq.m7g.xlarge	Single-instance	N/A	30,000	10,000
	Cluster	9,000,000	15,000	10,000
mq.m7g.2xlarge	Single-instance	N/A	40,000	20,000
	Cluster	10,000,000	40,000	20,000
mq.m7g.4xlarge	Single-instance	N/A	60,000	40,000
	Cluster	12,000,000	30,000	40,000
mq.m7g.8xlarge	Single-instance	N/A	80,000	80,000
	Cluster	20,000,000	40,000	80,000
mq.m7g.12xlarge	Single-instance	N/A	100,000	120,000
	Cluster	30,000,000	20,000	120,000
mq.m7g.16xlarge	Single-instance	N/A	120,000	160,000
	Cluster	40,000,000	50,000	160,000

Instance type	Deployment mode	max-length	max-queues	max-connections
t3.micro	Single-instance	N/A	500	500

Instance type	Deployment mode	max-length	max-queues	max-connections
m5.large	Single-instance	N/A	20,000	4,000
m5.large	Cluster	8,000,000	10,000	15,000
m5.xlarge	Single-instance	N/A	30,000	8,000
m5.xlarge	Cluster	9,000,000	10,000	20,000
m5.2xlarge	Single-instance	N/A	60,000	15,000
m5.2xlarge	Cluster	10,000,000	10,000	40,000
m5.4xlarge	Single-instance	N/A	150,000	30,000
m5.4xlarge	Cluster	12,000,000	10,000	100,000

Configuring a RabbitMQ broker

A configuration contains all the settings for your RabbitMQ broker in Cuttlefish format. You can create a configuration before creating any brokers. You can then apply the configuration to one or more brokers.

Attributes

A broker configuration has several attributes, for example:

- A name (MyConfiguration)
- An ID (c-1234a5b6-78cd-901e-2fgh-3i45j6k178l9)
- An Amazon Resource Name (ARN) (arn:aws:mq:us-east-2:123456789012:configuration:c-1234a5b678cd-901e-2fgh-3i45j6k178l9)

For a full list of configuration attributes, see the following in the Amazon MQ REST API Reference:

- REST Operation ID: Configuration
- REST Operation ID: Configurations

For a full list of configuration revision attributes, see the following:

- REST Operation ID: Configuration Revision
- REST Operation ID: Configuration Revisions

Topics

Creating and applying RabbitMQ broker configurations

A *configuration* contains all of the settings for your RabbitMQ broker in Cuttlefish format. You can create a configuration before creating any brokers. You can then apply the configuration to one or more brokers

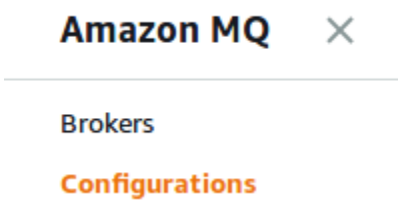
The following examples show how you can create and apply a RabbitMQ broker configuration using the AWS Management Console.

> **Important**
>
> You can only **delete** a configuration using the `DeleteConfiguration` API. For more information, see Configurations in the *Amazon MQ API Reference*.

Create a New Configuration

To apply a configuration to your broker, you must first create the configuration.

1. Sign in to the Amazon MQ console.
2. On the left, expand the navigation panel and choose **Configurations**.



3. On the **Configurations** page, choose **Create configuration**.
4. On the **Create configuration** page, in the **Details** section, type the **Configuration name** (for example, `MyConfiguration`) and select a **Broker engine** version.

   To learn more about RabbitMQ engine versions supported by Amazon MQ for RabbitMQ, see the section called “Version management”.
5. Choose **Create configuration**.

Create a New Configuration Revision

After you create a configuration, you must edit the configuration using a configuration revision.

1. From the configuration list, choose ***`MyConfiguration`***.

   > **Note**
   >
   > The first configuration revision is always created for you when Amazon MQ creates the configuration.

   On the ***`MyConfiguration`*** page, the broker engine type and version that your new configuration revision uses (for example, **RabbitMQ 3.xx.xx**) are displayed.
2. On the **Configuration details** tab, the configuration revision number, description, and broker configuration in Cuttlefish format are displayed.

   > **Note**
   >
   > Editing the current configuration creates a new configuration revision.
3. Choose **Edit configuration** and make changes to the Cuttlefish configuration.
4. Choose **Save**.

The **Save revision** dialog box is displayed.

5. (Optional) Type `A description of the changes in this revision`.
6. Choose **Save**.

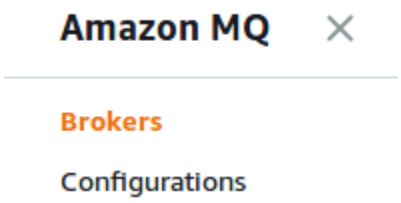
   The new revision of the configuration is saved.

> **Important**
>
> Making changes to a configuration does *not* apply the changes to the broker immediately. To apply your changes, you must wait for the next maintenance window or reboot the broker.
> Currently, you can't delete a configuration.

Apply a Configuration Revision to Your Broker

After creating the configuration revision, you can apply the configuration revision to your broker.

1. On the left, expand the navigation panel and choose **Brokers**.



2. From the broker list, select your broker (for example, **MyBroker**) and then choose **Edit**.
3. On the **Edit *`MyBroker`*** page, in the **Configuration** section, select a **Configuration** and a **Revision** and then choose **Schedule Modifications**.
4. In the **Schedule broker modifications** section, choose whether to apply modifications **During the next scheduled maintenance window** or **Immediately**.

> **Important**
>
> Single instance brokers are offline while being rebooted. For cluster brokers, only one node is down at a time while the broker reboots.

5. Choose **Apply**.

Your configuration revision is applied to your broker at the specified time.

Edit a Amazon MQ for RabbitMQ Configuration Revision

The following instructions describe how to edit a configuration revision for your broker.

1. Sign in to the Amazon MQ console.
2. From the broker list, select your broker (for example, **MyBroker**) and then choose **Edit**.
3. On the ***MyBroker*** page, choose **Edit**.
4. On the **Edit *MyBroker*** page, in the **Configuration** section, select a **Configuration** and a **Revision** and then choose **Edit**.

   > **Note**
   >
   > Unless you select a configuration when you create a broker, the first configuration revision is always created for you when Amazon MQ creates the broker.

   On the ***MyBroker*** page, the broker engine type and version that the configuration uses (for example, **RabbitMQ 3.xx.xx**) are displayed.
5. On the **Configuration details** tab, the configuration revision number, description, and broker configuration in Cuttlefish format are displayed.

   > **Note**
   >
   > Editing the current configuration creates a new configuration revision.
6. Choose **Edit configuration** and make changes to the Cuttlefish configuration.
7. Choose **Save**.

   The **Save revision** dialog box is displayed.
8. (Optional) Type `A description of the changes in this revision`.
9. Choose **Save**.

   The new revision of the configuration is saved.

> **Important**
>
> Making changes to a configuration does *not* apply the changes to the broker immediately. To apply your changes, you must wait for the next maintenance window or reboot the broker.
> Currently, you can't delete a configuration.

Configurable values

You can set the value of the following broker configuration options by modifying the broker configuration file in the AWS Management Console.

In addition to the values described in the following table, Amazon MQ supports additional broker configuration options related to authentication and authorization as well as resource limits. For more information about these configuration options, see

- OAuth 2.0 configuration
- LDAP configuration
- HTTP configuration
- SSL configuration
- mTLS configuration
- ARN support
- Resource limits
- AMQP client SSL configuration

Configuration	Default Value	Recommended Value	Values	Applicable Versions	Description
consumer_timeout	1800000 ms (30 minutes)	1800000 ms (30 minutes)	0 to 2,147,483,647 ms. Amazon MQ also supports the value 0,	All versions	A timeout on consumer delivery acknowledgement to detect when

Configuration	Default Value	Recommended Value	Values	Applicable Versions	Description
			which means "infinite".		consumers do not ack deliveries.
heartbeat	60 seconds	60 seconds	60 to 3600 seconds	All versions	Defines the time before a connection is considered unavailable by RabbitMQ.
management.restrictions.operator_policy_changes.disabled	true	true	true, false	All versions	Turns off making changes to the operator policies. If you make this change, you are highly encouraged to include the HA properties in your own operator policies.

Configuration	Default Value	Recommended Value	Values	Applicable Versions	Description
quorum_queue.property_equivalence.relaxed_checks_on_redeclaration	true	true	true, false	All versions	When set to TRUE, your application avoids a channel exception when redeclaring a quorum queue.
secure.management.http.headers.enabled	true	true	true, false	All versions	Turns on unmodifiable HTTP security headers.

Configuring consumer delivery acknowledgement

You can configure consumer_timeout to detect when consumers do not ack deliveries. If the consumer does not send an acknowledgment within the timeout value, the channel will be closed. For example, if you are using the default value 1800000 milliseconds, if the consumer does not send a delivery acknowledgement within 1800000 milliseconds, the channel will be closed. Amazon MQ also supports the value 0, which means "infinite".

Configuring heartbeat

You can configure a heartbeat timeout to find out when connections are disrupted or have failed. The heartbeat value defines the time limit before a connection is considered down.

Configuring operator policies

The default operator policy on each virtual host has the following recommended HA properties:

```
{
```

```
"name": "default_operator_policy_AWS_managed",
"pattern": ".*",
"apply-to": "all",
"priority": 0,
"definition": {
"ha-mode": "all",
"ha-sync-mode": "automatic"
}
}
```

Changes to the operator policies via the AWS Management Console or Management API are not available by default. You can enable changes by adding the following line to the broker configuration:

```
management.restrictions.operator_policy_changes.disabled=false
```

If you make this change, you are highly encouraged to include the HA properties in your own operator policies.

Configuring relaxed checks on queue declaration

If you have migrated your classic queues to quorum queues but not updated your client code, you can avoid a channel exception when redeclaring a quorum queue by configuring quorum_queue.property_equivalence.relaxed_checks_on_redeclaration set to true.

Configuring HTTP security headers

The secure.management.http.headers.enabled configuration enables the following HTTP security headers:

- X-Content-Type-Options: nosniff: prevents browsers from performing content sniffing, algorithms that are used to deduce the file format of websites.
- X-Frame-Options: DENY: prevents others from embedding the management plugin into a frame on their own website to deceive others
- Strict-Transport-Security: max-age=47304000; includeSubDomains: enforces browsers to use HTTPS when making subsequent connections to the website and its subdomains for a long period of time (1.5 years).

Amazon MQ for RabbitMQ brokers created on versions 3.10 and above will have secure.management.http.headers.enabled set to true by default. You can turn on these HTTP security headers by setting secure.management.http.headers.enabled to true. If you wish to opt out of these HTTP security headers, set secure.management.http.headers.enabled to false.

Configuring OAuth 2.0 authentication and authorization

For information about OAuth 2.0 configuration options and setting up OAuth 2.0 authentication for your brokers, see Supported OAuth 2.0 configurations and Using OAuth 2.0 authentication and authorization.

Configuring LDAP authentication and authorization

For information about LDAP configuration options and setting up LDAP authentication for your brokers, see Supported LDAP configurations and Using LDAP authentication and authorization.

Configuring HTTP authentication and authorization

For information about HTTP authentication configuration values and setting up HTTP authentication for your brokers, see HTTP authentication and authorization and Using HTTP authentication and authorization.

> **Note**
>
> The HTTP authentication plugin is only available for Amazon MQ for RabbitMQ version 4 and above.

Configuring SSL certificate authentication

For information about SSL certificate authentication configuration values and setting up SSL certificate authentication for your brokers, see SSL certificate authentication and Using SSL certificate authentication.

> **Note**
>
> The SSL certificate authentication plugin is only available for Amazon MQ for RabbitMQ version 4 and above.

Configuring mTLS

Amazon MQ for RabbitMQ supports mutual TLS (mTLS) for secure connections to various endpoints and external services. mTLS provides enhanced security by requiring both client and server to authenticate using certificates.

> **Note**
>
> The use of private certificate authorities for mTLS is only available for Amazon MQ for RabbitMQ version 4 and above.

> **Important**
>
> Amazon MQ for RabbitMQ enforces the use of AWS ARNs for certificate and private key files. See ARN support in RabbitMQ configuration for more details.

On this page

AMQP endpoint

Configure mTLS for client connections to the AMQP endpoint. This is used with SSL certificate authentication. For supported configurations, see SSL certificate authentication.

RabbitMQ management plugin

Configure mTLS for connections to the RabbitMQ management interface.

> **Note**
>
> Strict mTLS is not supported for the management API.

Supported configurations

`aws.arns.management.ssl.cacertfile`

Certificate authority file for validating client certificates connecting to the management interface.

`management.ssl.verify`

Peer verification mode. Supported values: `verify_none`, `verify_peer`

`management.ssl.depth`

Maximum certificate chain depth for verification.

`management.ssl.hostname_verification`

Hostname verification mode. Supported values: `wildcard`, none

Unsupported SSL options

The following SSL configuration values are not supported:

View complete list

- `management.ssl.cert`
- `management.ssl.client_renegotiation`
- `management.ssl.dh`
- `management.ssl.dhfile`
- `management.ssl.fail_if_no_peer_cert`
- `management.ssl.honor_cipher_order`
- `management.ssl.honor_ecc_order`
- `management.ssl.key.RSAPrivateKey`
- `management.ssl.key.DSAPrivateKey`
- `management.ssl.key.PrivateKeyInfo`

- `management.ssl.log_alert`
- `management.ssl.password`
- `management.ssl.psk_identity`
- `management.ssl.reuse_sessions`
- `management.ssl.secure_renegotiate`
- `management.ssl.versions.$version`
- `management.ssl.sni`

RabbitMQ OAuth 2.0 plugin

Configure mTLS for connections from Amazon MQ to the OAuth 2.0 identity provider. For supported configurations, see OAuth 2.0 authentication and authorization.

RabbitMQ HTTP authentication plugin

Configure mTLS for connections from Amazon MQ to the HTTP authentication server. For supported configurations, see HTTP authentication and authorization.

RabbitMQ LDAP plugin

Configure mTLS for connections from Amazon MQ to the LDAP server. For supported configurations, see LDAP authentication and authorization.

AMQP client connections

Configure TLS peer verification for AMQP client connections used by federation and shovel. For more information, see AMQP client SSL configuration.

> **Important**
>
> Amazon MQ does not currently support configuring client certificates for AMQP client connections. As a result, federation and shovel cannot connect to mTLS-enabled brokers that require client certificate authentication.

Resource Limit Configuration

Amazon MQ for RabbitMQ supports configuring broker resource limits from RabbitMQ 4 onwards. When you create a broker, Amazon MQ automatically applies default values to these resource

limits. These defaults act as guardrails to protect your broker availability while accommodating common customer usage patterns. You can customize your broker behavior by changing the limit configuration values to better match your specific workload requirements. For more details about default and maximum allowed values, see the section called “Sizing guidelines”.

Resource names and configuration keys

Resource Name	Configuration Key
Connection	`connection_max`
Channel	`channel_max_per_node`
Queue	`cluster_queue_limit`
Vhost	`vhost_max`
Shovel	`runtime_parameters.limits.shovel`
Exchange	`cluster_exchange_limit`
Consumer per channel	`consumer_max_per_channel`
Maximum message size	`max_message_size`

How to override resource limits

You can override resource limits using the Amazon MQ API and Amazon MQ console.

The following example shows how to override the queue count default limit using the AWS CLI:

```
aws mq update-configuration --configuration-id <config-id> --data "$(echo
 "cluster_queue_limit=500" | base64 --wrap=0)"
```

A successful invocation creates a configuration revision. You must associate the configuration to your RabbitMQ broker and reboot the broker to apply the override. For more details see RabbitMQ Broker Configurations

Resource limit override errors

Associating or creating a broker with configuration values outside the supported range results in an error response similar to the following:

```
Configuration Revision N for configuration:cluster_queue_limit has limit: of value:
 100000000 larger than maximum allowed limit:5000
```

ARN support in RabbitMQ configuration

Amazon MQ for RabbitMQ supports AWS ARNs for the values of some RabbitMQ configuration settings. This is enabled by the RabbitMQ community plugin rabbitmq-aws. This plugin is developed and maintained by Amazon MQ and can also be used in self-hosted RabbitMQ brokers not managed by Amazon MQ.

Important considerations

- The resolved ARN values retrieved by the aws plugin are passed directly to the RabbitMQ process at runtime. They are not stored elsewhere on the RabbitMQ node.
- Amazon MQ for RabbitMQ requires an IAM role that can be assumed by Amazon MQ to access the configured ARNs. This is configured by setting `aws.arns.assume_role_arn`.
- Users calling CreateBroker or UpdateBroker APIs with a broker configuration that includes an IAM role must have the `iam:PassRole` permission for that role.
- The IAM role must exist in the same AWS account as the RabbitMQ broker. All ARNs in the configuration must be present in the same AWS region as the RabbitMQ broker.
- Amazon MQ adds IAM global conditional keys `aws:SourceAccount` and `aws:SourceArn` when assuming the IAM role. These values must be used in the IAM policy attached to the role for confused deputy protection.

On this page

- Related broker quarantine states
- Example scenario

Supported keys

Required IAM role

`aws.arns.assume_role_arn`

IAM role ARN that Amazon MQ assumes to access other AWS resources. Required when any other ARN configuration is used.

AMQP endpoint

Configuration key	Description
`aws.arns.ssl_options.cacertfile`	Certificate authority file for SSL/TLS client connections. Amazon MQ requires using Amazon S3 or to store the certificate.

RabbitMQ management plugin

Configuration key	Description
`aws.arns.management.ssl.cacertfile`	Certificate authority file for management interface SSL/TLS connections. Amazon MQ requires using Amazon S3 or to store the certificate.

RabbitMQ OAuth 2.0 plugin

Configuration key	Description
`aws.arns.auth_oauth2.https.cacertfile`	Certificate authority file for OAuth 2.0 HTTPS connections. Amazon MQ requires using Amazon S3 or to store the certificate.

RabbitMQ HTTP authentication plugin

Configuration key	Description
`aws.arns.auth_http.ssl_options.cacertfile`	Certificate authority file for HTTP authentication SSL/TLS connections. Amazon MQ requires using Amazon S3 or to store the certificate.
`aws.arns.auth_http.ssl_options.certfile`	Certificate file for mutual TLS connections between Amazon MQ and the HTTP authentication server. Amazon MQ requires using Amazon S3 or to store the certificate.
`aws.arns.auth_http.ssl_options.keyfile`	Private key file for mutual TLS connections between Amazon MQ and the HTTP authentication server. Amazon MQ requires using AWS Secrets Manager to store the private key.

RabbitMQ LDAP plugin

Configuration key	Description
`aws.arns.auth_ldap.ssl_options.cacertfile`	Certificate authority file for LDAP SSL/TLS connections. Amazon MQ requires using Amazon S3 or to store the certificate.
`aws.arns.auth_ldap.ssl_options.certfile`	Certificate file for mutual TLS connections between Amazon MQ and the LDAP server. Amazon MQ requires using Amazon S3 or to store the certificate.
`aws.arns.auth_ldap.ssl_options.keyfile`	Private key file for mutual TLS connections between Amazon MQ and the LDAP server. Amazon MQ requires using AWS Secrets Manager to store the private key.
`aws.arns.auth_ldap.dn_lookup_bind.password`	Password for LDAP DN lookup bind. Amazon MQ requires using AWS Secrets Manager to store the password as a plaintext value.

Configuration key	Description
`aws.arns.auth_ldap.other_bind.password`	Password for LDAP other bind. Amazon MQ requires using AWS Secrets Manager to store the password as a plaintext value.

IAM policy samples

For IAM policy examples including assume role policy documents and role policy documents, see the CDK sample implementation.

See Using LDAP authentication and authorization for steps on how to set up AWS Secrets Manager and Amazon S3 resources.

Access validation

To troubleshoot scenarios where ARN values cannot be fetched, the aws plugin supports a RabbitMQ management API endpoint that can be called to check if Amazon MQ is able to successfully assume the role and resolve AWS ARNs. This avoids the need to update broker configuration, update broker with the new configuration revision and reboot broker to test configuration changes.

> **Note**
>
> Use of this API requires an existing RabbitMQ administrator user. Amazon MQ recommends creating test brokers with an internal user in addition to other access methods. See enabling both OAuth 2.0 and simple (internal) authentication. This user can then be used to access the validation API.

> **Note**
>
> Though aws plugin supports passing a new role as an input to the validation API, this parameter is not supported by Amazon MQ. The IAM role used for validation should match the value of `aws.arns.assume_role_arn` in broker configuration.

Related broker quarantine states

For information about broker quarantine states related to ARN support issues, see:

- RABBITMQ_INVALID_ASSUMEROLE
- RABBITMQ_INVALID_ARN_LDAP
- RABBITMQ_INVALID_ARN

Example scenario

- Broker `b-f0fc695e-2f9c-486b-845a-988023a3e55b` has been configured to use IAM role `<role>` to access AWS Secrets Manager secret `<arn>`
- If the role provided to Amazon MQ does not have read permission on the AWS Secrets Manager secret, the following error will be shown in RabbitMQ logs:

  ```
  [error] <0.254.0> aws_arn_config: {handle_assume_role,{error,{assume_role_failed,"AWS
   service is unavailable"}}}
  ```

  Additionally, the broker will enter the `INVALID_ASSUMEROLE` quarantine state. For more information, see INVALID_ASSUMEROLE.
- LDAP authentication attempts will fail with the following error:

  ```
  [error] <0.254.0> LDAP bind failed: invalid_credentials
  ```
- Fix the IAM role with the proper permissions
- Call the validation endpoint to verify if RabbitMQ is now able to access the secret:

  ```
  curl -4su 'guest:guest' -XPUT -H 'content-type: application/json' <broker-endpoint>/
  api/aws/arn/validate -d '{"assume_role_arn":"arn:aws:iam::<account-id>:role/<role-
  name>","arns":["arn:aws:secretsmanager:<region>:<account-id>:secret:<secret-name>"]}'
   | jq '.'
  ```

AMQP client SSL configuration

Federation and shovel use AMQP for communication between upstream and downstream brokers. By default, *TLS peer verification* is enabled for AMQP clients in Amazon MQ for RabbitMQ 4. With

this setting, federation and shovel AMQP clients running on Amazon MQ brokers will perform peer verification when establishing connections with upstream broker.

AMQP clients running on Amazon MQ brokers support the same certificate authorities as Mozilla. If you don't use ACM, use a certificate issued by a CA on the Mozilla Included CA Certificate List. If your on-premises broker uses certificates from other certificate authorities, SSL verification will fail. You can disable *TLS peer verification* for these use cases.

Important

Amazon MQ does not currently support configuring client certificates for AMQP client connections. As a result, federation and shovel cannot connect to mTLS-enabled brokers that require client certificate authentication.

Important

On Amazon MQ for RabbitMQ 3 SSL properties of AMQP clients is configured with RabbitMQ defaults*(verify_none)*. Amazon MQ for RabbitMQ 3 does not support overriding these defaults.

Note

With the default `verify_peer` setting, you can establish federation and shovel connections between any 2 Amazon MQ brokers but this does not support establishing the connection between Amazon MQ broker and private brokers or on-premises brokers that are running with non-Amazon MQ CA certificates. To connect with private or on-premises brokers, you need to disable peer verification on the downstream Amazon MQ broker.

AMQP client SSL configuration key

Configuration	Configuration Key	Supported Values
AMQP client SSL peer verification	`amqp_client.ssl_options.verify`	`verify_none` , `verify_peer`

How to override AMQP client SSL peer verification

You can override AMQP client SSL peer verification using the Amazon MQ API and Amazon MQ console on RabbitMQ 4 brokers.

The following example shows how to override the AMQP client SSL peer verification using the AWS CLI:

```
aws mq update-configuration --configuration-id <config-id> --data "$(echo
 "amqp_client.ssl_options.verify=verify_none" | base64 --wrap=0)"
```

A successful invocation creates a configuration revision. You must associate the configuration to your RabbitMQ broker and reboot the broker to apply the override. For more details see Creating and applying broker configurations

Important

When using `verify_none`, SSL encryption is still active, but the identity of the peer is not verified. Use this setting only when necessary and ensure that you trust the network path to the destination broker.

Amazon MQ for RabbitMQ Authentication and Authorization

Amazon MQ for RabbitMQ supports the following authentication and authorization methods:

Simple authentication and authorization

In this method, broker users are stored internally in the RabbitMQ broker and managed through the web console or management API. Permissions for vhosts, exchanges, queues, and topics are configured directly in RabbitMQ. This is the default method. For more information, see Simple authentication and authorization.

OAuth 2.0 authentication and authorization

In this method, broker users and their permissions are managed by an external OAuth 2.0 identity provider (IdP). User authentication and resource permissions for vhosts, exchanges, queues,

and topics are centralized through the OAuth 2.0 provider's scope system. This simplifies user management and enables integration with existing identity systems. For more information, see OAuth 2.0 authentication and authorization.

IAM authentication and authorization

In this method, broker users authenticate using AWS IAM credentials through IAM outbound federation. IAM credentials are used to obtain JWT tokens from AWS Security Token Service (STS), and these JWT tokens serve as OAuth 2.0 tokens for authentication. This method leverages the existing OAuth 2.0 support in Amazon MQ for RabbitMQ, where AWS acts as the OAuth 2.0 identity provider. User authentication is handled by AWS IAM, while resource permissions for vhosts, exchanges, queues, and topics are managed through IAM policies and scope aliases configured in RabbitMQ. For more information, see IAM authentication and authorization.

LDAP authentication and authorization

In this method, broker users and their permissions are managed by an external LDAP directory service. User authentication and resource permissions are centralized through the LDAP server, allowing users to access RabbitMQ using their existing directory service credentials. For more information, see LDAP authentication and authorization.

HTTP authentication and authorization

In this method, broker users and their permissions are managed by an external HTTP server. User authentication and resource permissions are centralized through the HTTP server, allowing users to access RabbitMQ using their own Authentication and Authorization provider. For more information about this method, see HTTP authentication and authorization.

SSL certificate authentication

Amazon MQ supports mutual TLS (mTLS) for RabbitMQ brokers. The SSL authentication plugin uses client certificates from mTLS connections to authenticate users. In this method, broker users are authenticated using X.509 client certificates instead of username and password credentials. The client's certificate is validated against a trusted Certificate Authority (CA), and the username is extracted from a field in the certificate, such as the Common Name (CN) or Subject Alternative Name (SAN). This method provides strong authentication without transmitting credentials over the network. For more information, see SSL certificate authentication.

> **Note**
>
> RabbitMQ supports multiple authentication and authorization methods to be used simultaneously. For example, you can enable both OAuth 2.0 and simple (internal) authentication. For more information, see the OAuth 2.0 tutorial section on enabling both OAuth 2.0 and simple (internal) authentication and the RabbitMQ access control documentation.
> Amazon MQ recommends creating an internal user when testing authentication configurations. This allows access configuration to be validated using RabbitMQ management API. For more information, see Access validation.

Simple authentication and authorization

Amazon MQ for RabbitMQ broker users

> **Note**
>
> This topic describes managing broker users with RabbitMQ's default internal authentication and authorization mechanism. For information about all supported authentication and authorization methods, see Amazon MQ for RabbitMQ Authentication and Authorization.

Every AMQP 0-9-1 client connection has an associated user. This user must be authenticated. Each client connection also targets a virtual host (vhost). The user must have a set of permissions for this vhost. A user may have permission to **configure**, **write** to, and **read** from queues and exchanges in a vhost. You specify user credentials and the target vhost when the connection is established.

When you first create an Amazon MQ for RabbitMQ broker, Amazon MQ uses the sign-in credentials you provide to create a RabbitMQ user with the `administrator` tag. You can then add and manage users via the RabbitMQ management API or the RabbitMQ web console. You can also use the RabbitMQ web console or the management API to set or modify user permissions and tags.

> **Note**
>
> RabbitMQ users will not be stored or displayed via the Amazon MQ Users API.

Important

Amazon MQ for RabbitMQ does not support the username "guest", and will delete the default guest account when you create a new broker. Amazon MQ will also periodically delete any customer created account called "guest".

To create a new user with the RabbitMQ management API, use the following API endpoint and request body. Replace *username* and *password* with your new sign-in credentials.

```
PUT /api/users/username HTTP/1.1

    {"password":"password","tags":"administrator"}
```

Important

- Do not add personally identifiable information (PII) or other confidential or sensitive information in broker usernames. Broker usernames are accessible to other AWS services, including CloudWatch Logs. Broker usernames are not intended to be used for private or sensitive data.
- If you lose access to all administrator accounts, see recovering broker access to use IAM authentication for recovery.

The `tags` key is mandatory, and is a comma-separated list of tags for the user. Amazon MQ supports `administrator`, `management`, `monitoring`, and `policymaker` user tags.

You can set permissions for an individual user by using the following API endpoint and request body. Replace *vhost* and *username* with your information. For the default vhost `/`, use `%2F`.

```
PUT /api/permissions/vhost/username HTTP/1.1

    {"configure":".*","write":".*","read":".*"}
```

Note

The `configure`, `read`, and `write` keys are all mandatory.

By using the wildcard `.*` value, this operation will grant read, write, and configure permissions for all queues in the specified vhost to the user. For more information about managing users via the RabbitMQ management API, see RabbitMQ Management HTTP API.

OAuth 2.0 authentication and authorization for Amazon MQ for RabbitMQ

Amazon MQ for RabbitMQ supports multiple authentication and authorization methods. For information about all supported methods, see Authentication and authorization for Amazon MQ for RabbitMQ brokers.

In OAuth 2.0 authentication and authorization, broker users and their permissions are managed by an external OAuth 2.0 identity provider (IdP). User authentication and resource permissions for vhosts, exchanges, queues, and topics are centralized through the OAuth 2.0 provider's scope system. This simplifies user management and enables integration with existing identity systems.

⚠ Important considerations

- OAuth 2.0 integration isn't supported on Amazon MQ for ActiveMQ brokers.
- Amazon MQ for RabbitMQ doesn't support server certificate issued by a private CA.
- The RabbitMQ OAuth 2.0 plugin doesn't support token introspection endpoints and opaque access tokens. It also doesn't perform token revocation checks.
- You must include the IAM permission, `mq:UpdateBrokerAccessConfiguration`, to enable OAuth 2.0 on existing brokers.
- Amazon MQ automatically creates a system user named `monitoring-AWS-OWNED-DO-NOT-DELETE` with monitoring-only permissions. This user uses RabbitMQ's internal authentication system even on OAuth 2.0-enabled brokers and is restricted to loopback interface access only.

For information about how to configure OAuth 2.0 for your Amazon MQ for RabbitMQ brokers, see Using OAuth 2.0 authentication and authorization.

On this page

Supported OAuth 2.0 configurations

Amazon MQ for RabbitMQ supports all configurable variables in RabbitMQ OAuth 2.0 plugin, with the following exceptions:

- `auth_oauth2.https.cacertfile`
- `auth_oauth2.oauth_providers.{id/index}.https.cacertfile`
- `management.oauth_client_secret`

  Because Amazon MQ doesn't support this key, we don't support UAA as an IdP.
- `management.oauth_resource_servers.{id/index}.oauth_client_secret`
- `auth_oauth2.signing_keys.{id/index}`

Additional validations for OAuth 2.0 authentication

Amazon MQ also enforces the following additional validations for OAuth 2.0 authentication:

- All URLs need to start with `https://`.
- Supported signature algorithms: Ed25519, Ed25519ph, Ed448, Ed448ph, EdDSA, ES256K, ES256, ES384, ES512, HS256, HS384, HS512, PS256, PS384, PS512, RS256, RS384, and RS512.

IAM authentication and authorization for Amazon MQ for RabbitMQ

Amazon MQ for RabbitMQ supports multiple authentication and authorization methods. For information about all supported methods, see Authentication and authorization for Amazon MQ for RabbitMQ brokers.

IAM authentication and authorization allows broker users to authenticate using AWS IAM credentials through IAM outbound federation. In this method, IAM credentials are used to obtain JWT tokens from AWS Security Token Service (STS). These JWT tokens serve as OAuth 2.0 tokens for authentication, leveraging the existing OAuth 2.0 support in Amazon MQ for RabbitMQ where AWS acts as the OAuth 2.0 identity provider. AWS IAM handles user authentication, while resource permissions for virtual hosts, exchanges, queues, and topics are managed through IAM policies and scope aliases configured in RabbitMQ.

Important considerations

- IAM authentication is supported on RabbitMQ versions 3.13, 4.2 and above. It isn't supported on Amazon MQ for ActiveMQ brokers.
- IAM authentication requires IAM outbound federation to be configured and available in your AWS account.
- This method builds on the existing OAuth 2.0 infrastructure in Amazon MQ for RabbitMQ, with AWS serving as the OAuth 2.0 identity provider.
- Amazon MQ automatically creates a system user named `monitoring-AWS-OWNED-DO-NOT-DELETE` with monitoring-only permissions. This user uses RabbitMQ's internal authentication system even on IAM-enabled brokers and is restricted to loopback interface access only.

On this page

How IAM authentication works

IAM authentication for Amazon MQ for RabbitMQ uses IAM outbound federation to enable AWS IAM credentials to authenticate with RabbitMQ brokers. IAM credentials are used to obtain JWT tokens from AWS Security Token Service (STS), and these JWT tokens serve as OAuth 2.0 tokens for authentication with the RabbitMQ broker.

Limitations

IAM authentication for Amazon MQ for RabbitMQ has the following limitation:

- **Scope claim configuration** – You cannot use a scope claim directly because the JWT token from STS is nested. The key is `sts.amazonaws.com`, which requires using scope aliases in the RabbitMQ configuration to map IAM roles to RabbitMQ permissions. This limitation also prevents using IAM policies for authorization fully, requiring RabbitMQ configuration for authorization instead.

For information about how to configure IAM authentication and authorization for your Amazon MQ for RabbitMQ brokers, see Using IAM authentication and authorization.

HTTP authentication and authorization for Amazon MQ for RabbitMQ

Amazon MQ for RabbitMQ supports authentication and authorization of broker users using an external HTTP server. For other supported methods, see Authentication and authorization for Amazon MQ for RabbitMQ brokers.

> **Note**
>
> The HTTP authentication plugin is only available for Amazon MQ for RabbitMQ version 4 and above.

> **Important considerations**
>
> - The HTTP server needs to be accessible over the public internet. Amazon MQ for RabbitMQ can be configured to authenticate to the HTTP server using mutual TLS.
> - Amazon MQ for RabbitMQ enforces the use of AWS ARNs for settings that require access to the local file system. See ARN support in RabbitMQ configuration for more details.
> - You must include the IAM permission, `mq:UpdateBrokerAccessConfiguration`, to enable HTTP authentication on existing brokers.
> - Amazon MQ automatically creates a system user named `monitoring-AWS-OWNED-DO-NOT-DELETE` with monitoring-only permissions. This user uses RabbitMQ's internal authentication system even on HTTP-enabled brokers and is restricted to loopback interface access only. Amazon MQ prevents deletion of this user by adding the protected user tag.

For information about how to configure HTTP authentication for your Amazon MQ for RabbitMQ brokers, see Using HTTP authentication and authorization.

On this page

Supported HTTP configurations

Amazon MQ for RabbitMQ supports all configurable variables in RabbitMQ HTTP authentication plugin, with the following exceptions that require AWS ARNs. For details about ARN support, see ARN support in RabbitMQ configuration.

Configurations requiring ARNs

`auth_http.ssl_options.cacertfile`

Use `aws.arns.auth_http.ssl_options.cacertfile` instead

`auth_http.ssl_options.certfile`

Use `aws.arns.auth_http.ssl_options.certfile` instead

`auth_http.ssl_options.keyfile`

Use `aws.arns.auth_http.ssl_options.keyfile` instead

Unsupported SSL options

The following SSL configuration options are also not supported:

View complete list

- `auth_http.ssl_options.cert`
- `auth_http.ssl_options.client_renegotiation`
- `auth_http.ssl_options.dh`
- `auth_http.ssl_options.dhfile`
- `auth_http.ssl_options.honor_cipher_order`
- `auth_http.ssl_options.honor_ecc_order`
- `auth_http.ssl_options.key.RSAPrivateKey`
- `auth_http.ssl_options.key.DSAPrivateKey`
- `auth_http.ssl_options.key.PrivateKeyInfo`
- `auth_http.ssl_options.log_alert`
- `auth_http.ssl_options.password`
- `auth_http.ssl_options.psk_identity`

- `auth_http.ssl_options.reuse_sessions`
- `auth_http.ssl_options.secure_renegotiate`
- `auth_http.ssl_options.versions.$version`
- `auth_http.ssl_options.sni`
- `auth_http.ssl_options.crl_check`

Additional validations for HTTP configurations in Amazon MQ

Amazon MQ also enforces the following additional validations for HTTP authentication and authorization:

- `auth_http.http_method` must be either `get` or `post`
- The following path configurations must use HTTPS URLs:
  - `auth_http.user_path`
  - `auth_http.vhost_path`
  - `auth_http.resource_path`
  - `auth_http.topic_path`
- If any setting requires the use of an AWS ARN, `aws.arns.assume_role_arn` must be provided.

SSL certificate authentication for Amazon MQ for RabbitMQ

Amazon MQ for RabbitMQ supports authentication of broker users using X.509 client certificates. For other supported methods, see Authentication and authorization for Amazon MQ for RabbitMQ brokers.

Note

The SSL certificate authentication plugin is only available for Amazon MQ for RabbitMQ version 4 and above.

Important considerations

- Client certificates must be signed by a trusted Certificate Authority (CA). Amazon MQ for RabbitMQ validates the certificate chain during authentication.

- Amazon MQ for RabbitMQ enforces the use of AWS ARNs for certificate-related settings such as CA certificates and for settings that require access to the local file system. See ARN support in RabbitMQ configuration for more details.
- Amazon MQ automatically creates a system user named `monitoring-AWS-OWNED-DO-NOT-DELETE` with monitoring-only permissions. This user uses RabbitMQ's internal authentication system even on SSL certificate-enabled brokers and is restricted to loopback interface access only. Amazon MQ prevents deletion of this user by adding the protected user tag.

For information about how to configure SSL certificate authentication for your Amazon MQ for RabbitMQ brokers, see Using SSL certificate authentication.

On this page

Supported SSL configurations

Amazon MQ for RabbitMQ supports SSL/TLS configuration for client connections. For details about ARN support, see ARN support in RabbitMQ configuration.

Configurations requiring ARNs

`ssl_options.cacertfile`

Use `aws.arns.ssl_options.cacertfile` instead

SSL certificate login configurations

The following configurations control how usernames are extracted from client certificates:

`ssl_cert_login_from`

Specifies which certificate field to use for username extraction. Supported values:

- `distinguished_name` - Use the full Distinguished Name
- `common_name` - Use the Common Name (CN) field

- `subject_alternative_name` or `subject_alt_name` - Use Subject Alternative Name

`ssl_cert_login_san_type`

When using Subject Alternative Name, specifies the SAN type. Supported values: `dns`, `ip`, `email`, `uri`, `other_name`

`ssl_cert_login_san_index`

When using Subject Alternative Name, specifies the index of the SAN entry to use (zero-based). Must be a non-negative integer.

SSL options for client connections

The following SSL options apply to client connections:

`ssl_options.verify`

Peer verification mode. Supported values: `verify_none`, `verify_peer`

`ssl_options.fail_if_no_peer_cert`

Whether to reject connections if client doesn't provide a certificate. Boolean value.

`ssl_options.depth`

Maximum certificate chain depth for verification.

`ssl_options.hostname_verification`

Hostname verification mode. Supported values: `wildcard`, `none`

Unsupported SSL options

The following SSL configuration options are not supported:

View complete list

- `ssl_options.cert`
- `ssl_options.client_renegotiation`
- `ssl_options.dh`
- `ssl_options.dhfile`
- `ssl_options.honor_cipher_order`

- `ssl_options.honor_ecc_order`
- `ssl_options.key.RSAPrivateKey`
- `ssl_options.key.DSAPrivateKey`
- `ssl_options.key.PrivateKeyInfo`
- `ssl_options.log_alert`
- `ssl_options.password`
- `ssl_options.psk_identity`
- `ssl_options.reuse_sessions`
- `ssl_options.secure_renegotiate`
- `ssl_options.versions.$version`
- `ssl_options.sni`
- `ssl_options.crl_check`

Additional validations for SSL configurations in Amazon MQ

Amazon MQ also enforces the following additional validations for SSL certificate authentication:

- If any setting requires the use of an AWS ARN, `aws.arns.assume_role_arn` must be provided.

LDAP authentication and authorization for Amazon MQ for RabbitMQ

Amazon MQ for RabbitMQ supports authentication and authorization of broker users using an external LDAP server. For other supported methods, see Authentication and authorization for Amazon MQ for RabbitMQ brokers.

⚠ Important considerations

- The LDAP server needs to be accessible over the public internet. Amazon MQ for RabbitMQ can be configured to authenticate to the LDAP server using mutual TLS.
- Amazon MQ for RabbitMQ enforces the use of AWS ARNs for sensitive LDAP settings such as passwords and for settings that require access to the local file system. See ARN support in RabbitMQ configuration for more details.
- You must include the IAM permission, `mq:UpdateBrokerAccessConfiguration`, to enable LDAP on existing brokers.

- Amazon MQ automatically creates a system user named `monitoring-AWS-OWNED-DO-NOT-DELETE` with monitoring-only permissions. This user uses RabbitMQ's internal authentication system even on LDAP-enabled brokers and is restricted to loopback interface access only. Amazon MQ prevents deletion of this user by adding the protected user tag.

For information about how to configure LDAP for your Amazon MQ for RabbitMQ brokers, see Using LDAP authentication and authorization.

On this page

Supported LDAP configurations

Amazon MQ for RabbitMQ supports all configurable variables in RabbitMQ LDAP plugin, with the following exceptions that require AWS ARNs. For details about ARN support, see ARN support in RabbitMQ configuration.

Configurations requiring ARNs

`auth_ldap.dn_lookup_bind.password`

Use `aws.arns.auth_ldap.dn_lookup_bind.password` instead

`auth_ldap.other_bind.password`

Use `aws.arns.auth_ldap.other_bind.password` instead

`auth_ldap.ssl_options.cacertfile`

Use `aws.arns.auth_ldap.ssl_options.cacertfile` instead

`auth_ldap.ssl_options.certfile`

Use `aws.arns.auth_ldap.ssl_options.certfile` instead

`auth_ldap.ssl_options.keyfile`

Use `aws.arns.auth_ldap.ssl_options.keyfile` instead

Unsupported SSL options

The following SSL configuration options are also not supported:

View complete list

- `auth_ldap.ssl_options.cert`
- `auth_ldap.ssl_options.client_renegotiation`
- `auth_ldap.ssl_options.dh`
- `auth_ldap.ssl_options.dhfile`
- `auth_ldap.ssl_options.honor_cipher_order`
- `auth_ldap.ssl_options.honor_ecc_order`
- `auth_ldap.ssl_options.key.RSAPrivateKey`
- `auth_ldap.ssl_options.key.DSAPrivateKey`
- `auth_ldap.ssl_options.key.PrivateKeyInfo`
- `auth_ldap.ssl_options.log_alert`
- `auth_ldap.ssl_options.password`
- `auth_ldap.ssl_options.psk_identity`
- `auth_ldap.ssl_options.reuse_sessions`
- `auth_ldap.ssl_options.secure_renegotiate`
- `auth_ldap.ssl_options.versions.$version`
- `auth_ldap.ssl_options.sni`

Additional validations for LDAP configurations in Amazon MQ

Amazon MQ also enforces the following additional validations for LDAP authentication and authorization:

- `auth_ldap.log` cannot be set to `network_unsafe`
- LDAP server must use LDAPS. Either `auth_ldap.use_ssl` or `auth_ldap.use_starttls` must be explicitly enabled
- If any setting requires the use of an AWS ARN, `aws.arns.assume_role_arn` must be provided.

- `auth_ldap.servers` must be a valid IP address or valid FQDN
- The following keys must be a valid LDAP Distinguished Name:
  - `auth_ldap.dn_lookup_base`
  - `auth_ldap.dn_lookup_bind.user_dn`
  - `auth_ldap.other_bind.user_dn`
  - `auth_ldap.group_lookup_base`

Plugins

Amazon MQ for RabbitMQ also supports the following plugins.

- RabbitMQ management plugin
- Shovel plugin
- Federation plugin
- Consistent Hash exchange plugin
- OAuth 2 plugin
- LDAP plugin
- HTTP plugin
- SSL certificate plugin
- aws plugin
- JMS Topic Exchange plugin

RabbitMQ management plugin

Amazon MQ for RabbitMQ supports the RabbitMQ management plugin, which provides an HTTP-based management API along with a browser based UI for the RabbitMQ web console. You can use the web console and the management API to create and manage broker users and policies.

Shovel plugin

Amazon MQ for RabbitMQ supports the RabbitMQ shovel plugin, which allows you to move messages from queues and exchanges on one broker to another. You can use shovel to connect loosely coupled brokers and distribute messages away from nodes with heavier message loads.

> **Important**
>
> You cannot configure shovel between queues or exchanges if the shovel destination is a private broker.
> Amazon MQ does not support using static shovels.

Only dynamic shovels are supported. Dynamic shovels are configured using runtime parameters and can be started and stopped at any time programmatically by a client connection. For example, using the RabbitMQ management API, you can create a PUT request to the following API endpoint to configure a dynamic shovel. In the example, {vhost} can be replaced by the name of the broker's vhost, and {name} replaced by the name of the new dynamic shovel.

```
/api/parameters/shovel/{vhost}/{name}
```

In the request body, you must specify either a queue or an exchange but not both. This example below configures a dynamic shovel between a local queue specified in src-queue and a remote queue defined in dest-queue. Similarly, you can use src-exchange and dest-exchange parameters to configure a shovel between two exchanges.

```
{
"value": {
"src-protocol": "amqp091",
"src-uri": "amqp://localhost",
"src-queue": "source-queue-name",
"dest-protocol": "amqp091",
"dest-uri": "amqps://b-c8352341-ec91-4a78-ad9c-a43f23d325bb.mq.us-
west2.amazonaws.com:5671",
"dest-queue": "destination-queue-name"
}
}
```

Federation plugin

Amazon MQ supports federated exchanges and queues using the RabbitMQ federation plugin. With federation, you can replicate the flow of messages between queues, exchanges and consumers on separate brokers. Federated queues and exchanges use point-to-point links to connect to peers in other brokers. While federated exchanges, by default, route messages once, federated queues can move messages any number of times as needed by consumers.

You can use federation to allow a downstream broker to consume a message from an exchange or a queue on an upstream. You can enable federation on downstream brokers by using the RabbitMQ web console or the management API.

> **Important**
>
> You cannot configure federation if the upstream queue or exchange is in a private broker. You can only configure federation between queues or exchanges in public brokers, or between an upstream queue or exchange in a public broker, and a downstream queue or exchange in a private broker.

For example, using the management API, you can configure federation by doing the following.

- Configure one or more upstreams that define federation connections to other nodes. You can define federation connections by using the RabbitMQ web console or the management API. Using the management API, you can create a POST request to /api/parameters/federation-upstream/%2f/myupstream with the following request body.

  ```
  {"value":{"uri":"amqp://server-name","expires":3600000}}
  ```

- Configure a policy to enable your queues or exchanges to become federated. You can configure policies by using the RabbitMQ web console, or the management API. Using the management API, you can create a POST request to /api/policies/%2f/federate-me with the following request body.

  ```
  {"pattern":"^amq\.", "definition":{"federation-upstream-set":"all"}, "apply-to":"exchanges"}
  ```

  > **Note**
  >
  > The request body assumes exchanges on the server are named beginning with amq. Using regular expression ^amq\. will ensure that federation is enabled for all exchanges whose names begin with "amq." The exchanges on your RabbitMQ server can be named differently.

Consistent Hash exchange plugin

Amazon MQ for RabbitMQ supports the RabbitMQ Consistent Hash Exchange Type plugin. Consistent Hash exchanges route messages to queues based on a hash value calculated from the routing key of a message. Given a reasonably even routing key, Consistent Hash exchanges can distribute messages between queues reasonably evenly.

For queues bound to a Consistent Hash exchange, the binding key is a number-as-a-string that determines the binding weight of each queue. Queues with a higher binding weight will receive a proportionally higher distribution of messages from the Consistent Hash exchange to which they are bound. In a Consistent Hash exchange topology, publishers can simply publish messages to the exchange, but consumers must be explicitly configured to consume messages from specific queues.

OAuth 2.0 plugin

Amazon MQ for RabbitMQ supports the OAuth 2 authentication backend plugin. This plugin is conditionally enabled based on your broker configuration. When enabled, this plugin provides OAuth 2.0 authentication and authorization with integration to external OAuth 2.0 identity providers for centralized user management and access control. For more information about OAuth 2.0 authentication, see OAuth 2.0 authentication and authorization.

LDAP plugin

Amazon MQ for RabbitMQ supports the LDAP authentication backend plugin. This plugin is conditionally enabled based on your broker configuration. When enabled, this plugin provides LDAP authentication and authorization with integration to external LDAP directory services for centralized user authentication and authorization. For more information about LDAP authentication, see LDAP authentication and authorization.

HTTP plugin

Amazon MQ for RabbitMQ supports the HTTP authentication backend plugin. This plugin is conditionally enabled based on your broker configuration. When enabled, this plugin provides HTTP authentication and authorization with integration to external HTTP servers for centralized user authentication and authorization. For more information about HTTP authentication, see HTTP authentication and authorization.

> **Note**
>
> The HTTP authentication plugin is only available for Amazon MQ for RabbitMQ version 4 and above.

SSL certificate plugin

Amazon MQ supports mutual TLS (mTLS) for RabbitMQ brokers. The SSL authentication plugin uses client certificates from mTLS connections to authenticate users. This plugin is conditionally enabled based on your broker configuration. When enabled, it provides certificate-based authentication using X.509 client certificates for strong authentication without transmitting credentials over the network. For more information about SSL certificate authentication, see SSL certificate authentication.

> **Note**
>
> The SSL certificate authentication plugin is only available for Amazon MQ for RabbitMQ version 4 and above.

aws plugin

The aws plugin is conditionally enabled by Amazon MQ for RabbitMQ based on your broker configuration. This community plugin, developed and maintained by Amazon MQ, provides secure retrieval of credentials and certificates from AWS services using AWS ARNs in RabbitMQ configuration settings. For more information about ARN support, see ARN support in RabbitMQ configuration.

JMS Topic Exchange plugin

The JMS Topic Exchange Plugin is always enabled by Amazon MQ for RabbitMQ. It works with RabbitMQ JMS client to allow new and existing JMS applications to connect to Amazon MQ for RabbitMQ.

> **Note**
>
> The JMS Topic Exchange plugin is only available for Amazon MQ for RabbitMQ version 4 and above. It is enabled by default but only activates when the RabbitMQ JMS client is used to run JMS workloads.

Supported protocols

You can access your RabbitMQ brokers by using any programming language that RabbitMQ supports and by enabling TLS for any of the following protocol specifications:

- AMQP (0-9-1)
- AMQP 1.0
- JMS 1.1
- JMS 2.0
- JMS 3.1

Amazon MQ for RabbitMQ JMS support

You can now run JMS 1.1, 2.0, and 3.1 workloads on Amazon MQ for RabbitMQ 4 with RabbitMQ JMS client.

RabbitMQ JMS client

The RabbitMQ JMS client is an open-source JMS client library that you need to connect your JMS application to Amazon MQ RabbitMQ brokers. For more information, please visit the official GitHub repository.

Supported JMS 1.1, 2.0 and 3.1 APIs

From Amazon MQ for RabbitMQ 4 onward, the plugin `jms-topic-exchange` is always enabled. Hence, you can use Amazon MQ for RabbitMQ 4 and RabbitMQ JMS client for your JMS workload. All JMS APIs defined in the JMS 1.1 are supported except:

- Server sessions APIs are not supported.
- XA transaction APIs are not supported.

- JMS selector for JMS Queue destination is not supported.
- JMS `NoLocal` subscription attribute is not supported.

All newly added APIs in JMS 2.0 and JMS 3.1 are supported except:

- `JMSProducer.setDeliveryDelay` API is not supported.

To learn more about connecting your JMS application to Amazon MQ for RabbitMQ broker, please see the tutorial on Connecting your JMS application to Amazon MQ for RabbitMQ broker

Authentication and Authorization

All authentication and authorization mechanisms listed in this section are supported. The credentials used for connecting to the broker using the JMS client are the same as if you were connecting to the RabbitMQ broker using an AMQP Java client.

Interoperability with AMQP queues on RabbitMQ

You can use the RabbitMQ JMS client to send JMS messages to an AMQP exchange and consume messages from an AMQP queue (this feature does not support JMS topics). This enables you to interoperate or migrate certain JMS workloads to AMQP workloads. For more information, please visit the official client documentation.

Applying policies to Amazon MQ for RabbitMQ

You can apply custom policies and limits with Amazon MQ recommended default values. If you have deleted the recommended default policies and limits, and want to re-create them, or you have created additional vhosts and want to apply the default policies and limits to your new vhosts, you can use the following steps.

Important

On Amazon MQ for RabbitMQ engine versions 3.13 and below, the current default operator policy is:

```
vhost name pattern apply-to definition priority/
 default_operator_policy_AWS_managed .* classic_queues {"ha-mode":"all","ha-
sync-mode":"automatic","queue-version":2} 0
```

On versions 4.0 and above, the default operator policy has changed to:

```
vhost name pattern apply-to definition priority/
 default_operator_policy_AWS_managed .* classic_queues {"queue-version":2} 0
```

This change is required because classic queue mirroring and HA policy settings are not supported in RabbitMQ 4.
You cannot create a policy that applies to both classic mirrored queues and quorum queues. If you want your policy to only apply to quorum queues, you must set `--apply-to` to `quorum_queues`. If you are using classic mirrored queues and quorum queues, you must create a separate policy with `--apply-to:classic_queues` as well as a quorum queues policy.

Important

To perform the following steps, you must have an Amazon MQ for RabbitMQ broker user with administrator permissions. You can use the administrator user created when you first created the broker, or another user that you might have created afterwards. The following table provides the necessary administrator user tag and permissions as regular expression (regexp) patterns.

Tags	Read regexp	Configure regexp	Write regexp
`administrator`	`.*`	`.*`	`.*`

For more information about creating RabbitMQ users and managing user tags and permissions, see Amazon MQ for RabbitMQ broker users.

To apply default policies and virtual host limits using the RabbitMQ web console

1. Sign in to the Amazon MQ console.
2. In the left navigation pane, choose **Brokers**.
3. From the list of brokers, choose the name of the broker to which you want to apply the new policy.

4. On the broker details page, in the **Connections** section, choose the **RabbitMQ web console** URL. The RabbitMQ web console opens in a new browser tab or window.
5. Log in to the RabbitMQ web console with your broker administrator user name and password.
6. In the RabbitMQ web console, at the top of the page, choose **Admin**.
7. On the **Admin** page, in the right navigation pane, choose **Policies**.
8. On the **Policies** page, you can see a list of the broker's current **User policies**. Below **User policies**, expand **Add / update a policy**.
9. To create a new broker policy, under **Add / update a policy**, do the following:

    a. For **Virtual host**, choose the name of the vhost to which you want to attach the policies from the dropdown list. To choose the default vhost, choose **/**.

    > **Note**
    >
    > If you have not created additional vhosts, the **Virtual host** option is not shown in the RabbitMQ console, and the policies are applied only to the default vhost.

    b. For **Name**, enter a name for your policy, for example, **`policy-defaults`**.

    c. For **Pattern**, enter the regexp pattern **`.*`** so that the policy matches all queues on the broker.

    d. For **Apply to**, choose **Exchanges and queues** from the dropdown list.

    e. For **Priority**, enter an integer greater than all other policies applied to the vhost. You can apply exactly one set of policy definitions to RabbitMQ queues and exchanges at any given time. RabbitMQ chooses the matching policy with the highest priority value. For more information about policy priorities and how to combine policies, see Policies in the RabbitMQ Server Documentation.

    f. For **Definition**, add the following key-value pairs:

    - **`queue-mode`**=**`lazy`**. Choose **String** from the dropdown list.
    - **`overflow`**=**`reject-publish`**. Choose **String** from the dropdown list.

    > **Note**
    >
    > Does not apply to single-instance brokers.

- **`max-length=number-of-messages`**. Replace *`number-of-messages`* with the Amazon MQ recommended value according to the broker's instance size and deployment mode, for example, **`8000000`** for an `mq.m7g.large` cluster. Choose **Number** from the dropdown list.

  > **Note**
  >
  > Does not apply to single-instance brokers.

g. Choose **Add / update policy**.

10. Confirm that the new policy appears in the list of **User policies**.

> **Note**
>
> For cluster brokers, Amazon MQ automatically applies the `ha-mode: all` and `ha-sync-mode: automatic` policy definitions.

11. From the right navigation pane, choose **Limits**.
12. On the **Limits** page, you can see a list of the broker's current **Virtual host limits**. Below **Virtual host limits**, expand **Set / update a virtual host limit**.
13. To create a new vhost limit, under **Set / update a virtual host limit**, do the following:

    a. For **Virtual host**, choose the name of the vhost to which you want to attach the policies from the dropdown list. To choose the default vhost, choose **/**.

    b. For **Limit**, choose **max-connections** from the dropdown options.

    c. For **Value**, enter the Amazon MQ recommended value according to the broker's instance size and deployment mode, for example, **`15000`** for an `mq.m5.large` cluster.

    d. Choose **Set / update limit**.

    e. Repeat the steps above, and for **Limit**, choose **max-queues** from the dropdown options.

14. Confirm that the new limits appear in the list of **Virtual host limits**.

To apply default policies and virtual host limits using the RabbitMQ management API

1. Sign in to the Amazon MQ console.
2. In the left navigation pane, choose **Brokers**.

3. From the list of brokers, choose the name of the broker to which you want to apply the new policy.
4. On the broker's page, in the **Connections** section, note the **RabbitMQ web console** URL. This is the broker endpoint that you use in an HTTP request.
5. Open a new terminal or command line window of your choice.
6. To create a new broker policy, enter the following `curl` command. This command assumes a queue on the default `/` vhost, which is encoded as `%2F`. To apply the policy to another vhost, replace `%2F` with the vhost's name.

   > **Note**
   >
   > Replace *`username`* and *`password`* with your administrator sign-in credentials. Replace *`number-of-messages`* with the Amazon MQ recommended value according to the broker's instance size and deployment mode. Replace *`policy-name`* with a name for your policy. Replace *`broker-endpoint`* with the URL that you noted previously.

   ```
   curl -i -u username:password -H "content-type:application/json" -XPUT \
   -d '{"pattern":".*", "priority":1, "definition":{"queue-mode":lazy,
    "overflow":"reject-publish", "max-length":"number-of-messages"}}' \
   broker-endpoint/api/policies/%2F/policy-name
   ```

7. To confirm that the new policy is added to your broker's user policies, enter the following `curl` command to list all broker policies.

   ```
   curl -i -u username:password broker-endpoint/api/policies
   ```

8. To create a new `max-connections` virtual host limits, enter the following `curl` command. This command assumes a queue on the default `/` vhost, which is encoded as `%2F`. To apply the policy to another vhost, replace `%2F` with the vhost's name.

   > **Note**
   >
   > Replace *`username`* and *`password`* with your administrator sign-in credentials. Replace *`max-connections`* with the Amazon MQ recommended value according to the

broker's instance size and deployment mode. Replace the broker endpoint with the URL that you noted previously.

```
curl -i -u username:password -H "content-type:application/json" -XPUT \
-d '{"value":"number-of-connections"}' \
broker-endpoint/api/vhost-limits/%2F/max-connections
```

9. To create a new `max-queues` virtual host limit, repeat the previous step, but modify the curl command as shown in the following.

```
curl -i -u username:password -H "content-type:application/json" -XPUT \
-d '{"value":"number-of-queues"}' \
broker-endpoint/api/vhost-limits/%2F/max-queues
```

10. To confirm that the new limits are added to your broker's virtual host limits, enter the following `curl` command to list all broker virtual host limits.

```
curl -i -u username:password broker-endpoint/api/vhost-limits
```

Quorum queues for RabbitMQ on Amazon MQ

Quorum queues are a replicated queue type made up of a leader (primary replica) and followers (other replicas). If the leader becomes unavailable, quorum queues uses the Raft consensus algorithm to elect a new leader node by majority of votes, and the previous leader is demoted to a follower node in the same cluster. The remaining followers continue replicating as before. Because each node is in a different availability zone, if one node is temporarily unavailable, message delivery continues with the newly elected leader replica in another availability zone.

Quorum queues are useful for handling poison messages, which occur when a message fails and is requeued multiple times.

You should not use quorum queues if you:

- use transient queues
- have long queue backlogs

- prioritize low latency

To declare a quorum queue, set the header `x-queue-type` to `quorum`.

Topics

Migrating from classic queues to quorum queues on Amazon MQ for RabbitMQ

You can migrate your classic mirrored queues to quorum queues on Amazon MQ brokers on version 3.13 or above by creating a new virtual host on the same cluster, or by migrating in place.

Option 1: Migrating from classic mirrored queues to quorum queues with a new virtual host

You can migrate your classic mirrored queues to quorum queues on Amazon MQ brokers on version 3.13 or above by creating a new virtual host on the same cluster.

1. In your existing cluster, create a new virtual host (vhost) with the default queue type as quorum.
2. Create the Federation plugin from the new vhost with the URI pointing to the old vhost using classic mirrored queues.
3. Using `rabbitmqadmin`, export the definitions from the old vhost to a new file. You must make changes to the schema file so it is compatible with quorum queues. For the full list of changes you need to make to the file, see Moving definitions in the RabbitMQ quorum queues documentation. After applying the necessary changes to the file, reimport the definitions to the new vhost.
4. Create a new policy in the new vhost. For recommendations on Amazon MQ policy configurations for quorum queues, see Policy configurations for quorum queues for Amazon MQ for RabbitMQ . Then, start the Federation you created earlier from the old vhost to the new vhost.
5. Point consumers and producers to the new vhost.

6. Configure the Shovel plug in to move any remaining messages. Once a queue is empty, delete the Shovel.

Migrating from classic mirrored queues to quorum queues in place

You can migrate your classic mirrored queues to quorum queues on Amazon MQ brokers on version 3.13 or above by migrating in place.

1. Stop the consumers and producers.
2. Create a new temporary quorum queue.
3. Configure the Shovel plug in to move any messages from the old classic mirrored queue to the new temporary quorum queue. After all messages are moved to the temporary quorum queue, delete the Shovel.
4. Delete the source classic mirrored queue. Then, recreate a quorum queue with the same name and bindings as the source classic mirrored queue.
5. Create a new Shovel to move the messages from the temporary quorum queue to the new quorum queue.

Policy configurations for quorum queues for Amazon MQ for RabbitMQ

You can add specific policy configurations to quorum queues for your RabbitMQ broker on Amazon MQ.

When you create a policy for quorum queues, you must do the following:

- Remove all policy attributes that start with `ha`, such as `ha-mode`, `ha-params`, `ha-sync-mode`, `ha-sync-batch-size`, `ha-promote-on-shutdown`, and `ha-promote-on-failure`.
- Remove `queue-mode`.
- Change overflow when it is set to `reject-publish-dlx`

> **Important**
>
> Amazon MQ for RabbitMQ applies all or none of the attributes within a policy. You cannot create a policy that applies to both classic mirrored queues and quorum queues. If you want your policy to only apply to quorum queues, you must set `--apply-to` to

`quorum_queues`. If you are using classic mirrored queues and quorum queues, you must create a separate policy with `--apply-to:classic_queues` as well as a quorum queues policy.

You do not need to modify `AWS-DEFAULT` policies because they automatically adopt the new queue type in the "applies to" parameter. For more information on default policies for Amazon MQ for RabbitMQ, see Configuring operator policies.

Best practices for quorum queues for Amazon MQ for RabbitMQ

We recommend using the following best practices to improve performance when working with quorum queues.

Handling poison messages by setting a delivery limit

Poison messages occur when a message fails and is redelivered multiple times. You can set a message delivery limit using the `delivery-limit` policy argument to drop messages that are redelivered multiple times. If a message is redelivered more times than the delivery limit allows, the message is then dropped and deleted by RabbitMQ. When you set a delivery limit, the message is requeued near the head of the queue.

Message priority for quorum queues

Quorum queues do not have message priority. If you need message priority, you must create multiple quorum queues. For more information on prioritizing messages with multiple quorum queues, see Message priority in the RabbitMQ documentation.

Using the default replication factor

Amazon MQ for RabbitMQ defaults to a replication factor of three (3) nodes for cluster brokers using quorum queues. If you make changes to `x-quorum-initial-group-size`, Amazon MQ will default again to the replication factor of 3.

Amazon MQ for RabbitMQ best practices

Follow these production-readiness guidelines to maximize broker performance and optimize message throughput efficiency when working with Amazon MQ for RabbitMQ brokers.

> **Important**
>
> Currently, Amazon MQ does not support streams, or using structured logging in JSON, introduced in RabbitMQ 3.9.x.

Topics

Best practices for broker setup and connection management in Amazon MQ for RabbitMQ

Broker setup and connection management are the first step in preventing issues with broker message throughput, resource utilization, and ability to handle production workloads. When creating and configuring an Amazon MQ for RabbitMQ broker, complete the following best practices for selecting appropriate instance types, managing connections efficiently, and configuring message pre-fetching to maximize your broker's performance.

> **Important**
>
> Amazon MQ for RabbitMQ does not support the username "guest", and will delete the default guest account when you create a new broker. Amazon MQ will also periodically delete any customer created account called "guest".

Step 1: Use cluster deployments

For production workloads, we recommend using cluster deployments instead of single-instance brokers to ensure high availability and message resiliency. Cluster deployments remove single points of failure and provide better fault tolerance.

Cluster deployments consist of three RabbitMQ broker nodes distributed across three Availability Zones, providing automatic failover and ensuring operations continue even if an entire Availability

Zone becomes unavailable. Amazon MQ automatically replicates messages across all nodes to ensure availability during node failures or maintenance.

Cluster deployments are essential for production environments and are supported by the Amazon MQ Service Level Agreement.

For more information, see Cluster deployment in Amazon MQ for RabbitMQ.

Step 2: Choose the correct broker instance type

The message throughput of a broker instance type depends on your application use case. `M7g.medium` should only be used for testing application performance. Using this smaller instance before using larger instances in production can improve application performance. On instance types `m7g.large` and above, you can use cluster deployments for high availability and message durability. Larger broker instance types can handle production levels of clients and queues, high throughput, messages in memory, and redundant messages.

For more information on choosing the correct instance type, see Sizing guidelines in Amazon MQ for RabbitMQ.

Step 3: Use quorum queues

Quorum queues, with cluster deployment, should be the default choice for replicated queue types in production environments for RabbitMQ brokers on 3.13 and above. Quorum queues are a modern replicated queue type that provide high reliability, high throughput, and stable latency.

Quorum queues use the Raft consensus algorithm to provide better fault tolerance. When the leader node becomes unavailable, quorum queues automatically elect a new leader by majority vote, ensuring message delivery continues with minimal disruption. Since each node is in a different Availability Zone, your messaging system remains available even if an entire Availability Zone becomes temporarily unavailable.

To declare a quorum queue, set the header `x-queue-type` to `quorum` when creating your queues.

For more information on quorum queues, including migration strategies and best practices, see Quorum queues in Amazon MQ for RabbitMQ.

Step 4: Use multiple channels

To avoid connection churn, use multiple channels over a single connection. Applications should avoid a 1:1 connection to channel ratio. We recommend using one connection for each process, and then one channel for each thread. Avoid excessive channel usage to prevent channel leaks.

Best practices for message durability and reliability in Amazon MQ for RabbitMQ

Before moving your application to production, complete the following best practices for preventing message loss and resource overutilization.

Step 1: Use persistent messages and durable queues

Persistent messages can help protect data durability in situations where a broker crashes or restarts. Persistent messages are written to disk as soon as they arrive. Unlike lazy queues, however, persistent messages are cached both in memory and in disk unless more memory is needed by the broker. In cases where more memory is needed, messages are removed from memory by the RabbitMQ broker mechanism that manages storing messages to disk, commonly referred to as the *persistence layer*.

To enable message persistence, you can declare your queues as `durable` and set message delivery mode to `persistent`. The following example demonstrates using the RabbitMQ Java client library to declare a durable queue. When working with AMQP 0-9-1, you can mark messages as persistent by setting delivery mode "2".

```
boolean durable = true;
channel.queueDeclare("my_queue", durable, false, false, null);
```

Once you have configured your queue as durable, you can send a persistent message to your queue by setting `MessageProperties` to `PERSISTENT_TEXT_PLAIN` as shown in the following example.

```
import com.rabbitmq.client.MessageProperties;

channel.basicPublish("", "my_queue",
            MessageProperties.PERSISTENT_TEXT_PLAIN,
            message.getBytes());
```

Step 2: Configure publisher confirms and consumer delivery acknowledgement

The process of confirming a message has been sent to the broker is known as *publisher confirmation*. Publisher confirms let your application know when messages have been reliably stored. Publisher confirms can also help control the rate of messages stored to the broker. Without

publisher confirms, there is no confirmation that a message is processed successfully, and your broker may drop messages it cannot process.

Similarly, when a client application sends confirmation of delivery and consumption of messages back to the broker, it is known as *consumer delivery acknowledgment*. Both confirmation and acknowledgement are essential to ensuring data safety when working with RabbitMQ brokers.

Consumer delivery acknowledgement is typically configured on the client application. When working with AMQP 0-9-1, acknowledgement can be enabled by configuring the `basic.consume` method. AMQP 0-9-1 clients can also configure publisher confirms by sending the `confirm.select` method.

Typically, delivery acknowledgement is enabled in a channel. For example, when working with the RabbitMQ Java client library, you can use the `Channel#basicAck` to set up a simple `basic.ack` positive acknowledgement as shown in the following example.

```
// this example assumes an existing channel instance

boolean autoAck = false;
channel.basicConsume(queueName, autoAck, "a-consumer-tag",
     new DefaultConsumer(channel) {
         @Override
         public void handleDelivery(String consumerTag,
                                    Envelope envelope,
                                    AMQP.BasicProperties properties,
                                    byte[] body)
             throws IOException
         {
             long deliveryTag = envelope.getDeliveryTag();
             // positively acknowledge a single delivery, the message will
             // be discarded
             channel.basicAck(deliveryTag, false);
         }
     });
```

Note

Unacknowledged messages must be cached in memory. You can limit the number of messages that a consumer pre-fetches by configuring pre-fetch settings for a client application.

You can configure `consumer_timeout` to detect when consumers do not acknowledge deliveries. If the consumer does not send an acknowledgment within the timeout value, the channel will be closed, and you will recieve a `PRECONDITION_FAILED`. To diagnose the error, use the UpdateConfiguration API to increase the `consumer_timeout` value.

Step 3: Keep queues short

In cluster deployments, queues with a large number of messages can lead to resource overutilization. When a broker is overutilized, rebooting an Amazon MQ for RabbitMQ broker can cause further degradation of performance. If rebooted, overutilized brokers might become unresponsive in the `REBOOT_IN_PROGRESS` state.

During maintenance windows, Amazon MQ performs all maintenance work one node at a time to ensure that the broker remains operational. As a result, queues might need to synchronize as each node resumes operation. During synchronization, messages that need to be replicated to mirrors are loaded into memory from the corresponding Amazon Elastic Block Store (Amazon EBS) volume to be processed in batches. Processing messages in batches lets queues synchronize faster.

If queues are kept short and messages are small, the queues successfully synchronize and resume operation as expected. However, if the amount of data in a batch approaches the node's memory limit, the node raises a high memory alarm, pausing the queue sync. You can confirm memory usage by comparing the `RabbitMemUsed` and `RabbitMqMemLimit` broker node metrics in CloudWatch. Synchronization can't complete until messages are consumed or deleted, or the number of messages in the batch is reduced.

If queue synchronization is paused for a cluster deployment, we recommend consuming or deleting messages to lower the number of messages in queues. Once queue depth is reduced and queue sync completes, the broker status will change to `RUNNING`. To resolve a paused queue sync, you can also apply a policy to reduce the queue synchronization batch-size.

You can also define auto-delete and TTL policies to proactively reduce resource usage, as well as keep NACKs from consumers to a minimum. Requeueing messages on the broker is CPU-intensive so a high number of NACKs can affect broker performance.

Best practices for performance optimization and efficiency in Amazon MQ for RabbitMQ

You can optimize your Amazon MQ for RabbitMQ broker performance by maximizing throughput, minimizing latency, and ensuring efficient resource utilization. Complete the following best practices to optimize your application performance.

Step 1: Keep message sizes under 1 MB

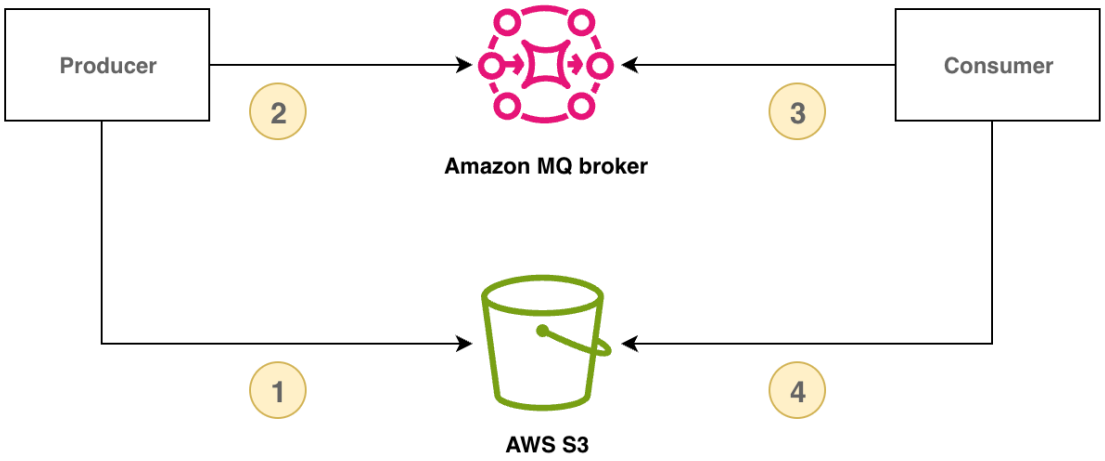
We recommend keeping messages under 1 Megabyte (MB) for optimal performance and reliability.

RabbitMQ 3.13 supports message sizes up to 128 MB by default, but large messages may trigger unpredictable memory alarms that block publishing and potentially create high memory pressure while replicating messages across nodes. Oversized messages can also affect broker restart and recovery processes, which increases risks to service continuity and may cause performance degradation.

Store and retrieve large payloads using the claim check pattern

To manage large messages, you can implement the claim check pattern by storing the message payload in external storage and sending only the payload reference identifier through RabbitMQ. The consumer uses the payload reference identifier to retrieve and process the large message.

The following diagram demonstrates how to use Amazon MQ for RabbitMQ and Amazon S3 to implement the claim check pattern.



The following example demonstrates this pattern using Amazon MQ, the AWS SDK for Java 2.x, and Amazon S3:

1. First, define a Message class that will hold the Amazon S3 reference identifier.

```
class Message {
    // Other data fields of the message...

    public String s3Key;
    public String s3Bucket;
}
```

2. Create a publisher method that stores the payload in Amazon S3 and sends a reference message through RabbitMQ.

```
public void publishPayload() {
    // Store the payload in S3.
    String payload = PAYLOAD;
    String prefix = S3_KEY_PREFIX;
    String s3Key = prefix + "/" + UUID.randomUUID();
    s3Client.putObject(PutObjectRequest.builder()
        .bucket(S3_BUCKET).key(s3Key).build(),
        RequestBody.fromString(payload));

    // Send the reference through RabbitMQ.
    Message message = new Message();
    message.s3Key = s3Key;
    message.s3Bucket = S3_BUCKET;
    // Assign values to other fields in your message instance.

    publishMessage(message);
}
```

3. Implement a consumer method that retrieves the payload from Amazon S3, processes the payload, and deletes the Amazon S3 object.

```
public void consumeMessage(Message message) {
    // Retrieve the payload from S3.
    String payload = s3Client.getObjectAsBytes(GetObjectRequest.builder()
        .bucket(message.s3Bucket).key(message.s3Key).build())
        .asUtf8String();

    // Process the complete message.
    processPayload(message, payload);

    // Delete the S3 object.
```

```
    s3Client.deleteObject(DeleteObjectRequest.builder()
        .bucket(message.s3Bucket).key(message.s3Key).build());
}
```

Step 2: Use `basic.consume` and long-lived consumers

Using `basic.consume` with a long-lived consumer is more efficient than polling for individual messages using `basic.get`. For more information, see Polling for individual messages.

Step 3: Configure pre-fetching

You can use the RabbitMQ pre-fetch value to optimize how your consumers consume messages. RabbitMQ implements the channel pre-fetch mechanism provided by AMQP 0-9-1 by applying the pre-fetch count to consumers as opposed to channels. The pre-fetch value is used to specify how many messages are being sent to the consumer at any given time. By default, RabbitMQ sets an unlimited buffer size for client applications.

There are a variety of factors to consider when setting a pre-fetch count for your RabbitMQ consumers. First, consider your consumers' environment and configuration. Because consumers need to keep all messages in memory as they are being processed, a high pre-fetch value can have a negative impact on your consumers' performance, and in some cases, can result in a consumer potentially crashing all together. Similarly, the RabbitMQ broker itself keeps all messages that it sends cached in memory until it recieves consumer acknowledgement. A high pre-fetch value can cause your RabbitMQ server to run out of memory quickly if automatic acknowledgement is not configured for consumers, and if consumers take a relatively long time to process messages.

With the above considerations in mind, we recommend always setting a pre-fetch value in order to prevent situations where a RabbitMQ broker or its consumers run out of memory due to a large number number of unprocessed, or unacknowledged messages. If you need to optimize your brokers to process large volumes of messages, you can test your brokers and consumers using a range of pre-fetch counts to determine the value at which point network overhead becomes largely insignificant compared to the time it takes a consumer to process messages.

Note

- If your client applications have configured to automatically acknowledge delivery of messages to consumers, setting a pre-fetch value will have no effect.

- All pre-fetched messages are removed from the queue.

The following example desmonstrate setting a pre-fetch value of `10` for a single consumer using the RabbitMQ Java client library.

```
ConnectionFactory factory = new ConnectionFactory();

Connection connection = factory.newConnection();
Channel channel = connection.createChannel();

channel.basicQos(10, false);

QueueingConsumer consumer = new QueueingConsumer(channel);
channel.basicConsume("my_queue", false, consumer);
```

Note

In the RabbitMQ Java client library, the default value for the `global` flag is set to `false`, so the above example can be written simply as `channel.basicQos(10)`.

Step 4: Use Celery 5.5 or later with quorum queues

Python Celery, a distributed task queue system, can generate many non-critical messages when experiencing high task load. This additional broker activity can trigger the section called "`RABBITMQ_MEMORY_ALARM`" and lead to broker unavailability. To reduce the chance of triggering memory alarm, do the following:

For all Celery versions

1. Turn off `task_create_missing_queues` to mitigate queue churn.
2. Then, turn off `worker_enable_remote_control` to stop dynamic creation of `celery@...pidbox` queues. This will reduce queue churn on the broker.

   ```
   worker_enable_remote_control = false
   ```
3. To further reduce non-critical message activity, turn off Celery worker-send-task-events by not including `-E` or `--task-events` flag when starting your Celery application.

4. Start your Celery application using the following parameters:

```
celery -A app_name worker --without-heartbeat --without-gossip --without-mingle
```

For Celery versions 5.5 and above

1. Upgrade to Celery version 5.5, the minimum version that supports quorum queues, or a later version. To check what version of Celery you are using, use `celery --version`. For more information on quorum queues, see the section called “Quorum queues”.
2. After upgrading to Celery 5.5 or later, configure `task_default_queue_type` to "quorum".
3. Then, you must also turn on Publish Confirms in Broker Transport Options:

```
broker_transport_options = {"confirm_publish": True}
```

Best practices for network resilience and monitoring in Amazon MQ for RabbitMQ

Network resilience and monitoring broker metrics are essential for maintaining reliable messaging applications. Complete the following best practices to implement automatic recovery mechanisms and resource monitoring strategies.

Step 1: Automatically recover from network failures

We recommend always enabling automatic network recovery to prevent significant downtime in cases where client connections to RabbitMQ nodes fail. The RabbitMQ Java client library supports automatic network recovery by default, beginning with version `4.0.0`.

Automatic connection recovery is triggered if an unhandled exception is thrown in the connection's I/O loop, if a socket read operation timeout is detected, or if the server misses a heartbeat.

In cases where the initial connection between a client and a RabbitMQ node fails, automatic recovery will not be triggered. We recommend writing your application code to account for initial connection failures by retrying the connection. The following example demonstrates retrying initial network failures using the RabbitMQ Java client library.

```
ConnectionFactory factory = new ConnectionFactory();
```

```
// enable automatic recovery if using RabbitMQ Java client library prior to version
 4.0.0.
factory.setAutomaticRecoveryEnabled(true);
// configure various connection settings

try {
  Connection conn = factory.newConnection();
} catch (java.net.ConnectException e) {
  Thread.sleep(5000);
  // apply retry logic
}
```

> **Note**
>
> If an application closes a connection by using the `Connection.Close` method, automatic network recovery will not be enabled or triggered.

Step 2: Monitor broker metrics and alarms

We recommend regularly monitoring CloudWatch metrics and alarms for your Amazon MQ for RabbitMQ broker to identify and address potential issues before they impact your messaging application. Proactive monitoring is essential for maintaining a resilient messaging application and ensuring optimal performance.

Amazon MQ for RabbitMQ publishes metrics to CloudWatch that provide insights into broker performance, resource utilization, and message flow. Key metrics to monitor include memory usage and disk usage. You can set up CloudWatch alarms for when your broker approaches resource limits or experiences performance degradation.

Monitor the following essential metrics:

`RabbitMQMemUsed` and `RabbitMQMemLimit`

Monitor memory usage to prevent memory alarms that can block message publishing.

`RabbitMQDiskFree` and `RabbitMQDiskFreeLimit`

Monitor disk usage to avoid disk space issues that can cause broker failures.

For cluster deployments, also monitor node-specific metrics to identify node-specific issues.

> **Note**
>
> For more information about how to prevent high memory alarm, see Address and prevent high memory alarm.

RabbitMQ tutorials

The following tutorials show how you can configure and use RabbitMQ on Amazon MQ. To learn more about working with supported client libraries in a variety of programming languages such as Node.js, Python, .NET, and more, see RabbitMQ Tutorials in the *RabbitMQ Getting Started Guide*.

Topics

Editing broker preferences

You can edit your broker preferences, such as enabling or disabling CloudWatch logs using the AWS Management Console.

Edit RabbitMQ broker options

1. Sign in to the Amazon MQ console.

2. From the broker list, select your broker (for example, **MyBroker**) and then choose **Edit**.
3. On the **Edit *MyBroker*** page, in the **Specifications** section, select a **Broker engine version** or a **Broker Instance type**.
4. In the **CloudWatch Logs** section, click the toggle button to enable or disable general logs. No other steps are required.

   > **Note**
   >
   > - For RabbitMQ brokers, Amazon MQ automatically uses a Service-Linked Role (SLR) to publish general logs to CloudWatch. For more information, see the section called "Using service-linked roles"
   > - Amazon MQ does not support audit logging for RabbitMQ brokers.

5. In the **Maintenance** section, configure your broker's maintenance schedule:

   To upgrade the broker to new versions as AWS releases them, choose **Enable automatic minor version upgrades**. Automatic upgrades occur during the *maintenance window* defined by the day of the week, the time of day (in 24-hour format), and the time zone (UTC by default).
6. Choose **Schedule modifications**.

   > **Note**
   >
   > If you choose only **Enable automatic minor version upgrades**, the button changes to **Save** because no broker reboot is necessary.

   Your preferences are applied to your broker at the specified time.

Using Python Pika with Amazon MQ for RabbitMQ

The following tutorial shows how you can set up a Python Pika client with TLS configured to connect to an Amazon MQ for RabbitMQ broker. Pika is a Python implementation of the AMQP 0-9-1 protocol for RabbitMQ. This tutorial guides you through installing Pika, declaring a queue, setting up a publisher to send messages to the broker's default exchange, and setting up a consumer to receive messages from the queue.

Topics

Prerequisites

To complete the steps in this tutorial, you need the following prerequisites:

- An Amazon MQ for RabbitMQ broker. For more information, see Creating an Amazon MQ for RabbitMQ broker.
- Python 3 installed for your operating system.
- Pika installed using Python `pip`. To install Pika, open a new terminal window and run the following.

```
$ python3 -m pip install pika
```

Permissions

For this tutorial, you need at least one Amazon MQ for RabbitMQ broker user with permission to write to, and read from, a vhost. The following table describes the necessary minimum permissions as regular expression (regexp) patterns.

Tags	Configure regexp	Write regexp	Read regexp
none		.*	.*

The user permissions listed provide only read and write permissions to the user, without granting access to the management plugin to perform administrative operations on the broker. You can further restrict permissions by providing regexp patterns that limit the user's access to specified

queues. For example, if you change the read regexp pattern to `^[hello world].*`, the user will only have permission to read from queues that start with `hello world`.

For more information about creating RabbitMQ users and managing user tags and permissions, see Amazon MQ for RabbitMQ broker users.

Step one: Create a basic Python Pika client

To create a Python Pika client base class that defines a constructor and provides the SSL context necessary for TLS configuration when interacting with an Amazon MQ for RabbitMQ broker, do the following.

1. Open a new terminal window, create a new directory for your project, and navigate to the directory.

```
$ mkdir pika-tutorial
$ cd pika-tutorial
```

2. Create a new file, `basicClient.py`, that contains the following Python code.

```
import ssl
import pika

class BasicPikaClient:

    def __init__(self, rabbitmq_broker_id, rabbitmq_user, rabbitmq_password,
 region):

        # SSL Context for TLS configuration of Amazon MQ for RabbitMQ
        ssl_context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2)
        ssl_context.set_ciphers('ECDHE+AESGCM:!ECDSA')

        url = f"amqps://{rabbitmq_user}:
{rabbitmq_password}@{rabbitmq_broker_id}.mq.{region}.amazonaws.com:5671"
        parameters = pika.URLParameters(url)
        parameters.ssl_options = pika.SSLOptions(context=ssl_context)

        self.connection = pika.BlockingConnection(parameters)
        self.channel = self.connection.channel()
```

You can now define additional classes for your publisher and consumer that inherit from `BasicPikaClient`.

Step two: Create a publisher and send a message

To create a publisher that declares a queue, and sends a single message, do the following.

1. Copy the contents of the following code sample, and save locally as `publisher.py` in the same directory you created in the previous step.

```
from basicClient import BasicPikaClient

class BasicMessageSender(BasicPikaClient):

    def declare_queue(self, queue_name):
        print(f"Trying to declare queue({queue_name})...")
        self.channel.queue_declare(queue=queue_name)

    def send_message(self, exchange, routing_key, body):
        channel = self.connection.channel()
        channel.basic_publish(exchange=exchange,
                              routing_key=routing_key,
                              body=body)
        print(f"Sent message. Exchange: {exchange}, Routing Key: {routing_key},
 Body: {body}")

    def close(self):
        self.channel.close()
        self.connection.close()

if __name__ == "__main__":

    # Initialize Basic Message Sender which creates a connection
    # and channel for sending messages.
    basic_message_sender = BasicMessageSender(
        "<broker-id>",
        "<username>",
        "<password>",
        "<region>"
    )

    # Declare a queue
    basic_message_sender.declare_queue("hello world queue")
```

```
    # Send a message to the queue.
    basic_message_sender.send_message(exchange="", routing_key="hello world queue",
body=b'Hello World!')

    # Close connections.
    basic_message_sender.close()
```

The `BasicMessageSender` class inherits from `BasicPikaClient` and implements additional methods for delaring a queue, sending a message to the queue, and closing connections. The code sample routes a message to the default exchange, with a routing key equal to the name of the queue.

2. Under `if __name__ == "__main__":`, replace the parameters passed to the `BasicMessageSender` constructor statement with the following information.

   - **<broker-id>** – The unique ID that Amazon MQ generates for the broker. You can parse the ID from your broker ARN. For example, given the following ARN, `arn:aws:mq:us-east-2:123456789012:broker:MyBroker:b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9`, the broker ID would be `b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9`.
   - **<username>** – The username for a broker user with sufficient permissions to write messages to the broker.
   - **<password>** – The password for a broker user with sufficient permissions to write messages to the broker.
   - **<region>** – The AWS region in which you created your Amazon MQ for RabbitMQ broker. For example, `us-west-2`.

3. Run the following command in the same directory you created `publisher.py`.

```
$ python3 publisher.py
```

If the code runs successfully, you will see the following output in your terminal window.

```
Trying to declare queue(hello world queue)...
Sent message. Exchange: , Routing Key: hello world queue, Body: b'Hello World!'
```

Step three: Create a consumer and receive a message

To create a consumer that receives a single message from the queue, do the following.

1. Copy the contents of the following code sample, and save locally as `consumer.py` in the same directory.

```
from basicClient import BasicPikaClient

class BasicMessageReceiver(BasicPikaClient):

    def get_message(self, queue):
        method_frame, header_frame, body = self.channel.basic_get(queue)
        if method_frame:
            print(method_frame, header_frame, body)
            self.channel.basic_ack(method_frame.delivery_tag)
            return method_frame, header_frame, body
        else:
            print('No message returned')

    def close(self):
        self.channel.close()
        self.connection.close()


if __name__ == "__main__":

    # Create Basic Message Receiver which creates a connection
    # and channel for consuming messages.
    basic_message_receiver = BasicMessageReceiver(
        "<broker-id>",
        "<username>",
        "<password>",
        "<region>"
    )

    # Consume the message that was sent.
    basic_message_receiver.get_message("hello world queue")

    # Close connections.
    basic_message_receiver.close()
```

Similar to the publisher you created in the previous step, `BasicMessageReceiver` inherits from `BasicPikaClient` and implements additional methods for recieving a single message, and closing connections.

2. Under the `if __name__ == "__main__":` statement, replace the parameters passed to the `BasicMessageReceiver` constructor with your information.
3. Run the following command in your project directory.

```
$ python3 consumer.py
```

If the code runs successfully, you will see the message body, and headers including the routing key, displayed in your terminal window.

```
<Basic.GetOk(['delivery_tag=1', 'exchange=', 'message_count=0',
 'redelivered=False', 'routing_key=hello world queue'])> <BasicProperties> b'Hello
 World!'
```

Step four: (Optional) Set up an event loop and consume messages

To consume multiple messages from a queue, use Pika's [basic_consume] method and a callback function as shown in the following

1. In `consumer.py`, add the following method definition to the `BasicMessageReceiver` class.

```
def consume_messages(self, queue):
    def callback(ch, method, properties, body):
        print(" [x] Received %r" % body)

    self.channel.basic_consume(queue=queue, on_message_callback=callback,
 auto_ack=True)

    print(' [*] Waiting for messages. To exit press CTRL+C')
    self.channel.start_consuming()
```

2. In `consumer.py`, under `if __name__ == "__main__":`, invoke the `consume_messages` method you defined in the previous step.

```
if __name__ == "__main__":

    # Create Basic Message Receiver which creates a connection and channel for
 consuming messages.
    basic_message_receiver = BasicMessageReceiver(
        "<broker-id>",
```

```
        "<username>",
        "<password>",
        "<region>"
    )

    # Consume the message that was sent.
    # basic_message_receiver.get_message("hello world queue")

    # Consume multiple messages in an event loop.
    basic_message_receiver.consume_messages("hello world queue")

    # Close connections.
    basic_message_receiver.close()
```

3. Run `consumer.py` again, and if successful, the queued messages will be displayed in your terminal window.

```
[*] Waiting for messages. To exit press CTRL+C
[x] Received b'Hello World!'
[x] Received b'Hello World!'
...
```

What's next?

- For more information about other supported RabbitMQ client libraries, see RabbitMQ Client Documentation on the RabbitMQ website.

Resolving RabbitMQ paused queue synchronization

In an Amazon MQ for RabbitMQ cluster deployment, messages published to each queue are replicated across three broker nodes. This replication, referred to as *mirroring*, provides high availability (HA) for RabbitMQ brokers. Queues in a cluster deployment consist of a *main* replica on one node and one or more *mirrors*. Every operation applied to a mirrored queue, including enqueuing messages, is first applied to the main queue and then replicated across its mirrors.

For example, consider a mirrored queue replicated across three nodes: the main node (`main`) and two mirrors (`mirror-1` and `mirror-2`). If all messages in this mirrored queue are successfully propagated to all mirrors, then the queue is synchronized. If a node (`mirror-1`) becomes unavailable for an interval of time, the queue is still operational and can continue to enqueue

messages. However, for the queue to synchronize, messages published to `main` while `mirror-1` is unavailable must be replicated to `mirror-1`.

For more information about mirroring, see Classic Mirrored Queues on the RabbitMQ website.

Maintenance and queue synchronization

During maintenance windows, Amazon MQ performs all maintenance work one node at a time to ensure that the broker remains operational. As a result, queues might need to synchronize as each node resumes operation. During synchronization, messages that need to be replicated to mirrors are loaded into memory from the corresponding Amazon Elastic Block Store (Amazon EBS) volume to be processed in batches. Processing messages in batches lets queues synchronize faster.

If queues are kept short and messages are small, the queues successfully synchronize and resume operation as expected. However, if the amount of data in a batch approaches the node's memory limit, the node raises a high memory alarm, pausing the queue sync. You can confirm memory usage by comparing the `RabbitMemUsed` and `RabbitMqMemLimit` broker node metrics in CloudWatch. Synchronization can't complete until messages are consumed or deleted, or the number of messages in the batch is reduced.

> **Note**
>
> Reducing the queue synchronization batch size can result in a higher number of replication transactions.

To resolve a paused queue synchronization, follow the steps in this tutorial, which demonstrates applying an `ha-sync-batch-size` policy and restarting the queue sync.

Topics

Prerequisites

For this tutorial, you must have an Amazon MQ for RabbitMQ broker user with administrator permissions. You can use the administrator user created when you first created the broker, or another user that you might have created afterwards. The following table provides the necessary administrator user tag and permissions as regular expression (regexp) patterns.

Tags	Read regexp	Configure regexp	Write regexp
`administrator`	`.*`	`.*`	`.*`

For more information about creating RabbitMQ users and managing user tags and permissions, see Amazon MQ for RabbitMQ broker users.

Step 1: Apply an `ha-sync-batch-size` policy

The following procedures demonstrate adding a policy that applies to all queues created on the broker. You can use the RabbitMQ web console or the RabbitMQ management API. For more information, see Management Plugin on the RabbitMQ website.

To apply an `ha-sync-batch-size` policy using the RabbitMQ web console

1. Sign in to the Amazon MQ console.
2. In the left navigation pane, choose **Brokers**.
3. From the list of brokers, choose the name of the broker to which you want to apply the new policy.
4. On the broker's page, in the **Connections** section, choose the **RabbitMQ web console** URL. The RabbitMQ web console opens in a new browser tab or window.
5. Log in to the RabbitMQ web console with your broker administrator sign-in credentials.
6. In the RabbitMQ web console, at the top of the page, choose **Admin**.
7. On the **Admin** page, in the right navigation pane, choose **Policies**.
8. On the **Policies** page, you can see a list of the broker's current **User policies**. Below **User policies**, expand **Add / update a policy**.

> **Note**
>
> By default, Amazon MQ for RabbitMQ clusters are created with an initial broker policy named `ha-all-AWS-OWNED-DO-NOT-DELETE`. Amazon MQ manages this policy to ensure that every queue on the broker is replicated to all three nodes and that queues are synchronized automatically.

9. To create a new broker policy, under **Add / update a policy**, do the following:

   a. For **Name**, enter a name for your policy, for example, **`batch-size-policy`**.

   b. For **Pattern**, enter the regexp pattern **`.*`** so that the policy matches all queues on the broker.

   c. For **Apply to**, choose **Exchanges and queues** from the dropdown list.

   d. For **Priority**, enter an integer greater than all other policies in applied to the vhost. You can apply exactly one set of policy definitions to RabbitMQ queues and exchanges at any given time. RabbitMQ chooses the matching policy with the highest priority value. For more information about policy priorities and how to combine policies, see Policies in the RabbitMQ Server Documentation.

   e. For **Definition**, add the following key-value pairs:

   - **`ha-sync-batch-size`**=*`100`*. Choose **Number** from the dropdown list.

     > **Note**
     >
     > You might need to adjust and calibrate the value of `ha-sync-batch-size` based on the number and size of unsynchronized messages in your queues.

   - **`ha-mode`**=**`all`**. Choose **String** from the dropdown list.

     > **Important**
     >
     > The `ha-mode` definition is required for all HA-related policies. Omitting it results in a validation failure.

   - **`ha-sync-mode`**=**`automatic`**. Choose **String** from the dropdown list.

> **Note**
>
> The `ha-sync-mode` definition is required for all custom policies. If it is omitted, Amazon MQ automatically appends the definition.

f. Choose **Add / update policy**.

10. Confirm that the new policy appears in the list of **User policies**.

To apply an `ha-sync-batch-size` policy using the RabbitMQ management API

1. Sign in to the Amazon MQ console.
2. In the left navigation pane, choose **Brokers**.
3. From the list of brokers, choose the name of the broker to which you want to apply the new policy.
4. On the broker's page, in the **Connections** section, note the **RabbitMQ web console** URL. This is the broker endpoint that you use in an HTTP request.
5. Open a new terminal or command line window of your choice.
6. To create a new broker policy, enter the following `curl` command. This command assumes a queue on the default `/` vhost, which is encoded as `%2F`.

> **Note**
>
> Replace *`username`* and *`password`* with your broker administrator sign-in credentials. You might need to adjust and calibrate the value of `ha-sync-batch-size` (*`100`*) based on the number and size of unsynchronized messages in your queues. Replace the broker endpoint with the URL that you noted previously.

```
curl -i -u username:password -H "content-type:application/json" -XPUT \
-d '{"pattern":".*", "priority":1, "definition":{"ha-sync-batch-size":100, "ha-
mode":"all", "ha-sync-mode":"automatic"}}' \
https://b-589c045f-f8ln-4ab0-a89c-co62e1c32ef8.mq.us-west-2.amazonaws.com/api/
policies/%2F/batch-size-policy
```

7. To confirm that the new policy is added to your broker's user policies, enter the following `curl` command to list all broker policies.

```
curl -i -u username:password https://b-589c045f-f8ln-4ab0-a89c-co62e1c32ef8.mq.us-west-2.amazonaws.com/api/policies
```

Step 2: Restart the queue sync

After applying a new `ha-sync-batch-size` policy to your broker, restart the queue sync.

To restart the queue sync using the RabbitMQ web console

> **Note**
>
> To open the RabbitMQ web console, see the previous instructions in Step 1 of this tutorial.

1. In the RabbitMQ web console, at the top of the page, choose **Queues**.
2. On the **Queues** page, under **All queues**, locate your paused queue. In the **Policy** row, your queue should list the name of the new policy that you created (for example, `batch-size-policy`).
3. To restart the synchronization process with a reduced batch size, first cancel queue sync. Then restart the queue sync.

> **Note**
>
> If synchronization pauses and doesn't finish successfully, try reducing the `ha-sync-batch-size` value and restarting the queue sync again.

Next steps

- Once your queue synchronizes successfully, you can monitor the amount of memory that your RabbitMQ nodes use by viewing the Amazon CloudWatch metric `RabbitMQMemUsed`. You can also view the `RabbitMQMemLimit` metric to monitor a node's memory limit. For more information, see Accessing CloudWatch metrics for Amazon MQ and Available CloudWatch metrics for Amazon MQ for RabbitMQ brokers.

- To prevent paused queue synchronization, we recommend keeping queues short and processing messages. For workloads with larger message sizes, we also recommend upgrading your broker instance type to a larger instance size with more memory. For more information about broker instance types and editing broker preferences, see Editing broker preferences.
- When you create a new Amazon MQ for RabbitMQ broker, Amazon MQ applies a set of default policies and virtual host limits to optimize broker performance. If your broker does not have the recommended default policies and limits, we recommend creating them yourself. For more information about creating default policies and vhost limits, see https://docs.aws.amazon.com//amazon-mq/latest/developer-guide/rabbitmq-defaults.html.

Related resources

- UpdateBrokerInput – Use this broker property to update a broker instance type using the Amazon MQ API.
- Parameters and Policies (RabbitMQ Server Documentation) – Learn more about RabbitMQ parameters and policies on the RabbitMQ website.
- RabbitMQ Management HTTP API – Learn more about the RabbitMQ management API.

Reducing the number of connections and channels

Connections to your RabbitMQ on Amazon MQ broker can be closed either by your client applications, or by manually closing them using the RabbitMQ web console. To close a connection using the RabbitMQ web console, do the following:

1. Sign in to AWS Management Console and open your broker's RabbitMQ web console.
2. On the RabbitMQ console, choose the **Connections** tab.
3. On the **Connections** page, under **All connections**, choose the name of the connection you want to close from the list.
4. On the connection details page, choose **Close this connection** to expand the section, then choose **Force Close**. Optionally, you can replace the default text for **Reason** with your own description. RabbitMQ on Amazon MQ will return the reason you specify to the client when you close the connection.
5. Choose **OK** on the dialog box to confirm and close the connection.

When you close a connection, any channels associated with closed connection will also be closed.

> **Note**
>
> Your client applications may be configured to automatically re-establish connections to the broker after they are closed. In this case, closing connections from the broker web console will not be sufficient for reducing connection or channel counts.

For brokers without public access, you can temporarily block connections by denying inbound traffic on the appropriate message protocol port, for example, port 5671 for AMQP connections. You can block the port in the security group that you provided to Amazon MQ when creating the broker. For more information on modifying your security group, see Adding rules to a security group in the *Amazon VPC User Guide*.

Step 2: Connect a JVM-based application to your broker

After you create a RabbitMQ broker, you can connect your application to it. The following examples show how you can use the RabbitMQ Java client library to create a connection to your broker, create a queue, and send a message. You can connect to RabbitMQ brokers using supported RabbitMQ client libraries for a variety of languages. For more information on supported RabbitMQ client libraries, see RabbitMQ client libraries and developer tools.

Prerequisites

> **Note**
>
> The following prerequisite steps are only applicable to RabbitMQ brokers created without public accessibility. If you are creating a broker with public accessibility you can skip them.

Enable VPC attributes

To ensure that your broker is accessible within your VPC, you must enable the `enableDnsHostnames` and `enableDnsSupport` VPC attributes. For more information, see DNS Support in your VPC in the *Amazon VPC User Guide*.

Enable inbound connections

1. Sign in to the Amazon MQ console.

2. From the broker list, choose the name of your broker (for example, **MyBroker**).
3. On the ***MyBroker*** page, in the **Connections** section, note the addresses and ports of the broker's web console URL and wire-level protocols.
4. In the **Details** section, under **Security and network**, choose the name of your security group or  .

   The **Security Groups** page of the EC2 Dashboard is displayed.
5. From the security group list, choose your security group.
6. At the bottom of the page, choose **Inbound**, and then choose **Edit**.
7. In the **Edit inbound rules** dialog box, add a rule for every URL or endpoint that you want to be publicly accessible (the following example shows how to do this for a broker web console).

   a. Choose **Add Rule**.

   b. For **Type**, select **Custom TCP**.

   c. For **Source**, leave **Custom** selected and then type the IP address of the system that you want to be able to access the web console (for example, `192.0.2.1`).

   d. Choose **Save**.

   Your broker can now accept inbound connections.

Add Java dependencies

If you are using Apache Maven for automating builds, add the following dependency to your `pom.xml` file. For more information on Project Object Model files in Apache Maven, see Introduction to the POM.

```
<dependency>
    <groupId>com.rabbitmq</groupId>
    <artifactId>amqp-client</artifactId>
    <version>5.9.0</version>
</dependency>
```

If you are using Gradle for automating builds, declare the following dependency.

```
dependencies {
    compile 'com.rabbitmq:amqp-client:5.9.0'
}
```

Import `Connection` and `Channel` classes

RabbitMQ Java client uses `com.rabbitmq.client` as its top-level package, with `Connection` and `Channel` API classes representing an AMQP 0-9-1 connection and channel, respectively. Import the `Connection` and `Channel` classes before using them, as shown in the following example.

```
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.Channel;
```

Create a `ConnectionFactory` and connect to your broker

Use the following example to create an instance of the `ConnectionFactory` class with the given parameters. Use the `setHost` method to configure the broker endpoint you noted earlier. For AMQPS wire-level connections, use port 5671.

```
ConnectionFactory factory = new ConnectionFactory();

factory.setUsername(username);
factory.setPassword(password);

//Replace the URL with your information
factory.setHost("b-c8352341-ec91-4a78-ad9c-a43f23d325bb.mq.us-west-2.amazonaws.com");
factory.setPort(5671);

// Allows client to establish a connection over TLS
factory.useSslProtocol();

// Create a connection
Connection conn = factory.newConnection();

// Create a channel
Channel channel = conn.createChannel();
```

Publish a message to an exchange

You can use `Channel.basicPublish` to publish messages to an exchange. The following example uses the AMQP `Builder` class to build a message properties object with content-type `plain/text`.

```
byte[] messageBodyBytes = "Hello, world!".getBytes();
```

```
channel.basicPublish(exchangeName, routingKey,
             new AMQP.BasicProperties.Builder()
               .contentType("text/plain")
               .userId("userId")
               .build(),
               messageBodyBytes);
```

Note

Note that `BasicProperties` is an inner class of the autogenerated holder class, `AMQP`.

Subscribe to a queue and receive a message

You can receive a message by subscribing to a queue using the `Consumer` interface. Once subscribed, messages will then be delivered automatically as they arrive.

The easiest way to implement a `Consumer` is to use the subclass `DefaultConsumer`. A `DefaultConsumer` object can be passed as part of a `basicConsume` call to set up the subscription as shown in the following example.

```
boolean autoAck = false;
channel.basicConsume(queueName, autoAck, "myConsumerTag",
     new DefaultConsumer(channel) {
         @Override
         public void handleDelivery(String consumerTag,
                                    Envelope envelope,
                                    AMQP.BasicProperties properties,
                                    byte[] body)
             throws IOException
         {
             String routingKey = envelope.getRoutingKey();
             String contentType = properties.getContentType();
             long deliveryTag = envelope.getDeliveryTag();
             // (process the message components here ...)
             channel.basicAck(deliveryTag, false);
         }
     });
```

> **Note**
>
> Because we specified `autoAck = false`, it is necessary to acknowledge messages delivered to the `Consumer`, most conveniently done in the `handleDelivery` method, as shown in the example.

Close your connection and disconnect from the broker

In order to disconnect from your RabbitMQ broker, close both the channel and connection as shown in the following.

```
channel.close();
conn.close();
```

> **Note**
>
> For more information on working with the RabbitMQ Java client library, see the RabbitMQ Java Client API Guide.

Step 3: (Optional) Connect to an AWS Lambda function

AWS Lambda can connect to and consume messages from your Amazon MQ broker. When you connect a broker to Lambda, you create an event source mapping that reads messages from a queue and invokes the function synchronously. The event source mapping you create reads messages from your broker in batches and converts them into a Lambda payload in the form of a JSON object.

To connect your broker to a Lambda function

1. Add the following IAM role permissions to your Lambda function execution role.

   - mq:DescribeBroker
   - ec2:CreateNetworkInterface
   - ec2:DeleteNetworkInterface
   - ec2:DescribeNetworkInterfaces
   - ec2:DescribeSecurityGroups

- ec2:DescribeSubnets
- ec2:DescribeVpcs
- logs:CreateLogGroup
- logs:CreateLogStream
- logs:PutLogEvents
- secretsmanager:GetSecretValue

> **Note**
>
> Without the necessary IAM permissions, your function will not be able to successfully read records from Amazon MQ resources.

2. (Optional) If you have created a broker without public accessibility, you must do one of the following to allow Lambda to connect to your broker:
   - Configure one NAT gateway per public subnet. For more information, see Internet and service access for VPC-connected functions in the *AWS Lambda Developer Guide*.
   - Create a connection between your Amazon Virtual Private Cloud (Amazon VPC) and Lambda using a VPC endpoint. Your Amazon VPC must also connect to AWS Security Token Service (AWS STS) and Secrets Manager endpoints. For more information, see Configuring interface VPC endpoints for Lambda in the *AWS Lambda Developer Guide*.
3. Configure your broker as an event source for a Lambda function using the AWS Management Console. You can also use the `create-event-source-mapping` AWS Command Line Interface command.
4. Write some code for your Lambda function to process the messages from your consumed from your broker. The Lambda payload that retrieved by your event source mapping depends on the engine type of the broker. The following is an example of a Lambda payload for an Amazon MQ for RabbitMQ queue.

> **Note**
>
> In the example, `test` is the name of the queue, and `/` is the name of the default virtual host. When receiving messages, the event source lists messages under `test::/`.

```
{
  "eventSource": "aws:rmq",
  "eventSourceArn": "arn:aws:mq:us-
west-2:112556298976:broker:test:b-9bcfa592-423a-4942-879d-eb284b418fc8",
  "rmqMessagesByQueue": {
    "test::/": [
      {
        "basicProperties": {
          "contentType": "text/plain",
          "contentEncoding": null,
          "headers": {
            "header1": {
              "bytes": [
                118,
                97,
                108,
                117,
                101,
                49
              ]
            },
            "header2": {
              "bytes": [
                118,
                97,
                108,
                117,
                101,
                50
              ]
            },
            "numberInHeader": 10
          }
          "deliveryMode": 1,
          "priority": 34,
          "correlationId": null,
          "replyTo": null,
          "expiration": "60000",
          "messageId": null,
          "timestamp": "Jan 1, 1970, 12:33:41 AM",
          "type": null,
          "userId": "AIDACKCEVSQ6C2EXAMPLE",
```

```
          "appId": null,
          "clusterId": null,
          "bodySize": 80
        },
        "redelivered": false,
        "data": "eyJ0aW1lb3V0IjowLCJkYXRhIjoiQ1pybWYwR3c4T3Y0YnFMUXhENEUifQ=="
      }
    ]
  }
}
```

For more information on connecting Amazon MQ to Lambda, the options Lambda supports for an Amazon MQ event source, and event source mapping errors, see Using Lambda with Amazon MQ in the *AWS Lambda Developer Guide*.

Using OAuth 2.0 authentication and authorization for Amazon MQ for RabbitMQ

This tutorial describes how to configure OAuth 2.0 authentication for your Amazon MQ for RabbitMQ brokers using Amazon Cognito as the OAuth 2.0 provider.

> **Note**
>
> Amazon Cognito is not available in China (Beijing) and China (Ningxia).

> **Important**
>
> This tutorial is specific to Amazon Cognito, but you can use other identity providers (IdPs). For more information, see OAuth 2.0 Authentication Examples.

On this page

Prerequisites to configure OAuth 2.0 authentication

You can set the Amazon Cognito resources required in this tutorial by deploying the AWS CDK stack, Amazon Cognito stack for RabbitMQ OAuth 2 plugin. If you're setting up Amazon Cognito manually, make sure that you fulfill the following prerequisites before configuring OAuth 2.0 on your Amazon MQ for RabbitMQ brokers:

Prerequisites to set up Amazon Cognito

- Set up an Amazon Cognito endpoint by creating a user pool. To do this, see the blog titled How to use OAuth 2.0 in Amazon Cognito: Learn about the different OAuth 2.0 grants.
- Create a resource server called `rabbitmq` in the user pool with the following scopes defined: `read:all`, `write:all`, `configure:all`, and `tag:administrator`. These scopes will be associated with RabbitMQ permissions.

  For information about creating a resource server, see Defining a resource server for your user pool (AWS Management Console) in the *Amazon Cognito Developer Guide*.
- Create the following application clients:
  - Application client for the user pool of type `Machine-to-Machine application`. This is a confidential client with a client secret that will be used for RabbitMQ AMQP clients. For more information about application clients and creating one, see App client types and Creating an app client.
  - Application client for the user pool of type `Single-page application`. This is a public client that will be used to log in users into the RabbitMQ management console. You must update this application client to include the endpoint of the Amazon MQ for RabbitMQ broker you'll create in the following procedure as an allowed callback URL. For more information, see Setting up managed login with the Amazon Cognito console.

Prerequisite to set up Amazon MQ

- A working Docker installation to run a bash script that verifies whether or not the OAuth 2.0 setup is successful.
- AWS CLI version >= `2.28.23` to make adding a username and password optional during broker creation.

Configuring OAuth 2.0 authentication with Amazon Cognito using AWS CLI

The following procedure shows how to set up OAuth 2.0 authentication for your Amazon MQ for RabbitMQ brokers using Amazon Cognito as the IdP. This procedure uses AWS CLI to create and configure the necessary resources.

In the following procedure, make sure to replace the placeholder values, such as configurationID and Revision, *<c-fa3390a5-7e01-4559-ae0c-eb15b38b22ca>* and *<2>*, with their actual values.

1. Create a new configuration using the create-configuration AWS CLI command as shown in the following example.

```
aws mq create-configuration \
  --name "rabbitmq-oauth2-config" \
  --engine-type "RABBITMQ" \
  --engine-version "3.13"
```

   This command returns a response similar to the following example.

```
{
    "Arn": "arn:aws:mq:us-west-2:123456789012:configuration:c-fa3390a5-7e01-4559-
ae0c-eb15b38b22ca",
    "AuthenticationStrategy": "simple",
    "Created": "2025-07-17T16:03:01.759943+00:00",
    "Id": "c-fa3390a5-7e01-4559-ae0c-eb15b38b22ca",
    "LatestRevision": {
    "Created": "2025-07-17T16:03:01.759000+00:00",
    "Description": "Auto-generated default for rabbitmq-oauth2-config on RabbitMQ
 3.13",
    "Revision": 1
    },
    "Name": "rabbitmq-oauth2-config"
}
```

2. Create a configuration file called **`rabbitmq.conf`** to use OAuth 2.0 as the authentication and authorization method, as shown in the following example.

```
auth_backends.1 = oauth2
```

```
# FIXME: Update this value with the token signing key URL of your Amazon Cognito
 user pool.
# If you used the AWS CDK stack to deploy Amazon Cognito, this is one of the stack
 outputs.
auth_oauth2.jwks_url = ${RabbitMqOAuth2TestStack.JwksUri}
auth_oauth2.resource_server_id = rabbitmq
# Amazon Cognito does not include an audience field in access tokens
auth_oauth2.verify_aud = false

# Amazon Cognito does not allow * in its custom scopes. Use aliases to translate
 between Amazon Cognito and RabbitMQ.
auth_oauth2.scope_prefix = rabbitmq/
auth_oauth2.scope_aliases.1.alias = rabbitmq/read:all
auth_oauth2.scope_aliases.1.scope = rabbitmq/read:*/*
auth_oauth2.scope_aliases.2.alias = rabbitmq/write:all
auth_oauth2.scope_aliases.2.scope = rabbitmq/write:*/*
auth_oauth2.scope_aliases.3.alias = rabbitmq/configure:all
auth_oauth2.scope_aliases.3.scope = rabbitmq/configure:*/*

# Allow OAuth 2.0 login for RabbitMQ management console
management.oauth_enabled = true
# FIXME: Update this value with the client ID of your public application client
management.oauth_client_id
 = ${RabbitMqOAuth2TestStack.ManagementConsoleAppClientId}
# FIXME: Update this value with the base JWKS URI (without /.well-known/jwks.json)
auth_oauth2.issuer = ${RabbitMqOAuth2TestStack.Issuer}
management.oauth_scopes = rabbitmq/tag:administrator
```

This configuration uses scope aliases to map the scopes defined in Amazon Cognito to RabbitMQ compatible scopes.

3. Update the configuration using the update-configuration AWS CLI command as shown in the following example. In this command, add the configuration ID you received in the response of Step 1 of this procedure. For example, **`c-fa3390a5-7e01-4559-ae0c-eb15b38b22ca`**.

```
aws mq update-configuration \
  --configuration-id "<c-fa3390a5-7e01-4559-ae0c-eb15b38b22ca>" \
  --data "$(cat rabbitmq.conf | base64 --wrap=0)"
```

This command returns a response similar to the following example.

```
{
```

```
    "Arn": "arn:aws:mq:us-west-2:123456789012:configuration:c-b600ac8e-8183-4f74-
a713-983e59f30e3d",
    "Created": "2025-07-17T16:57:04.520931+00:00",
    "Id": "c-b600ac8e-8183-4f74-a713-983e59f30e3d",
    "LatestRevision": {
        "Created": "2025-07-17T16:57:39.172000+00:00",
        "Revision": 2
    },
    "Name": "rabbitmq-oauth2-config",
    "Warnings": []
}
```

4. Create a broker with the OAuth 2.0 configuration you created in the Step 2 of this procedure. To do this, use the create-broker AWS CLI command as shown in the following example. In this command, provide the configuration ID and revision number you obtained in the responses of Step 1 and 2 respectively. For example, **`c-fa3390a5-7e01-4559-ae0c-eb15b38b22ca`** and **`2`**.

```
aws mq create-broker \
 --broker-name "rabbitmq-oauth2-broker" \
 --engine-type "RABBITMQ" \
 --engine-version "3.13" \
 --host-instance-type "mq.m7g.large" \
 --deployment-mode "CLUSTER_MULTI_AZ" \
 --logs '{"General": true}' \
 --publicly-accessible \
 --configuration '{"Id": "<c-fa3390a5-7e01-4559-ae0c-
eb15b38b22ca>","Revision": <2>}' \
```

This command returns a response similar to the following example.

```
{
    "BrokerArn": "arn:aws:mq:us-west-2:123456789012:broker:rabbitmq-oauth2-
broker:b-2a1b5133-a10c-49d2-879b-8c176c34cf73",
    "BrokerId": "b-2a1b5133-a10c-49d2-879b-8c176c34cf73"
}
```

5. Verify that the broker's status transitions from `CREATION_IN_PROGRESS` to `RUNNING`, using the describe-broker AWS CLI command as shown in the following example. In this command, provide the broker ID you obtained in the result of the previous step For example, **`b-2a1b5133-a10c-49d2-879b-8c176c34cf73`**.

```
aws mq describe-broker \
 --broker-id "<b-2a1b5133-a10c-49d2-879b-8c176c34cf73>"
```

This command returns a response similar to the following example. The following response is an abbreviated version of the complete output that the `describe-broker` command returns. This response shows the broker status and the authentication strategy used to secure the broker. In this case, the `config_managed` authentication strategy indicates that the broker uses OAuth 2 authentication method.

```
{
    "AuthenticationStrategy": "config_managed",
    ...,
    "BrokerState": "RUNNING",
    ...
}
```

To log in to the RabbitMQ management console using OAuth2, the broker endpoint needs to be added as a valid callback URL in the corresponding Amazon Cognito app client. For more information, refer to Step 5 in the setup of our sample Amazon Cognito CDK stack.

6. Verify OAuth 2.0 authentication and authorization with the following `perf-test.sh` script.

   Use this bash script to test connectivity to your Amazon MQ for RabbitMQ broker. This script obtains a token from Amazon Cognito and verifies if the connection was properly configured. If it's successfully configured, you'll see your broker publish and consume messages.

   If you receive an `ACCESS_REFUSED` error, you can troubleshoot your configuration settings by using the CloudWatch logs for your broker. You can find the link for the CloudWatch log group for your broker in the Amazon MQ console.

   In this script, you'll need to provide the following values:

   - `CLIENT_ID` and `CLIENT_SECRET`: You can find these values on the **App clients** page of the Amazon Cognito console.
   - Cognito domain: You can find this on the Amazon Cognito console. Under **Branding**, choose **Domain**. On the **Domain** page, you can find this value under the **Resource servers** section.
   - Amazon MQ broker endpoint: You can find this value under **Connections** on the broker details page of the Amazon MQ console.

```
#! /bin/bash
set -e

# Client information
## FIXME: Update this value with the client ID and secret of your confidential
 application client
CLIENT_ID=${RabbitMqOAuth2TestStack.AmqpAppClientId}
CLIENT_SECRET=${RabbitMqOAuth2TestStack.AmqpAppClientSecret}

# FIXME: Update this value with the domain of your Amazon Cognito user pool
RESPONSE=$(curl -X POST ${RabbitMqOAuth2TestStack.TokenEndpoint} \
                -H "Content-Type: application/x-www-form-urlencoded" \
                -d
 "grant_type=client_credentials&client_id=${CLIENT_ID}&client_secret=${CLIENT_SECRET}&scope
configure:all rabbitmq/read:all rabbitmq/tag:administrator rabbitmq/write:all")


# Extract the access_token from the response.
# This token will be passed in the password field when connecting to the broker.
# Note that the username is left blank, the field is ignored by the plugin.
BROKER_PASSWORD=$(echo ${RESPONSE} | jq -r '.access_token')

# FIXME: Update this value with the endpoint of your broker. For
 example, b-89424106-7e0e-4abe-8e98-8de0dada7630.mq.us-east-1.on.aws.
BROKER_DNS=<broker_dns>
CONNECTION_STRING=amqps://:${BROKER_PASSWORD}@${BROKER_DNS}:5671

# Produce/consume messages using the above connection string
QUEUES_COUNT=1
PRODUCERS_COUNT=1
CONSUMERS_COUNT=1
PRODUCER_RATE=1

docker run -it --rm --ulimit nofile=40960:40960 pivotalrabbitmq/perf-test:latest \
    --queue-pattern 'test-queue-%d' --queue-pattern-from 1 --queue-pattern-to
 $QUEUES_COUNT \
    --producers $PRODUCERS_COUNT --consumers $CONSUMERS_COUNT \
    --id "test${QUEUES_COUNT}q${PRODUCERS_COUNT}p${CONSUMERS_COUNT}c
${PRODUCER_RATE}r" \
    --uri ${CONNECTION_STRING} \
    --flag persistent --rate $PRODUCER_RATE
```

Configuring OAuth 2.0 and simple authentication with Amazon Cognito

When you create a broker with OAuth 2.0 authentication, you can specify one of the following authentication methods:

- **OAuth 2.0 only**: To use this method, don't provide a username and password while creating the broker. The previous procedure shows how to use only OAuth 2.0 authentication method.
- **Both OAuth 2.0 and simple authentication**: To use this method, provide a username and password while creating the broker. Also, add `auth_backends.2 = internal` to your broker configuration, as shown in the following procedure.

In the following procedure, make sure to replace the placeholder values, such as *<ConfigurationId>* and *<Revision>*, with their actual values.

1. To use both the authentication methods, create your broker configuration, as shown in the following example.

```
auth_backends.1 = oauth2
auth_backends.2 = internal

# FIXME: Update this value with the token signing key URL of your Amazon Cognito
 user pool
auth_oauth2.jwks_url = ${RabbitMqOAuth2TestStack.JwksUri}
auth_oauth2.resource_server_id = rabbitmq
auth_oauth2.verify_aud = false

auth_oauth2.scope_prefix = rabbitmq/
auth_oauth2.scope_aliases.1.alias = rabbitmq/read:all
auth_oauth2.scope_aliases.1.scope = rabbitmq/read:*/*
auth_oauth2.scope_aliases.2.alias = rabbitmq/write:all
auth_oauth2.scope_aliases.2.scope = rabbitmq/write:*/*
auth_oauth2.scope_aliases.3.alias = rabbitmq/configure:all
auth_oauth2.scope_aliases.3.scope = rabbitmq/configure:*/*
```

   This configuration uses scope aliases to map the scopes defined in Amazon Cognito to RabbitMQ compatible scopes.

2. Create a broker that uses both the authentication methods, as shown in the following example.

```
aws mq create-broker \
 --broker-name "rabbitmq-oauth2-broker-with-internal-user" \
 --engine-type "RABBITMQ" \
 --engine-version "3.13" \
 --host-instance-type "mq.m7g.large" \
 --deployment-mode "CLUSTER_MULTI_AZ" \
 --logs '{"General": true}' \
 --publicly-accessible \
 --configuration '{"Id": "<ConfigurationId>","Revision": <Revision>}' \
 --users '[{"Username":"<myUser>","Password":"<myPassword11>"}]'
```

3. Verify the broker status and the configuration for setting up the authentication method was successful as described in Steps 5 and 6 of the Configuring OAuth 2.0 authentication with Amazon Cognito procedure.

Using IAM authentication and authorization for Amazon MQ for RabbitMQ

The following procedure demonstrates how to enable AWS IAM authentication and authorization for an Amazon MQ for RabbitMQ broker. After enabling IAM, users can authenticate using AWS IAM credentials to access the RabbitMQ Management API and connect via AMQP. For details on how IAM authentication works with Amazon MQ for RabbitMQ, see the section called "IAM authentication and authorization".

Prerequisites

- AWS administrator credentials for the AWS account that owns the Amazon MQ for RabbitMQ broker
- A shell environment configured with these administrator credentials (using AWS CLI profiles or environment variables)
- AWS CLI installed and configured
- `jq` command-line JSON processor installed
- `curl` command-line tool installed

Configuring IAM authentication and authorization using AWS CLI

1. **Set environment variables**

Set the required environment variables for your broker:

```
export AWS_DEFAULT_REGION=<region>
export BROKER_ID=<broker-id>
```

2. **Enable outbound JWT tokens**

   Enable outbound web identity federation for your AWS account:

```
ISSUER_IDENTIFIER=$(aws iam enable-outbound-web-identity-federation --query
 'IssuerIdentifier' --output text)
echo $ISSUER_IDENTIFIER
```

   The output displays a unique issuer identifier URL for your account in the format `https://<id>.tokens.sts.global.api.aws`.

3. **Create the IAM policy document**

   Create a policy document that grants permissions to obtain web identity tokens:

```
cat > policy.json << 'EOF'
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "sts:GetWebIdentityToken",
                "sts:TagGetWebIdentityToken"
            ],
            "Resource": "*"
        }
    ]
}
EOF
```

4. **Create the trust policy**

   Retrieve your caller identity and create a trust policy document:

```
CALLER_ARN=$(aws sts get-caller-identity --query Arn --output text)
cat > trust-policy.json << EOF
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "$CALLER_ARN"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
EOF
```

5. **Create the IAM role**

   Create the IAM role and attach the policy:

```
aws iam create-role --role-name RabbitMqAdminRole --assume-role-policy-document
 file://trust-policy.json
aws iam put-role-policy --role-name RabbitMqAdminRole --policy-name
 RabbitMqAdminRolePolicy --policy-document file://policy.json
```

6. **Configure RabbitMQ OAuth2 settings**

   Create a RabbitMQ configuration file with OAuth2 authentication and authorization settings:

```
cat > rabbitmq.conf << EOF
auth_backends.1 = oauth2
auth_backends.2 = internal
```

```
auth_oauth2.jwks_url = ${ISSUER_IDENTIFIER}/.well-known/jwks.json
auth_oauth2.resource_server_id = rabbitmq
auth_oauth2.scope_prefix = rabbitmq/

auth_oauth2.additional_scopes_key = sub
auth_oauth2.scope_aliases.1.alias = arn:aws:iam::$(aws sts get-caller-identity --
query Account --output text):role/RabbitMqAdminRole
auth_oauth2.scope_aliases.1.scope = rabbitmq/tag:administrator rabbitmq/read:*/*
 rabbitmq/write:*/* rabbitmq/configure:*/*
auth_oauth2.https.hostname_verification = wildcard

management.oauth_enabled = true
EOF
```

7. **Update the broker configuration**

   Apply the new configuration to your broker:

```
# Retrieve the configuration ID
CONFIG_ID=$(aws mq describe-broker --broker-id $BROKER_ID --query
 'Configurations[0].Id' --output text)

# Create a new configuration revision
REVISION=$(aws mq update-configuration --configuration-id $CONFIG_ID --data "$(cat
 rabbitmq.conf | base64 --wrap=0)" --query 'LatestRevision.Revision' --output text)

# Apply the configuration to the broker
aws mq update-broker --broker-id $BROKER_ID --configuration Id=$CONFIG_ID,Revision=
$REVISION

# Reboot the broker to apply changes
aws mq reboot-broker --broker-id $BROKER_ID
```

   Wait for the broker status to return to `RUNNING` before proceeding to the next step.

8. **Obtain a JWT token**

   Assume the IAM role and obtain a web identity token:

```
# Assume the RabbitMqAdminRole
ROLE_CREDS=$(aws sts assume-role --role-arn arn:aws:iam::$(aws sts get-caller-
identity --query Account --output text):role/RabbitMqAdminRole --role-session-name
 rabbitmq-session)

# Configure the session with temporary credentials
export AWS_ACCESS_KEY_ID=$(echo "$ROLE_CREDS" | jq -r '.Credentials.AccessKeyId')
export AWS_SECRET_ACCESS_KEY=$(echo "$ROLE_CREDS" | jq -r
 '.Credentials.SecretAccessKey')
export AWS_SESSION_TOKEN=$(echo "$ROLE_CREDS" | jq -r '.Credentials.SessionToken')

# Obtain the web identity token
TOKEN_RESPONSE=$(aws sts get-web-identity-token \
    --audience "rabbitmq" \
    --signing-algorithm ES384 \
    --duration-seconds 300 \
    --tags Key=scope,Value="rabbitmq/tag:administrator")

# Extract the token
TOKEN=$(echo "$TOKEN_RESPONSE" | jq -r '.WebIdentityToken')
```

9. **Access the RabbitMQ Management API**

   Use the JWT token to access the RabbitMQ Management API:

   ```
   BROKER_URL=<broker-id>.mq.<region>.on.aws

   curl -u ":$TOKEN" \
       -X GET https://${BROKER_URL}/api/overview \
       -H "Content-Type: application/json"
   ```

   A successful response confirms that IAM authentication is working correctly. The response contains broker overview information in JSON format.

10. **Connect via AMQP using the JWT token**

    Test AMQP connectivity using the JWT token with the perf-test tool:

```
BROKER_DNS=<broker-endpoint>
CONNECTION_STRING=amqps://:${TOKEN}@${BROKER_DNS}:5671

docker run -it --rm --ulimit nofile=40960:40960 pivotalrabbitmq/perf-test:latest \
    --queue-pattern 'test-queue-%d' --queue-pattern-from 1 --queue-pattern-to 1 \
    --producers 1 --consumers 1 \
    --uri ${CONNECTION_STRING} \
    --flag persistent --rate 1
```

If you receive an ACCESS_REFUSED error, you can troubleshoot your configuration settings by using the CloudWatch logs for your broker. You can find the link for the CloudWatch Logs log group for your broker in the Amazon MQ console.

Using LDAP authentication and authorization for Amazon MQ for RabbitMQ

This tutorial describes how to configure LDAP authentication and authorization for your Amazon MQ for RabbitMQ brokers using AWS Managed Microsoft AD.

On this page

Prerequisites to configure LDAP authentication and authorization

You can set up the AWS resources required in this tutorial by deploying the AWS CDK stack for Amazon MQ for RabbitMQ LDAP integration with AWS Managed Microsoft AD.

This CDK stack automatically creates all the necessary AWS resources including AWS Managed Microsoft AD, LDAP users and groups, Network Load Balancer, certificates, and IAM roles. See the package README for a complete list of resources created by the stack.

If you're setting up the resources manually instead of using the CDK stack, ensure you have the equivalent infrastructure in place before configuring LDAP on your Amazon MQ for RabbitMQ brokers.

Prerequisite to set up Amazon MQ

AWS CLI version >= 2.28.23 to make adding a username and password optional during broker creation.

Configuring LDAP in RabbitMQ using AWS CLI

This procedure uses AWS CLI to create and configure the necessary resources. In the following procedure, make sure to replace the placeholder values, such as configurationID and Revision, <c-fa3390a5-7e01-4559-ae0c-eb15b38b22ca> and <2>, with their actual values.

1. Create a new configuration using the `create-configuration` AWS CLI command as shown in the following example.

```
aws mq create-configuration \
  --name "rabbitmq-ldap-config" \
  --engine-type "RABBITMQ" \
  --engine-version "3.13"
```

   This command returns a response similar to the following example.

```
{
"Arn": "arn:aws:mq:us-west-2:123456789012:configuration:c-fa3390a5-7e01-4559-ae0c-
eb15b38b22ca",
    "AuthenticationStrategy": "simple",
    "Created": "2025-07-17T16:03:01.759943+00:00",
    "Id": "c-fa3390a5-7e01-4559-ae0c-eb15b38b22ca",
    "LatestRevision": {
"Created": "2025-07-17T16:03:01.759000+00:00",
    "Description": "Auto-generated default for rabbitmq-ldap-config on RabbitMQ
 3.13",
    "Revision": 1
    },
    "Name": "rabbitmq-ldap-config"
}
```

2. Create a configuration file called `rabbitmq.conf` to use LDAP as the authentication and authorization method, as shown in the following example. Replace all placeholder values in

the template (marked with `${RabbitMqLdapTestStack.*}`) with actual values from your deployed AWS CDK prerequisite stack outputs or equivalent infrastructure.

```
auth_backends.1 = ldap

# LDAP authentication settings - For more information,
# see https://www.rabbitmq.com/docs/ldap#basic

# FIXME: Replace the ${RabbitMqLdapTestStack.*} placeholders with actual values
# from your deployed prerequisite CDK stack outputs.
auth_ldap.servers.1 = ${RabbitMqLdapTestStack.NlbDnsName}
auth_ldap.dn_lookup_bind.user_dn = ${RabbitMqLdapTestStack.DnLookupUserDn}
auth_ldap.dn_lookup_base = ${RabbitMqLdapTestStack.DnLookupBase}
auth_ldap.dn_lookup_attribute = ${RabbitMqLdapTestStack.DnLookupAttribute}
auth_ldap.port = 636
auth_ldap.use_ssl = true
auth_ldap.ssl_options.verify = verify_peer
auth_ldap.log = network

# AWS integration for secure credential retrieval
# - see: https://github.com/amazon-mq/rabbitmq-aws
# The aws plugin allows RabbitMQ to securely retrieve credentials and certificates
# from AWS services.

# Replace the ${RabbitMqLdapTestStack.*} placeholders with actual ARN values
# from your deployed prerequisite CDK stack outputs.
aws.arns.auth_ldap.ssl_options.cacertfile = ${RabbitMqLdapTestStack.CaCertArn}
aws.arns.auth_ldap.dn_lookup_bind.password =
 ${RabbitMqLdapTestStack.DnLookupUserPasswordArn}
aws.arns.assume_role_arn = ${RabbitMqLdapTestStack.AmazonMqAssumeRoleArn}

# LDAP authorization queries - For more information,
# see: https://www.rabbitmq.com/docs/ldap#authorisation

# FIXME: Replace the ${RabbitMqLdapTestStack.*} placeholders with actual group DN
# values from your deployed prerequisite CDK stack outputs
# Uses Active Directory groups created by the prerequisite CDK stack
auth_ldap.queries.tags = '''
[{administrator, {in_group,
 "${RabbitMqLdapTestStack.RabbitMqAdministratorsGroupDn}"}},
{management,    {in_group,
 "${RabbitMqLdapTestStack.RabbitMqMonitoringUsersGroupDn}"}}]
```

```
'''

# FIXME: This provides all authenticated users access to all vhosts
# - update to restrict access as required
auth_ldap.queries.vhost_access = '''
{constant, true}
'''

# FIXME: This provides all authenticated users full access to all
# queues and exchanges - update to restrict access as required
auth_ldap.queries.resource_access = '''
{for, [    {permission, configure, {constant, true}},
     {permission, write,
      {for, [{resource, queue,    {constant, true}},
             {resource, exchange, {constant, true}}]}},
     {permission, read,
      {for, [{resource, exchange, {constant, true}},
             {resource, queue,    {constant, true}}]}}
    ]
}
'''

# FIXME: This provides all authenticated users access to all topics
# - update to restrict access as required
auth_ldap.queries.topic_access = '''
{for, [{permission, write, {constant, true}},
     {permission, read,  {constant, true}}
    ]
}
'''
```

3. Update the configuration using the `update-configuration` AWS CLI command as shown in the following example. In this command, add the configuration ID you received in the response of Step 1 of this procedure. For example, `c-fa3390a5-7e01-4559-ae0c-eb15b38b22ca`.

```
aws mq update-configuration \
  --configuration-id "<c-fa3390a5-7e01-4559-ae0c-eb15b38b22ca>" \
  --data "$(cat rabbitmq.conf | base64 --wrap=0)"
```

This command returns a response similar to the following example.

```
{
    "Arn": "arn:aws:mq:us-west-2:123456789012:configuration:c-b600ac8e-8183-4f74-
a713-983e59f30e3d",
    "Created": "2025-07-17T16:57:04.520931+00:00",
    "Id": "c-b600ac8e-8183-4f74-a713-983e59f30e3d",
    "LatestRevision": {
        "Created": "2025-07-17T16:57:39.172000+00:00",
        "Revision": 2
    },
    "Name": "rabbitmq-ldap-config",
    "Warnings": []
}
```

4. Create a broker with the LDAP configuration you created in the Step 2 of this procedure. To do this, use the `create-broker` AWS CLI command as shown in the following example. In this command, provide the configuration ID and revision number you obtained in the responses of Step 1 and 2 respectively. For example, `c-fa3390a5-7e01-4559-ae0c-eb15b38b22ca` and 2.

```
aws mq create-broker \
 --broker-name "rabbitmq-ldap-test-1" \
 --engine-type "RABBITMQ" \
 --engine-version "3.13" \
 --host-instance-type "mq.m7g.large" \
 --deployment-mode "CLUSTER_MULTI_AZ" \
 --logs '{"General": true}' \
 --publicly-accessible \
 --configuration '{"Id": "<c-fa3390a5-7e01-4559-ae0c-eb15b38b22ca>","Revision":
 <2>}'
```

This command returns a response similar to the following example.

```
{
    "BrokerArn": "arn:aws:mq:us-west-2:123456789012:broker:rabbitmq-ldap-
broker:b-2a1b5133-a10c-49d2-879b-8c176c34cf73",
    "BrokerId": "b-2a1b5133-a10c-49d2-879b-8c176c34cf73"
```

```
}
```

> **Broker naming restriction**
>
> The IAM role created by the prerequisite CDK stack restricts broker names to start with `rabbitmq-ldap-test`. Ensure your broker name follows this pattern or the IAM role will not have permission to assume the role for ARN resolution.

5. Verify that the broker's status transitions from `CREATION_IN_PROGRESS` to `RUNNING`, using the `describe-broker` AWS CLI command as shown in the following example. In this command, provide the broker ID you obtained in the result of the previous step For example, `b-2a1b5133-a10c-49d2-879b-8c176c34cf73`.

```
aws mq describe-broker \
 --broker-id "<b-2a1b5133-a10c-49d2-879b-8c176c34cf73>"
```

This command returns a response similar to the following example. The following response is an abbreviated version of the complete output that the `describe-broker` command returns. This response shows the broker status and the authentication strategy used to secure the broker. In this case, the `config_managed` authentication strategy indicates that the broker uses LDAP authentication method.

```
{
"AuthenticationStrategy": "config_managed",
    ...,
    "BrokerState": "RUNNING",
    ...
}
```

6. Validate RabbitMQ access using one of the test users created by the prerequisite CDK stack

```
# FIXME: Replace ${RabbitMqLdapTestStack.ConsoleUserPasswordArn} with the actual
 ARN from your deployed prerequisite CDK stack outputs
CONSOLE_PASSWORD=$(aws secretsmanager get-secret-value \
```

```
  --secret-id ${RabbitMqLdapTestStack.ConsoleUserPasswordArn} \
  --query 'SecretString' --output text)

# FIXME: Replace BrokerConsoleURL with the actual ConsoleURL retrieved by
# calling describe-broker for the broker created above
# Call management API /api/overview (should succeed)
curl -u RabbitMqConsoleUser:$CONSOLE_PASSWORD \
  https://${BrokerConsoleURL}/api/overview

# Try to create a user (should fail - console user only has monitoring permissions)
curl -u RabbitMqConsoleUser:$CONSOLE_PASSWORD \
  -X PUT https://${BrokerConsoleURL}/api/users/testuser \
  -H "Content-Type: application/json" \
  -d '{"password":"testpass","tags":"management"}'
```

Using HTTP authentication and authorization for Amazon MQ for RabbitMQ

This tutorial describes how to configure HTTP authentication and authorization for your Amazon MQ for RabbitMQ brokers using an external HTTP server.

> **Note**
>
> The HTTP authentication plugin is only available for Amazon MQ for RabbitMQ version 4 and above.

On this page

Prerequisites to configure HTTP authentication and authorization

You can set up the AWS resources required in this tutorial by deploying the AWS CDK stack for Amazon MQ for RabbitMQ HTTP authentication integration.

This CDK stack automatically creates all the necessary AWS resources including the HTTP authentication server, certificates, and IAM roles. See the package README for a complete list of resources created by the stack.

If you're setting up the resources manually instead of using the CDK stack, ensure you have the equivalent infrastructure in place before configuring HTTP authentication on your Amazon MQ for RabbitMQ brokers.

Prerequisite to set up Amazon MQ

AWS CLI version >= 2.28.23 to make adding a username and password optional during broker creation.

Configuring HTTP authentication in RabbitMQ using AWS CLI

This procedure uses AWS CLI to create and configure the necessary resources. In the following procedure, make sure to replace the placeholder values with their actual values.

1. Create a new configuration using the `create-configuration` AWS CLI command as shown in the following example.

```
aws mq create-configuration \
  --name "rabbitmq-http-config" \
  --engine-type "RABBITMQ" \
  --engine-version "4.2"
```

This command returns a response similar to the following example.

```
{
    "Arn": "arn:aws:mq:us-west-2:123456789012:configuration:c-fa3390a5-7e01-4559-
ae0c-eb15b38b22ca",
    "AuthenticationStrategy": "simple",
    "Created": "2025-07-17T16:03:01.759943+00:00",
    "Id": "c-fa3390a5-7e01-4559-ae0c-eb15b38b22ca",
    "LatestRevision": {
        "Created": "2025-07-17T16:03:01.759000+00:00",
        "Description": "Auto-generated default for rabbitmq-http-config on RabbitMQ
 4.2",
        "Revision": 1
```

```
    },
    "Name": "rabbitmq-http-config"
}
```

2. Create a configuration file called `rabbitmq.conf` to use HTTP as the authentication and authorization method, as shown in the following example. Replace all placeholder values in the template (marked with `${...}`) with actual values from your deployed AWS CDK prerequisite stack outputs or equivalent infrastructure.

```
auth_backends.1 = cache
auth_backends.2 = http
auth_cache.cached_backend = http

# HTTP authentication settings
# For more information, see https://github.com/rabbitmq/rabbitmq-auth-backend-http

# FIXME: Replace the ${...} placeholders with actual values
# from your deployed prerequisite CDK stack outputs.
auth_http.http_method = post
auth_http.user_path = ${HttpServerUserPath}
auth_http.vhost_path = ${HttpServerVhostPath}
auth_http.resource_path = ${HttpServerResourcePath}
auth_http.topic_path = ${HttpServerTopicPath}

# TLS/HTTPS configuration
auth_http.ssl_options.verify = verify_peer
auth_http.ssl_options.sni = test.amazonaws.com

# AWS integration for secure credential retrieval
# For more information, see https://github.com/amazon-mq/rabbitmq-aws

# Replace the ${...} placeholders with actual ARN values
# from your deployed prerequisite CDK stack outputs.
aws.arns.assume_role_arn = ${AmazonMqAssumeRoleArn}
aws.arns.auth_http.ssl_options.cacertfile = ${CaCertArn}
```

3. Update the configuration using the `update-configuration` AWS CLI command. Use the configuration ID from Step 3.

```
aws mq update-configuration \
  --configuration-id "<c-fa3390a5-7e01-4559-ae0c-eb15b38b22ca>" \
  --data "$(cat rabbitmq.conf | base64 --wrap=0)"
```

This command returns a response similar to the following example.

```
{
    "Arn": "arn:aws:mq:us-west-2:123456789012:configuration:c-fa3390a5-7e01-4559-
ae0c-eb15b38b22ca",
    "Created": "2025-07-17T16:57:04.520931+00:00",
    "Id": "c-fa3390a5-7e01-4559-ae0c-eb15b38b22ca",
    "LatestRevision": {
        "Created": "2025-07-17T16:57:39.172000+00:00",
        "Revision": 2
    },
    "Name": "rabbitmq-http-config",
    "Warnings": []
}
```

4. Create a broker with the HTTP configuration. Use the configuration ID and revision number from the previous steps.

```
aws mq create-broker \
  --broker-name "rabbitmq-http-test-1" \
  --engine-type "RABBITMQ" \
  --engine-version "4.2" \
  --host-instance-type "mq.m7g.large" \
  --deployment-mode "SINGLE_INSTANCE" \
  --logs '{"General": true}' \
  --publicly-accessible \
  --configuration '{"Id": "<c-fa3390a5-7e01-4559-ae0c-eb15b38b22ca>","Revision":
 <2>}'
```

This command returns a response similar to the following example.

```
{
```

```
    "BrokerArn": "arn:aws:mq:us-west-2:123456789012:broker:rabbitmq-http-
test-1:b-2a1b5133-a10c-49d2-879b-8c176c34cf73",
    "BrokerId": "b-2a1b5133-a10c-49d2-879b-8c176c34cf73"
}
```

5. Verify that the broker's status transitions from `CREATION_IN_PROGRESS` to `RUNNING`, using the `describe-broker` AWS CLI command.

```
aws mq describe-broker \
  --broker-id "<b-2a1b5133-a10c-49d2-879b-8c176c34cf73>"
```

This command returns a response similar to the following example. The `config_managed` authentication strategy indicates that the broker uses HTTP authentication method.

```
{
    "AuthenticationStrategy": "config_managed",
    ...,
    "BrokerState": "RUNNING",
    ...
}
```

6. Validate RabbitMQ access using one of the test users created by the prerequisite CDK stack

```
# FIXME: Replace ${RabbitMqHttpAuthElbStack.ConsoleUserPasswordArn} with the actual
 ARN from your deployed prerequisite CDK stack outputs
CONSOLE_PASSWORD=$(aws secretsmanager get-secret-value \
  --secret-id ${RabbitMqHttpAuthElbStack.ConsoleUserPasswordArn} \
  --query 'SecretString' --output text)

# FIXME: Replace BrokerConsoleURL with the actual ConsoleURL retrieved by
# calling describe-broker for the broker created above
# Call management API /api/overview (should succeed)
curl -u RabbitMqConsoleUser:$CONSOLE_PASSWORD \
  https://${BrokerConsoleURL}/api/overview
```

```
# Try to create a vhost (should fail - console user only has management
 permissions)
curl -u RabbitMqConsoleUser:$CONSOLE_PASSWORD \
  -X PUT https://${BrokerConsoleURL}/api/vhosts/test-vhost \
  -H "Content-Type: application/json" \
  -d '{}'
```

Using SSL certificate authentication for Amazon MQ for RabbitMQ

This tutorial describes how to configure SSL certificate authentication for your Amazon MQ for RabbitMQ brokers using a private certificate authority.

> **Note**
>
> The SSL certificate authentication plugin is only available for Amazon MQ for RabbitMQ version 4 and above.

On this page

Prerequisites to configure SSL certificate authentication

SSL certificate authentication uses mutual TLS (mTLS) to authenticate clients using X.509 certificates. You can set up the AWS resources required in this tutorial by deploying the AWS CDK stack for Amazon MQ for RabbitMQ mTLS integration.

This CDK stack automatically creates all the necessary AWS resources including certificate authority, client certificates, and IAM roles. See the package README for a complete list of resources created by the stack.

> **Note**
>
> Before deploying the CDK stack, set the `RABBITMQ_TEST_USER_NAME` environment variable. This value will be used as the Common Name (CN) in the client certificate

and must match the username you use in the tutorial steps. For example: `export RABBITMQ_TEST_USER_NAME="myuser"`

If you're setting up the resources manually instead of using the CDK stack, ensure you have the equivalent infrastructure in place before configuring SSL certificate authentication on your Amazon MQ for RabbitMQ brokers.

Prerequisite to set up Amazon MQ

AWS CLI version >= 2.28.23 to make adding a username and password optional during broker creation.

Configuring SSL certificate authentication in RabbitMQ using AWS CLI

This procedure uses AWS CLI to create and configure the necessary resources. In the following procedure, make sure to replace the placeholder values, such as configurationID and Revision, `<c-fa3390a5-7e01-4559-ae0c-eb15b38b22ca>` and <2>, with their actual values.

1. Create a new configuration using the `create-configuration` AWS CLI command as shown in the following example.

```
aws mq create-configuration \
  --name "rabbitmq-ssl-config" \
  --engine-type "RABBITMQ" \
  --engine-version "4.2"
```

This command returns a response similar to the following example.

```
{
    "Arn": "arn:aws:mq:us-west-2:123456789012:configuration:c-fa3390a5-7e01-4559-
ae0c-eb15b38b22ca",
    "AuthenticationStrategy": "simple",
    "Created": "2025-07-17T16:03:01.759943+00:00",
    "Id": "c-fa3390a5-7e01-4559-ae0c-eb15b38b22ca",
    "LatestRevision": {
        "Created": "2025-07-17T16:03:01.759000+00:00",
```

```
        "Description": "Auto-generated default for rabbitmq-ssl-config on RabbitMQ
 4.2",
        "Revision": 1
    },
    "Name": "rabbitmq-ssl-config"
}
```

2. Create a configuration file called `rabbitmq.conf` to use SSL certificate authentication, as shown in the following example. Replace all placeholder values in the template (marked with `${...}`) with actual values from your deployed AWS CDK prerequisite stack outputs or equivalent infrastructure.

```
auth_mechanisms.1 = EXTERNAL
ssl_cert_login_from = common_name

auth_backends.1 = internal

# Reject if no client cert
ssl_options.verify = verify_peer
ssl_options.fail_if_no_peer_cert = true

# AWS integration for secure credential retrieval
# For more information, see https://github.com/amazon-mq/rabbitmq-aws

# FIXME: Replace the ${...} placeholders with actual ARN values
# from your deployed prerequisite CDK stack outputs.
aws.arns.assume_role_arn = ${AmazonMqAssumeRoleArn}
aws.arns.ssl_options.cacertfile = ${CaCertArn}
```

3. Update the configuration using the `update-configuration` AWS CLI command as shown in the following example. In this command, add the configuration ID you received in the response of Step 1 of this procedure. For example, `c-fa3390a5-7e01-4559-ae0c-eb15b38b22ca`.

```
aws mq update-configuration \
  --configuration-id "<c-fa3390a5-7e01-4559-ae0c-eb15b38b22ca>" \
  --data "$(cat rabbitmq.conf | base64 --wrap=0)"
```

This command returns a response similar to the following example.

```
{
    "Arn": "arn:aws:mq:us-west-2:123456789012:configuration:c-fa3390a5-7e01-4559-
ae0c-eb15b38b22ca",
    "Created": "2025-07-17T16:57:04.520931+00:00",
    "Id": "c-fa3390a5-7e01-4559-ae0c-eb15b38b22ca",
    "LatestRevision": {
        "Created": "2025-07-17T16:57:39.172000+00:00",
        "Revision": 2
    },
    "Name": "rabbitmq-ssl-config",
    "Warnings": []
}
```

4. Create a broker with the SSL certificate authentication configuration you created in Step 2 of this procedure. To do this, use the `create-broker` AWS CLI command as shown in the following example. In this command, provide the configuration ID and revision number you obtained in the responses of Step 1 and 2 respectively. For example, `c-fa3390a5-7e01-4559-ae0c-eb15b38b22ca` and 2.

```
aws mq create-broker \
  --broker-name "rabbitmq-ssl-test-1" \
  --engine-type "RABBITMQ" \
  --engine-version "4.2" \
  --host-instance-type "mq.m7g.large" \
  --deployment-mode "SINGLE_INSTANCE" \
  --logs '{"General": true}' \
  --publicly-accessible \
  --configuration '{"Id": "<c-fa3390a5-7e01-4559-ae0c-eb15b38b22ca>","Revision":
 <2>}' \
  --users '[{"Username":"testuser","Password":"testpassword"}]'
```

This command returns a response similar to the following example.

```
{
```

```
    "BrokerArn": "arn:aws:mq:us-west-2:123456789012:broker:rabbitmq-ssl-
test-1:b-2a1b5133-a10c-49d2-879b-8c176c34cf73",
    "BrokerId": "b-2a1b5133-a10c-49d2-879b-8c176c34cf73"
}
```

5. Verify that the broker's status transitions from `CREATION_IN_PROGRESS` to `RUNNING`, using the `describe-broker` AWS CLI command as shown in the following example. In this command, provide the broker ID you obtained in the result of the previous step. For example, `b-2a1b5133-a10c-49d2-879b-8c176c34cf73`.

```
aws mq describe-broker \
  --broker-id "<b-2a1b5133-a10c-49d2-879b-8c176c34cf73>"
```

   This command returns a response similar to the following example. The following response is an abbreviated version of the complete output that the `describe-broker` command returns. This response shows the broker status and the authentication strategy used to secure the broker. In this case, the `config_managed` authentication strategy indicates that the broker uses SSL certificate authentication method.

```
{
    "AuthenticationStrategy": "config_managed",
    ...,
    "BrokerState": "RUNNING",
    ...
}
```

6. Verify SSL certificate authentication with the following `ssl.sh` script.

   Use this bash script to test connectivity to your Amazon MQ for RabbitMQ broker. This script uses your client certificate for authentication and verifies if the connection was properly configured. If it's successfully configured, you'll see your broker publish and consume messages.

If you receive an ACCESS_REFUSED error, you can troubleshoot your configuration settings by using the CloudWatch logs for your broker. You can find the link for the CloudWatch log group for your broker in the Amazon MQ console.

In this script, you'll need to provide the following values:

- USERNAME: The common name (CN) from your client certificate.
- CLIENT_KEYSTORE: Path to your client keystore file (PKCS12 format). If you used the prerequisite CDK stack, the default path is `$(pwd)/certs/client-keystore.p12`.
- KEYSTORE_PASSWORD: Password for your client keystore. If you used the prerequisite CDK stack, the default password is `changeit`.
- BROKER_DNS: You can find this value under **Connections** on the broker details page of the Amazon MQ console.

```
#! /bin/bash
set -e

# Client information
## FIXME: Update this value with the client ID and secret of your confidential
 application client
USERNAME=<client_cert_common_name>
CLIENT_KEYSTORE=$(pwd)/certs/client-keystore.p12
KEYSTORE_PASSWORD=changeit

BROKER_DNS=<broker_dns>
CONNECTION_STRING=amqps://${BROKER_DNS}:5671

# Produce/consume messages using the above connection string
QUEUES_COUNT=1
PRODUCERS_COUNT=1
CONSUMERS_COUNT=1
PRODUCER_RATE=1

finch run --rm --ulimit nofile=40960:40960 \
    -v ${CLIENT_KEYSTORE}:/certs/client-keystore.p12:ro \
    -e JAVA_TOOL_OPTIONS="-Djavax.net.ssl.keyStore=/certs/client-
keystore.p12 -Djavax.net.ssl.keyStorePassword=${KEYSTORE_PASSWORD} -
Djavax.net.ssl.keyStoreType=PKCS12" \
    pivotalrabbitmq/perf-test:latest \
```

```
    --queue-pattern 'test-queue-cert-%d' --queue-pattern-from 1 --queue-pattern-to
 $QUEUES_COUNT \
    --producers $PRODUCERS_COUNT --consumers $CONSUMERS_COUNT \
    --id "cert-test${QUEUES_COUNT}q${PRODUCERS_COUNT}p${CONSUMERS_COUNT}c
${PRODUCER_RATE}r" \
    --uri ${CONNECTION_STRING} \
    --sasl-external \
    --use-default-ssl-context \
    --flag persistent --rate $PRODUCER_RATE
```

Using mTLS for AMQP and management endpoints

This tutorial describes how to configure mutual TLS (mTLS) for AMQP client connections and the RabbitMQ management interface using a private certificate authority.

> **Note**
>
> The use of private certificate authorities for mTLS is only available for Amazon MQ for RabbitMQ version 4 and above.

On this page

Prerequisites to configure mTLS

You can set up the AWS resources required in this tutorial by deploying the AWS CDK stack for Amazon MQ for RabbitMQ mTLS integration with .

This CDK stack automatically creates all the necessary AWS resources including certificate authority, client certificates, and IAM roles. See the package README for a complete list of resources created by the stack.

If you're setting up the resources manually instead of using the CDK stack, ensure you have the equivalent infrastructure in place before configuring mTLS on your Amazon MQ for RabbitMQ brokers.

Prerequisite to set up Amazon MQ

AWS CLI version >= 2.28.23 to make adding a username and password optional during broker creation.

Configuring mTLS in RabbitMQ using AWS CLI

This procedure uses AWS CLI to create and configure the necessary resources. In the following procedure, make sure to replace the placeholder values, such as configurationID and Revision, <c-fa3390a5-7e01-4559-ae0c-eb15b38b22ca> and <2>, with their actual values.

1. Create a new configuration using the `create-configuration` AWS CLI command as shown in the following example.

```
aws mq create-configuration \
  --name "rabbitmq-mtls-config" \
  --engine-type "RABBITMQ" \
  --engine-version "4.2"
```

   This command returns a response similar to the following example.

```
{
    "Arn": "arn:aws:mq:us-west-2:123456789012:configuration:c-fa3390a5-7e01-4559-
ae0c-eb15b38b22ca",
    "AuthenticationStrategy": "simple",
    "Created": "2025-07-17T16:03:01.759943+00:00",
    "Id": "c-fa3390a5-7e01-4559-ae0c-eb15b38b22ca",
    "LatestRevision": {
        "Created": "2025-07-17T16:03:01.759000+00:00",
        "Description": "Auto-generated default for rabbitmq-mtls-config on RabbitMQ
 4.2",
        "Revision": 1
    },
    "Name": "rabbitmq-mtls-config"
}
```

2. Create a configuration file called `rabbitmq.conf` to configure mTLS for AMQP and management endpoints, as shown in the following example. Replace all placeholder values

in the template (marked with `${...}`) with actual values from your deployed AWS CDK prerequisite stack outputs or equivalent infrastructure.

```
auth_backends.1 = internal

# TLS configuration
ssl_options.verify = verify_peer
ssl_options.fail_if_no_peer_cert = true
management.ssl.verify = verify_peer

# AWS integration for secure credential retrieval
# For more information, see https://github.com/amazon-mq/rabbitmq-aws

# FIXME: Replace the ${...} placeholders with actual ARN values
# from your deployed prerequisite CDK stack outputs.
aws.arns.assume_role_arn = ${AmazonMqAssumeRoleArn}
aws.arns.ssl_options.cacertfile = ${CaCertArn}
aws.arns.management.ssl.cacertfile = ${CaCertArn}
```

3. Update the configuration using the `update-configuration` AWS CLI command as shown in the following example. In this command, add the configuration ID you received in the response of Step 1 of this procedure. For example, `c-fa3390a5-7e01-4559-ae0c-eb15b38b22ca`.

```
aws mq update-configuration \
  --configuration-id "<c-fa3390a5-7e01-4559-ae0c-eb15b38b22ca>" \
  --data "$(cat rabbitmq.conf | base64 --wrap=0)"
```

This command returns a response similar to the following example.

```
{
    "Arn": "arn:aws:mq:us-west-2:123456789012:configuration:c-fa3390a5-7e01-4559-
ae0c-eb15b38b22ca",
    "Created": "2025-07-17T16:57:04.520931+00:00",
    "Id": "c-fa3390a5-7e01-4559-ae0c-eb15b38b22ca",
    "LatestRevision": {
        "Created": "2025-07-17T16:57:39.172000+00:00",
        "Revision": 2
```

```
    },
    "Name": "rabbitmq-mtls-config",
    "Warnings": []
}
```

4. Create a broker with the mTLS configuration you created in Step 2 of this procedure. To do this, use the `create-broker` AWS CLI command as shown in the following example. In this command, provide the configuration ID and revision number you obtained in the responses of Step 1 and 2 respectively. For example, `c-fa3390a5-7e01-4559-ae0c-eb15b38b22ca` and 2.

```
aws mq create-broker \
  --broker-name "rabbitmq-mtls-test-1" \
  --engine-type "RABBITMQ" \
  --engine-version "4.2" \
  --host-instance-type "mq.m7g.large" \
  --deployment-mode "SINGLE_INSTANCE" \
  --logs '{"General": true}' \
  --publicly-accessible \
  --configuration '{"Id": "<c-fa3390a5-7e01-4559-ae0c-eb15b38b22ca>","Revision":
 <2>}' \
  --users '[{"Username":"testuser","Password":"testpassword"}]'
```

This command returns a response similar to the following example.

```
{
    "BrokerArn": "arn:aws:mq:us-west-2:123456789012:broker:rabbitmq-mtls-
test-1:b-2a1b5133-a10c-49d2-879b-8c176c34cf73",
    "BrokerId": "b-2a1b5133-a10c-49d2-879b-8c176c34cf73"
}
```

5. Verify that the broker's status transitions from `CREATION_IN_PROGRESS` to `RUNNING`, using the `describe-broker` AWS CLI command as shown in the following example. In this command, provide the broker ID you obtained in the result of the previous step. For example, `b-2a1b5133-a10c-49d2-879b-8c176c34cf73`.

```
aws mq describe-broker \
  --broker-id "<b-2a1b5133-a10c-49d2-879b-8c176c34cf73>"
```

This command returns a response similar to the following example. The following response is an abbreviated version of the complete output that the `describe-broker` command returns.

```
{
    "AuthenticationStrategy": "simple",
    ...,
    "BrokerState": "RUNNING",
    ...
}
```

6. Verify mTLS authentication with the following `mtls.sh` script.

   Use this bash script to test connectivity to your Amazon MQ for RabbitMQ broker. This script uses your client certificate to authenticate and verifies if the connection was properly configured. If it's successfully configured, you'll see your broker publish and consume messages.

   If you receive an `ACCESS_REFUSED` error, you can troubleshoot your configuration settings by using the CloudWatch logs for your broker. You can find the link for the CloudWatch log group for your broker in the Amazon MQ console.

   In this script, you'll need to provide the following values:

   - `USERNAME` and `PASSWORD`: The RabbitMQ user credentials you created with the broker.
   - `CLIENT_KEYSTORE`: Path to your client keystore file (PKCS12 format). If you used the prerequisite CDK stack, the default path is `$(pwd)/certs/client-keystore.p12`.
   - `KEYSTORE_PASSWORD`: Password for your client keystore. If you used the prerequisite CDK stack, the default password is `changeit`.
   - `BROKER_DNS`: You can find this value under **Connections** on the broker details page of the Amazon MQ console.

```
#! /bin/bash
set -e

# Client information
## FIXME: Update this value with the client ID and secret of your confidential
 application client
USERNAME=<testuser>
PASSWORD=<testpassword>
CLIENT_KEYSTORE=$(pwd)/certs/client-keystore.p12
KEYSTORE_PASSWORD=changeit

BROKER_DNS=<broker_dns>
CONNECTION_STRING=amqps://${USERNAME}:${PASSWORD}@${BROKER_DNS}:5671

# Produce/consume messages using the above connection string
QUEUES_COUNT=1
PRODUCERS_COUNT=1
CONSUMERS_COUNT=1
PRODUCER_RATE=1

finch run --rm --ulimit nofile=40960:40960 \
    -v ${CLIENT_KEYSTORE}:/certs/client-keystore.p12:ro \
    -e JAVA_TOOL_OPTIONS="-Djavax.net.ssl.keyStore=/certs/client-
keystore.p12 -Djavax.net.ssl.keyStorePassword=${KEYSTORE_PASSWORD} -
Djavax.net.ssl.keyStoreType=PKCS12" \
    pivotalrabbitmq/perf-test:latest \
    --queue-pattern 'test-queue-cert-%d' --queue-pattern-from 1 --queue-pattern-to
 $QUEUES_COUNT \
    --producers $PRODUCERS_COUNT --consumers $CONSUMERS_COUNT \
    --id "cert-test${QUEUES_COUNT}q${PRODUCERS_COUNT}p${CONSUMERS_COUNT}c
${PRODUCER_RATE}r" \
    --uri ${CONNECTION_STRING} \
    --use-default-ssl-context \
    --flag persistent --rate $PRODUCER_RATE
```

Connecting your JMS application

This tutorial shows you how to connect your JMS application to Amazon MQ for RabbitMQ broker using the RabbitMQ JMS client. You will learn how to create a producer to send messages and a consumer to receive messages from RabbitMQ queues.

Before you begin, add the appropriate RabbitMQ JMS dependency to your Maven project:

For JMS 1.1 and 2.0:

```
<dependencies>

  <dependency>
    <groupId>com.rabbitmq.jms</groupId>
    <artifactId>rabbitmq-jms</artifactId>
    <version>2.12.0</version>
  </dependency>

</dependencies>
```

For JMS 3.1:

```
<dependencies>

  <dependency>
    <groupId>com.rabbitmq.jms</groupId>
    <artifactId>rabbitmq-jms</artifactId>
    <version>3.5.0</version>
  </dependency>

</dependencies>
```

Create a producer

The following code example shows how to write to a RabbitMQ queue using JMS:

```
import jakarta.jms.*;
import com.rabbitmq.jms.admin.*;

// Setting the connection factory
RMQConnectionFactory factory = new RMQConnectionFactory();
factory.setHost(envProps.getProperty("RABBITMQ_HOST", "localhost"));
factory.setPort(Integer.parseInt(envProps.getProperty("RABBITMQ_PORT", "5672")));
factory.setUsername(envProps.getProperty("RABBITMQ_USERNAME", "guest"));
factory.setPassword(envProps.getProperty("RABBITMQ_PASSWORD", "guest"));
factory.setVirtualHost(envProps.getProperty("RABBITMQ_VIRTUAL_HOST", "/"));
factory.useSslProtocol();
```

```
connection = factory.createConnection();
connection.start();

String queueName = "test-queue-jms";
Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);

RMQDestination destination = new RMQDestination(queueName, true, false);

// Send the message to the queue
MessageProducer producer = session.createProducer(destination);
producer.setDeliveryMode(DeliveryMode.PERSISTENT);

String msg_content = "Hello World!!";
TextMessage textMessage = session.createTextMessage(msg_content);
producer.send(textMessage);

System.out.printf("Published to AMQP queue '%s': %s", queueName, msg_content);
```

Create a consumer

The following code example shows how to read from a RabbitMQ queue using JMS:

```
import jakarta.jms.*;
import com.rabbitmq.jms.admin.*;

// Setting the connection factory
RMQConnectionFactory factory = new RMQConnectionFactory();
factory.setHost(envProps.getProperty("RABBITMQ_HOST", "localhost"));
factory.setPort(Integer.parseInt(envProps.getProperty("RABBITMQ_PORT", "5672")));
factory.setUsername(envProps.getProperty("RABBITMQ_USERNAME", "guest"));
factory.setPassword(envProps.getProperty("RABBITMQ_PASSWORD", "guest"));
factory.setVirtualHost(envProps.getProperty("RABBITMQ_VIRTUAL_HOST", "/"));
factory.useSslProtocol();

// Establish the connection and session
jakarta.jms.Connection connection = factory.createConnection();

String queueName = "test-queue-jms";
Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);

RMQDestination destination = new RMQDestination();
destination.setDestinationName(queueName);
destination.setAmqp(true);
```

```
destination.setAmqpQueueName(queueName);

// Initialize consumer
MessageConsumer consumer = session.createConsumer(destination);
consumer.setMessageListener(message -> {
    try {
        if (message instanceof TextMessage) {
            TextMessage textMessage = (TextMessage) message;
            System.out.printf("Message: %s%n", textMessage.getText());
        } else if (message instanceof BytesMessage) {
            BytesMessage bytesMessage = (BytesMessage) message;
            byte[] bytes = new byte[(int) bytesMessage.getBodyLength()];
            bytesMessage.readBytes(bytes);
            String content = new String(bytes);
            System.out.printf("Message: %s%n", content);
        } else {
            System.out.printf("Message: [%s]%n", message.getClass().getSimpleName());
        }
    } catch (JMSException e) {
        System.err.printf("Error processing message: %s%n", e.getMessage());
    }
});

connection.start();
```

Security in Amazon MQ

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 MQ, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using Amazon MQ. The following topics show you how to configure Amazon MQ 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 MQ resources.

Topics

Data protection in Amazon MQ

The AWS shared responsibility model applies to data protection in Amazon MQ. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud.

You are responsible for maintaining control over your content that is hosted on this infrastructure. 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 IAM Identity Center or 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 AWS CloudTrail. For information about using CloudTrail trails to capture AWS activities, see Working with CloudTrail trails in the *AWS 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.
- If you require FIPS 140-3 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-3.

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 MQ 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.

For both Amazon MQ for ActiveMQ and Amazon MQ for RabbitMQ brokers, do not use any personally identifiable information (PII) or other confidential or sensitive information for broker names or usernames when creating resources via the broker web console, or the Amazon MQ API. Broker names and usernames are accessible to other AWS services, including CloudWatch Logs. Broker usernames are not intended to be used for private or sensitive data.

> **Important**
>
> TLS 1.3 is not available for RabbitMQ brokers.

Encryption

User data stored in Amazon MQ is encrypted at rest. Amazon MQ encryption at rest provides enhanced security by encrypting your data using encryption keys stored in the AWS Key Management Service (KMS). This service helps reduce the operational burden and complexity involved in protecting sensitive data. With encryption at rest, you can build security-sensitive applications that meet encryption compliance and regulatory requirements.

All connections between Amazon MQ brokers use Transport layer Security (TLS) to provide encryption in transit.

Amazon MQ encrypts messages at rest and in transit using encryption keys that it manages and stores securely. For more information, see the *AWS Encryption SDK Developer Guide*.

Encryption at rest

Amazon MQ integrates with AWS Key Management Service (KMS) to offer transparent server-side encryption. Amazon MQ always encrypts your data at rest.

When you create an Amazon MQ for ActiveMQ broker or an Amazon MQ for RabbitMQ broker, you can specify the AWS KMS key that you want Amazon MQ to use to encrypt your data at rest. If you do not specify a KMS key, Amazon MQ creates an AWS owned KMS key for you and uses it on your behalf. Amazon MQ currently supports symmetric KMS keys. For more information about KMS keys, see AWS KMS keys.

When creating a broker, you can configure what Amazon MQ uses for your encryption key by selecting one of the following.

- **Amazon MQ owned KMS key (default)** — The key is owned and managed by Amazon MQ and is not in your account.
- **AWS managed KMS key** — The AWS managed KMS key (`aws/mq`) is a KMS key in your account that is created, managed, and used on your behalf by Amazon MQ.
- **Select existing customer managed KMS key** — Customer managed KMS keys are created and managed by you in AWS Key Management Service (KMS).

Important

- Revoking a grant cannot be undone. Delete the broker to revoke access rights.
- For **Amazon MQ for ActiveMQ** brokers that use Amazon Elastic File System (EFS) to store message data, it may take several hours for permissions to use the KMS keys in your account to be revoked after taking the required actions.
- For **Amazon MQ for RabbitMQ** and **Amazon MQ for ActiveMQ** brokers that use EBS to store message data, if you disable, schedule for deletion, or revoke the grant that gives Amazon EBS permission to use the KMS keys in your account, Amazon MQ cannot maintain your broker, and it may change to a degraded state.
- If you have deactivated the key or scheduled the key to be deleted, you can reactivate the key or cancel key deletion and keep your broker maintained.
- It may take serveral hours to deactivate a key or revoke a grant after taking the required actions.
- For encrypting or decrypting CloudWatch logs, you cannot configure what Amazon MQ uses for your encryption key. CloudWatch logs protects data at rest using encryption, and log groups are encrypted. The CloudWatch logs service manages the server-side encryption by defauly. For more information on how log groups are encrypted, see the *Amazon CloudWatch Logs User Guide*.

When creating a single instance broker with a KMS key for RabbitMQ, you will see two `CreateGrant` events logged in AWS CloudTrail. The first event is Amazon MQ creating a grant for the KMS key. The second event is EBS creating a grant for EBS to use.

CreateGrant AWS CloudTrail log entry: single instance broker

mq_grant


```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::111122223333:user/AmazonMqConsole",
        "accountId": "111122223333",
        "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
```

```
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AKIAIOSFODNN7EXAMPLE",
                "arn": "arn:aws:iam::111122223333:user/AmazonMqConsole",
                "accountId": "111122223333",
                "userName": "AmazonMqConsole"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2023-02-23T18:59:10Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "mq.amazonaws.com"
    },
    "eventTime": "2018-06-28T22:23:46Z",
    "eventSource": "amazonmq.amazonaws.com",
    "eventName": "CreateGrant",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "203.0.113.0",
    "userAgent": "PostmanRuntime/7.1.5",
    "requestParameters": {
        "granteePrincipal": "mq.amazonaws.com",
        "keyId": "arn:aws:kms:us-east-1:316438333700:key/bdbe42ae-f825-4e78-
a8a1-828d411c4be2",
        "retiringPrincipal": "mq.amazonaws.com",
        "operations": [
            "CreateGrant",
            "Decrypt",
            "GenerateDataKeyWithoutPlaintext",
            "ReEncryptFrom",
            "ReEncryptTo",
            "DescribeKey"
        ]
    },
    "responseElements": {
        "grantId":
 "0ab0ac0d0b000f00ea00cc0a0e00fc00bce000c000f0000000c0bc0a0000aaafSAMPLE",
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE",

    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
```

```
    "readOnly": false,
    "resources": [
        {
           "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management",
    "sessionCredentialFromConsole": "true"
}
```


EBS grant creation

You will see one event for EBS grant creation.


```
                                    {
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "mq.amazonaws.com"
    },
    "eventTime": "2023-02-23T19:09:40Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "CreateGrant",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "mq.amazonaws.com",
    "userAgent": "ExampleDesktop/1.0 (V1; OS)",
    "requestParameters": {
        "granteePrincipal": "mq.amazonaws.com",
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE",
        "constraints": {
            "encryptionContextSubset": {
                "aws:ebs:id": "vol-0b670f00f7d5417c0"
            }
        },
        "operations": [
```

```
            "Decrypt"
        ],
        "retiringPrincipal": "ec2.us-east-1.amazonaws.com"
    },
    "responseElements": {
        "grantId":
 "0ab0ac0d0b000f00ea00cc0a0e00fc00bce000c000f0000000c0bc0a0000aaafSAMPLE",
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE",
    },
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": false,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "sharedEventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventCategory": "Management"
}
```


When creating a cluster deployment with a KMS key for RabbitMQ, you will see five `CreateGrant` events logged in AWS CloudTrail. The first two events are grant creations for Amazon MQ. The next three events are grants created by EBS for EBS to use.

CreateGrant AWS CloudTrail log entry: cluster deployment

mq_grant


```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
```

```
        "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::111122223333:user/AmazonMqConsole",
        "accountId": "111122223333",
        "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AKIAIOSFODNN7EXAMPLE",
                "arn": "arn:aws:iam::111122223333:user/AmazonMqConsole",
                "accountId": "111122223333",
                "userName": "AmazonMqConsole"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2023-02-23T18:59:10Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "mq.amazonaws.com"
    },
    "eventTime": "2018-06-28T22:23:46Z",
    "eventSource": "amazonmq.amazonaws.com",
    "eventName": "CreateGrant",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "203.0.113.0",
    "userAgent": "PostmanRuntime/7.1.5",
    "requestParameters": {
        "granteePrincipal": "mq.amazonaws.com",
        "keyId": "arn:aws:kms:us-east-1:316438333700:key/bdbe42ae-f825-4e78-
a8a1-828d411c4be2",
        "retiringPrincipal": "mq.amazonaws.com",
        "operations": [
            "CreateGrant",
            "Encrypt",
            "Decrypt",
            "ReEncryptFrom",
            "ReEncryptTo",
            "GenerateDataKey",
            "GenerateDataKeyWithoutPlaintext",
            "DescribeKey"
        ]
    },
    "responseElements": {
```

```
        "grantId":
 "0ab0ac0d0b000f00ea00cc0a0e00fc00bce000c000f0000000c0bc0a0000aaafSAMPLE",
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE",

    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": false,
    "resources": [
        {
           "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management",
    "sessionCredentialFromConsole": "true"
}
```

mq_rabbit_grant

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::111122223333:user/AmazonMqConsole",
        "accountId": "111122223333",
        "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AKIAIOSFODNN7EXAMPLE",
                "arn": "arn:aws:iam::111122223333:user/AmazonMqConsole",
                "accountId": "111122223333",
                "userName": "AmazonMqConsole"
            },
```

```
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2023-02-23T18:59:10Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "mq.amazonaws.com"
    },
    "eventTime": "2018-06-28T22:23:46Z",
    "eventSource": "amazonmq.amazonaws.com",
    "eventName": "CreateGrant",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "203.0.113.0",
    "userAgent": "PostmanRuntime/7.1.5",
    "requestParameters": {
        "granteePrincipal": "mq.amazonaws.com",
        "retiringPrincipal": "mq.amazonaws.com",
        "operations": [
            "DescribeKey"
        ],
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE",
    },
    "responseElements": {
        "grantId":
 "0ab0ac0d0b000f00ea00cc0a0e00fc00bce000c000f0000000c0bc0a0000aaafSAMPLE",
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE",

    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": false,
    "resources": [
        {
           "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management",
```

```
    "sessionCredentialFromConsole": "true"
}
```

EBS grant creation

You will see three events for EBS grant creation.

```
                                    {
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "mq.amazonaws.com"
    },
    "eventTime": "2023-02-23T19:09:40Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "CreateGrant",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "mq.amazonaws.com",
    "userAgent": "ExampleDesktop/1.0 (V1; OS)",
    "requestParameters": {
        "granteePrincipal": "mq.amazonaws.com",
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE",
        "constraints": {
            "encryptionContextSubset": {
                "aws:ebs:id": "vol-0b670f00f7d5417c0"
            }
        },
        "operations": [
            "Decrypt"
        ],
        "retiringPrincipal": "ec2.us-east-1.amazonaws.com"
    },
    "responseElements": {
        "grantId":
 "0ab0ac0d0b000f00ea00cc0a0e00fc00bce000c000f0000000c0bc0a0000aaafSAMPLE",
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE",
    },
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": false,
```

```
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "sharedEventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventCategory": "Management"
}
```

For more information about KMS keys, see AWS KMS keys in the *AWS Key Management Service Developer Guide*.

Encryption in transit

Amazon MQ for ActiveMQ: Amazon MQ for ActiveMQ requires strong Transport Layer Security (TLS) and encrypts data in transit between the brokers of your Amazon MQ deployment. All data that passes between Amazon MQ brokers is encrypted using strong Transport Layer Security (TLS). This is true for all available protocols.

Amazon MQ for RabbitMQ: Amazon MQ for RabbitMQ requires strong Transport Layer Security (TLS) encryption for all client connections. RabbitMQ cluster replication traffic only transits your broker's VPC and all network traffic between AWS data centers is transparently encrypted at the physical layer. Amazon MQ for RabbitMQ clustered brokers currently do not support Inter-node encryption for cluster replication. To learn more about data-in-transit, see Encrypting Data-at-Rest and -in-Transit.

Amazon MQ for ActiveMQ protocols

You can access your ActiveMQ brokers using the following protocols with TLS enabled:

- AMQP
- MQTT
- MQTT over WebSocket

- OpenWire
- STOMP
- STOMP over WebSocket

Supported TLS Cipher Suites for ActiveMQ

ActiveMQ on Amazon MQ supports the following cipher suites:

- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
- TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
- TLS_DHE_RSA_WITH_AES_256_CBC_SHA
- TLS_RSA_WITH_AES_256_GCM_SHA384
- TLS_RSA_WITH_AES_256_CBC_SHA256
- TLS_RSA_WITH_AES_256_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
- TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
- TLS_DHE_RSA_WITH_AES_128_CBC_SHA
- TLS_RSA_WITH_AES_128_GCM_SHA256
- TLS_RSA_WITH_AES_128_CBC_SHA256
- TLS_RSA_WITH_AES_128_CBC_SHA

Amazon MQ for RabbitMQ protocols

You can access your RabbitMQ brokers using the following protocols with TLS enabled:

- AMQP (0-9-1)

Supported TLS Cipher Suites for RabbitMQ

RabbitMQ on Amazon MQ supports the following cipher suites:

- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256

Identity and access Management for Amazon MQ

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

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 MQ identity and access)
- **Service administrator** - determine user access and submit permission requests (see How Amazon MQ works with IAM)

- **IAM administrator** - write policies to manage access (see Amazon MQ Identity-based policy examples)

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*.

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.

Access Control Lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the *Amazon Simple Storage Service Developer Guide*.

Other policy types

AWS supports additional 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 MQ works with IAM

Before you use IAM to manage access to Amazon MQ, you should understand what IAM features are available to use with Amazon MQ. To get a high-level view of how Amazon MQ and other AWS services work with IAM, see AWS Services That Work with IAM in the *IAM User Guide*.

Amazon MQ uses IAM for Amazon MQ API operations to create, update, delete, and list brokers. For broker access to publish and subscribe to messages, Amazon MQ for ActiveMQ supports native ActiveMQ authentication and LDAP, while Amazon MQ for RabbitMQ supports IAM authentication and other methods. For more information, see the section called “Broker authentication and authorization”.

Topics

Amazon MQ identity-based policies

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. Amazon MQ supports specific actions, resources, and condition keys. To learn about all of the elements that you use in a JSON policy, see IAM JSON Policy Elements Reference in the *IAM User Guide*.

Actions

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Include actions in a policy to grant permissions to perform the associated operation.

Policy actions in Amazon MQ use the following prefix before the action: `mq:`. For example, to grant someone permission to run an Amazon MQ instance with the Amazon MQ `CreateBroker` API operation, you include the `mq:CreateBroker` action in their policy. Policy statements must include either an `Action` or `NotAction` element. Amazon MQ defines its own set of actions that describe tasks that you can perform with this service.

To specify multiple actions in a single statement, separate them with commas as follows:

```
"Action": [
      "mq:action1",
```

```
      "mq:action2"
```

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

```
"Action": "mq:Describe*"
```

To see a list of Amazon MQ actions, see Actions Defined by Amazon MQ in the *IAM User Guide*.

Resources

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": "*"
```

In the Amazon MQ, the primary AWS resources are an Amazon MQ message broker and its configuration. Amazon MQ brokers and configurations each have unique Amazon Resource Names (ARNs) associated with them, as shown in the following table.

Resource Types	ARN	Condition Keys
brokers	`arn:aws:mq:us-east-1:123456789012:broker:${brokerName}:${brokerId}`	aws:ResourceTag/${TagKey}
configurations	`arn:${Partition}:mq:${Region}:${Account}:configuration:${configuration-id}`	aws:ResourceTag/${TagKey}

For more information about the format of ARNs, see Amazon Resource Names (ARNs) and AWS Service Namespaces.

For example, to specify the broker named `MyBroker` with brokerId `b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9` in your statement, use the following ARN:

```
"Resource": "arn:aws:mq:us-
east-1:123456789012:broker:MyBroker:b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9"
```

To specify all brokers and configurations that belong to a specific account, use the wildcard (*):

```
"Resource": "arn:aws:mq:us-east-1:123456789012:*"
```

Some Amazon MQ actions, such as those for creating resources, cannot be performed on a specific resource. In those cases, you must use the wildcard (*).

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

The API action `CreateTags` requires both a broker and a configuration. To specify multiple resources in a single statement, separate the ARNs with commas.

```
"Resource": [
      "resource1",
      "resource2"
```

To see a list of Amazon MQ resource types and their ARNs, see Resources Defined by Amazon MQ in the *IAM User Guide*. To learn with which actions you can specify the ARN of each resource, see Actions Defined by Amazon MQ.

Condition keys

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 MQ does not define any service-specific condition keys, but supports using some global condition keys. To see a list of Amazon MQ condition keys, see the table below or Condition Keys for Amazon MQ in the *IAM User Guide*. To learn with which actions and resources you can use a condition key, see Actions Defined by Amazon MQ.

Condition Keys	Description	Type
aws:RequestTag/${TagKey}	Filters actions based on the tags that are passed in the request.	String
aws:ResourceTag/${TagKey}	Filters actions based on the tags associated with the resource.	String
aws:TagKeys	Filters actions based on the tag keys that are passed in the request.	String

Examples

To view examples of Amazon MQ identity-based policies, see Amazon MQ Identity-based policy examples.

Amazon MQ Resource-based policies

Currently, Amazon MQ doesn't support IAM authentication using resource-based permissions or resource-based policies.

Authorization based on Amazon MQ tags

You can attach tags to Amazon MQ resources or pass tags in a request to Amazon MQ. To control access based on tags, you provide tag information in the condition element of a policy using the `mq:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

Amazon MQ supports policies based on tags. For instance, you could deny access to Amazon MQ resources that include a tag with the key `environment` and the value `production`:

JSON

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

```
            "Action": [
                "mq:DeleteBroker",
                "mq:RebootBroker",
                "mq:DeleteTags"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/environment": "production"
                }
            }
        }
    ]
}
```

This policy will Deny the ability to delete or reboot an Amazon MQ broker that includes the tag `environment/production`.

For more information on tagging, see:

- Adding tags to Amazon MQ resources
- Controlling Access Using IAM Tags

Amazon MQ IAM roles

An IAM role is an entity within your AWS account that has specific permissions.

Using Temporary Credentials with Amazon MQ

You can use temporary credentials to sign in with federation, assume an IAM role, or to assume a cross-account role. You obtain temporary security credentials by calling AWS STS API operations such as AssumeRole or GetFederationToken.

Amazon MQ supports using temporary credentials.

Service roles

This feature allows a service to assume a service role on your behalf. This role allows the service to access resources in other services to complete an action on your behalf. Service roles appear in your IAM account and are owned by the account. This means that an IAM administrator can change the permissions for this role. However, doing so might break the functionality of the service.

Amazon MQ supports service roles.

Amazon MQ Identity-based policy examples

By default, users and roles don't have permission to create or modify Amazon MQ resources. They also can't perform tasks using the AWS Management Console, AWS CLI, or AWS API. An IAM administrator must create IAM policies that grant users and roles permission to perform specific API operations on the specified resources they need. The administrator must then attach those policies to the IAM users or groups that require those permissions.

To learn how to create an IAM identity-based policy using these example JSON policy documents, see Creating Policies on the JSON Tab in the *IAM User Guide*.

Topics

Policy best practices

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

Using the Amazon MQ console

To access the Amazon MQ console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the Amazon MQ resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (IAM users or roles) with that policy.

To ensure that those entities can still use the Amazon MQ console, also attach the following AWS managed policy to the entities. For more information, see Adding Permissions to a User in the *IAM User Guide*:

```
AmazonMQReadOnlyAccess
```

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that you're trying to perform.

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": "*"
        }
    ]
}
```

API authentication and authorization for Amazon MQ

Amazon MQ uses standard AWS request signing for API authentication. For more information, see Signing AWS API Requests in the *AWS General Reference*.

> **Note**
>
> Currently, Amazon MQ doesn't support IAM authentication using resource-based permissions or resource-based policies.

To authorize AWS users to work with brokers, configurations, and users, you must edit your IAM policy permissions.

Topics

IAM Permissions Required to Create an Amazon MQ Broker

To create a broker, you must either use the `AmazonMQFullAccess` IAM policy or include the following EC2 permissions in your IAM policy.

The following custom policy is comprised of two statements (one conditional) which grant permissions to manipulate the resources which Amazon MQ requires to create an ActiveMQ broker.

> **Important**
>
> - The `ec2:CreateNetworkInterface` action is required to allow Amazon MQ to create an elastic network interface (ENI) in your account on your behalf.
> - The `ec2:CreateNetworkInterfacePermission` action authorizes Amazon MQ to attach the ENI to an ActiveMQ broker.
> - The `ec2:AuthorizedService` condition key ensures that ENI permissions can be granted only to Amazon MQ service accounts.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [{
        "Action": [
            "mq:*",
            "ec2:CreateNetworkInterface",
            "ec2:DeleteNetworkInterface",
            "ec2:DetachNetworkInterface",
            "ec2:DescribeInternetGateways",
            "ec2:DescribeNetworkInterfaces",
            "ec2:DescribeRouteTables",
            "ec2:DescribeSecurityGroups",
            "ec2:DescribeSubnets",
            "ec2:DescribeVpcs"
        ],
        "Effect": "Allow",
        "Resource": "*"
    },{
        "Action": [
            "ec2:CreateNetworkInterfacePermission",
            "ec2:DeleteNetworkInterfacePermission",
            "ec2:DescribeNetworkInterfacePermissions"
        ],
        "Effect": "Allow",
        "Resource": "*",
        "Condition": {
            "StringEquals": {
                "ec2:AuthorizedService": "mq.amazonaws.com"
            }
        }
    }]
}
```

For more information, see Step 2: create a user and get your AWS credentials and Never Modify or Delete the Amazon MQ Elastic Network Interface.

Amazon MQ REST API permissions reference

The following table lists Amazon MQ REST APIs and the corresponding IAM permissions.

Amazon MQ REST APIs and Required Permissions

Amazon MQ REST APIs	Required Permissions
CreateBroker	mq:CreateBroker
CreateConfiguration	mq:CreateConfiguration
CreateTags	mq:CreateTags
CreateUser	mq:CreateUser
DeleteBroker	mq:DeleteBroker
DeleteUser	mq:DeleteUser
DescribeBroker	mq:DescribeBroker
DescribeConfiguration	mq:DescribeConfiguration
DescribeConfigurationRevision	mq:DescribeConfigurationRevision
DescribeUser	mq:DescribeUser
ListBrokers	mq:ListBrokers
ListConfigurationRevisions	mq:ListConfigurationRevisions
ListConfigurations	mq:ListConfigurations
ListTags	mq:ListTags
ListUsers	mq:ListUsers
RebootBroker	mq:RebootBroker
UpdateBroker	mq:UpdateBroker
UpdateConfiguration	mq:UpdateConfiguration
UpdateUser	mq:UpdateUser

Amazon MQ additional permissions reference

The following table lists the Amazon MQ API and the additional IAM permission required for specific features, such as OAuth 2.0 authentication.

Amazon MQ REST API	Permission	Description
UpdateBroker	`mq:UpdateBrokerAccessConfiguration`	You need this permission to update authentication and authorization options in the associated broker configuration. For more information, see OAuth 2.0 authentication and authorization for Amazon MQ for RabbitMQ.

Resource-level permissions for Amazon MQ API actions

The term *resource-level permissions* refers to the ability to specify the resources on which users are allowed to perform actions. Amazon MQ has partial support for resource-level permissions. For certain Amazon MQ actions, you can control when users are allowed to use those actions based on conditions that have to be fulfilled, or specific resources that users are allowed to use.

The following table describes the Amazon MQ API actions that currently support resource-level permissions, as well as the supported resources, resource ARNs, and condition keys for each action.

Important

If an Amazon MQ API action is not listed in this table, then it does not support resource-level permissions. If an Amazon MQ API action does not support resource-level permissions, you can grant users permission to use the action, but you have to specify a * wildcard for the resource element of your policy statement.

API Action	Resource Types (*required)
CreateConfiguration	configurations*

API Action	Resource Types (*required)
CreateTags	brokers, configurations
CreateUser	brokers*
DeleteBroker	brokers*
DeleteUser	brokers*
DescribeBroker	brokers*
DescribeConfiguration	configurations*
DescribeConfigurationRevision	configurations*
DescribeUser	brokers*
ListConfigurationRevisions	configurations*
ListConfigurationRevisions	configurations*
ListTags	brokers, configurations
ListUsers	brokers*
RebootBroker	brokers*
UpdateBroker	brokers*
UpdateConfiguration	configurations*
UpdateUser	brokers*

Broker authentication and authorization

Amazon MQ provides different authentication and authorization methods depending on your broker engine type.

Authentication and authorization for Amazon MQ for ActiveMQ

Amazon MQ for ActiveMQ supports the following authentication and authorization methods:

Simple authentication and authorization

In this method, broker users are created and managed through the Amazon MQ console or API. Users can be configured with specific permissions to access queues, topics, and the ActiveMQ Web Console. For more information about this method, see Creating an ActiveMQ broker user.

LDAP authentication and authorization

In this method, broker users authenticate through credentials stored in your LDAP server. You can add, delete, and modify users and assign permissions to topics and queues through the LDAP server, providing centralized authentication and authorization. For more information about this method, see Integrating ActiveMQ brokers with LDAP.

Authentication and authorization for Amazon MQ for RabbitMQ

Amazon MQ for RabbitMQ supports the following authentication and authorization methods:

Simple authentication and authorization

In this method, broker users are stored internally in the RabbitMQ broker and managed through the web console or management API. Permissions for vhosts, exchanges, queues, and topics are configured directly in RabbitMQ. This is the default method. For more information, see Simple authentication and authorization.

OAuth 2.0 authentication and authorization

In this method, broker users and their permissions are managed by an external OAuth 2.0 identity provider (IdP). User authentication and resource permissions for vhosts, exchanges, queues, and topics are centralized through the OAuth 2.0 provider's scope system. This simplifies user management and enables integration with existing identity systems. For more information, see OAuth 2.0 authentication and authorization.

IAM authentication and authorization

In this method, broker users authenticate using AWS IAM credentials through IAM outbound federation. IAM credentials are used to obtain JWT tokens from AWS Security Token Service (STS), and these JWT tokens serve as OAuth 2.0 tokens for authentication. This method leverages the existing OAuth 2.0 support in Amazon MQ for RabbitMQ, where AWS acts as the OAuth 2.0 identity provider. User authentication is handled by AWS IAM, while resource permissions for vhosts, exchanges, queues, and topics are managed through IAM policies and scope aliases configured in RabbitMQ. For more information, see IAM authentication and authorization.

LDAP authentication and authorization

In this method, broker users and their permissions are managed by an external LDAP directory service. User authentication and resource permissions are centralized through the LDAP server, allowing users to access RabbitMQ using their existing directory service credentials. For more information, see LDAP authentication and authorization.

HTTP authentication and authorization

In this method, broker users and their permissions are managed by an external HTTP server. User authentication and resource permissions are centralized through the HTTP server, allowing users to access RabbitMQ using their own Authentication and Authorization provider. For more information about this method, see HTTP authentication and authorization.

SSL certificate authentication

Amazon MQ supports mutual TLS (mTLS) for RabbitMQ brokers. The SSL authentication plugin uses client certificates from mTLS connections to authenticate users. In this method, broker users are authenticated using X.509 client certificates instead of username and password credentials. The client's certificate is validated against a trusted Certificate Authority (CA), and the username is extracted from a field in the certificate, such as the Common Name (CN) or Subject Alternative Name (SAN). This method provides strong authentication without transmitting credentials over the network. For more information, see SSL certificate authentication.

> **Note**
>
> RabbitMQ supports multiple authentication and authorization methods to be used simultaneously. For example, you can enable both OAuth 2.0 and simple (internal) authentication. For more information, see the OAuth 2.0 tutorial section on enabling

both OAuth 2.0 and simple (internal) authentication and the RabbitMQ access control documentation.
Amazon MQ recommends creating an internal user when testing authentication configurations. This allows access configuration to be validated using RabbitMQ management API. For more information, see Access validation.

AWS managed policies for Amazon MQ

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*.

Amazon MQ supports the following AWS managed policies:

- AmazonMQApiFullAccess
- AmazonMQApiReadOnlyAccess
- AmazonMQFullAccess
- AmazonMQReadOnlyAccess
- AmazonMQServiceRolePolicy

AWS managed policy: AmazonMQServiceRolePolicy

You can't attach `AmazonMQServiceRolePolicy` to your IAM entities. This policy is attached to a service-linked role that allows Amazon MQ to perform actions on your behalf. For more information about this permission policy and the actions it allows Amazon MQ to perform, see the section called “Service-linked role permissions for Amazon MQ”.

Amazon MQ updates to AWS managed policies

View details about updates to AWS managed policies for Amazon MQ since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the Amazon MQ Document history page.

Change	Description	Date
Amazon MQ started tracking changes	Amazon MQ started tracking changes for its AWS managed policies.	May 5, 2021

Using service-linked roles for Amazon MQ

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

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

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

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

Service-linked role permissions for Amazon MQ

Amazon MQ uses the service-linked role named **AWSServiceRoleForAmazonMQ** – Amazon MQ uses this service-linked role to call AWS services on your behalf.

The AWSServiceRoleForAmazonMQ service-linked role trusts the following services to assume the role:

- `mq.amazonaws.com`

Amazon MQ uses the permission policy `AmazonMQServiceRolePolicy`, which is attached to the AWSServiceRoleForAmazonMQ service-linked role, to complete the following actions on the specified resources:

- Action: `ec2:CreateVpcEndpoint` on the `vpc` resource.
- Action: `ec2:CreateVpcEndpoint` on the `subnet` resource.
- Action: `ec2:CreateVpcEndpoint` on the `security-group` resource.
- Action: `ec2:CreateVpcEndpoint` on the `vpc-endpoint` resource.
- Action: `ec2:DescribeVpcEndpoints` on the `vpc` resource.
- Action: `ec2:DescribeVpcEndpoints` on the `subnet` resource.
- Action: `ec2:CreateTags` on the `vpc-endpoint` resource.
- Action: `logs:PutLogEvents` on the `log-group` resource.
- Action: `logs:DescribeLogStreams` on the `log-group` resource.
- Action: `logs:DescribeLogGroups` on the `log-group` resource.
- Action: `CreateLogStream` on the `log-group` resource.
- Action: `CreateLogGroup` on the `log-group` resource.

When you create an Amazon MQ for RabbitMQ broker, the `AmazonMQServiceRolePolicy` permission policy allows Amazon MQ to perform the following tasks on your behalf.

- Create a Amazon VPC endpoint for the broker using the Amazon VPC, subnet, and security-group you provide. You can use the endpoint created for your broker to connect to the broker via the RabbitMQ management console, the management API, or programatically.
- Create log groups, and publish broker logs to Amazon CloudWatch Logs.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeVpcEndpoints"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateVpcEndpoint"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:vpc/*",
                "arn:aws:ec2:*:*:subnet/*",
                "arn:aws:ec2:*:*:security-group/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateVpcEndpoint"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:vpc-endpoint/*"
            ],
            "Condition": {
                "StringEquals": {
```

```
                    "aws:RequestTag/AMQManaged": "true"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateTags"
            ],
            "Resource": "arn:aws:ec2:*:*:vpc-endpoint/*",
            "Condition": {
                "StringEquals": {
                    "ec2:CreateAction": "CreateVpcEndpoint"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DeleteVpcEndpoints"
            ],
            "Resource": "arn:aws:ec2:*:*:vpc-endpoint/*",
            "Condition": {
                "StringEquals": {
                    "ec2:ResourceTag/AMQManaged": "true"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents",
                "logs:DescribeLogStreams",
                "logs:DescribeLogGroups",
                "logs:CreateLogStream",
                "logs:CreateLogGroup"
            ],
            "Resource": [
                "arn:aws:logs:*:*:log-group:/aws/amazonmq/*"
            ]
        }
    ]
}
```

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

Creating a service-linked role for Amazon MQ

You don't need to manually create a service-linked role. When you first create a broker, Amazon MQ creates a service-linked role to call AWS services on your behalf. All subsequent brokers that you create will use the same role and no new role is created.

> **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, see A New Role Appeared in My IAM 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.

You can also use the IAM console to create a service-linked role with the **Amazon MQ** use case. In the AWS CLI or the AWS API, create a service-linked role with the `mq.amazonaws.com` service name. For more information, see 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.

> **Important**
>
> Service Linked Roles are only created for Amazon MQ for RabbitMQ.

Editing a service-linked role for Amazon MQ

Amazon MQ does not allow you to edit the AWSServiceRoleForAmazonMQ service-linked role. However, you can edit the description of the role using IAM. For more information, see Editing a service-linked role in the *IAM User Guide*.

Deleting a service-linked role for Amazon MQ

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.

> **Note**
>
> If the Amazon MQ service is using the role when you try to delete the resources, then the deletion might fail. If that happens, wait for a few minutes and try the operation again.

To delete Amazon MQ resources used by the AWSServiceRoleForAmazonMQ

- Delete your Amazon MQ brokers using the AWS Management Console, Amazon MQ CLI, or Amazon MQ API. For more information about deleting brokers, see ???.

To manually delete the service-linked role using IAM

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

Supported regions for Amazon MQ service-linked roles

Amazon MQ supports using service-linked roles in all of the regions where the service is available. For more information, see AWS Regions and Endpoints.

Region name	Region identity	Support in Amazon MQ
US East (N. Virginia)	us-east-1	Yes
US East (Ohio)	us-east-2	Yes
US West (N. California)	us-west-1	Yes
US West (Oregon)	us-west-2	Yes
Asia Pacific (Mumbai)	ap-south-1	Yes
Asia Pacific (Osaka)	ap-northeast-3	Yes
Asia Pacific (Seoul)	ap-northeast-2	Yes

Region name	Region identity	Support in Amazon MQ
Asia Pacific (Singapore)	ap-southeast-1	Yes
Asia Pacific (Sydney)	ap-southeast-2	Yes
Asia Pacific (Tokyo)	ap-northeast-1	Yes
Canada (Central)	ca-central-1	Yes
Europe (Frankfurt)	eu-central-1	Yes
Europe (Ireland)	eu-west-1	Yes
Europe (London)	eu-west-2	Yes
Europe (Paris)	eu-west-3	Yes
South America (São Paulo)	sa-east-1	Yes
AWS GovCloud (US)	us-gov-west-1	No

Troubleshooting Amazon MQ identity and access

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

Topics

- I Am Not Authorized to Perform an Action in Amazon MQ
- I am not authorized to perform iam:PassRole
- I want to allow people outside of my AWS account to access my Amazon MQ resources

I Am Not Authorized to Perform an Action in Amazon MQ

If the AWS Management Console tells you that you're not authorized to perform an action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your sign-in credentials.

The following example error occurs when the `mateojackson` user tries to use the console to view details about a *`widget`* but does not have `mq:`*`GetWidget`* permissions.

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

In this case, Mateo asks his administrator to update his policies to allow him to access the *`my-example-widget`* resource using the `mq:`*`GetWidget`* action.

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 MQ.

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 MQ. 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 MQ 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 MQ supports these features, see How Amazon MQ 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 MQ

Third-party auditors assess the security and compliance of Amazon MQ as part of multiple AWS compliance programs. These include SOC, PCI, HIPAA, and others.

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 MQ

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, see AWS Global Infrastructure.

Infrastructure security in Amazon MQ

As a managed service, Amazon MQ is protected by AWS global network security. For information about AWS security services and how AWS protects infrastructure, see AWS Cloud Security. To design your AWS environment using the best practices for infrastructure security, see Infrastructure Protection in *Security Pillar AWS Well-Architected Framework*.

You use AWS published API calls to access Amazon MQ through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

Security best practices for Amazon MQ

The following design patterns can improve the security of your Amazon MQ broker.

Topics

For more information about how Amazon MQ encrypts your data, as well as a list of supported protocols, see Data Protection.

Prefer brokers without public accessibility

Brokers created without public accessibility can't be accessed from outside of your VPC. This greatly reduces your broker's susceptibility to Distributed Denial of Service (DDoS) attacks from the public internet. For more information, see How to Help Prepare for DDoS Attacks by Reducing Your Attack Surface on the AWS Security Blog.

Always configure an authorization map

Because ActiveMQ has no authorization map configured by default, any authenticated user can perform any action on the broker. Thus, it is a best practice to restrict permissions *by group*. For more information, see `authorizationEntry`.

> **Important**
>
> If you specify an authorization map which doesn't include the `activemq-webconsole` group, you can't use the ActiveMQ Web Console because the group isn't authorized to send messages to, or receive messages from, the Amazon MQ broker.

Block unnecessary protocols with VPC security groups

To improve security for private brokers, you should restrict the connections of unnecessary protocols and ports by properly configuring your Amazon VPC Security Group. For instance, to restrict access to most protocols while allowing access to OpenWire and the web console, you could allow access to only 61617 and 8162. This limits your exposure by blocking protocols you are not using, while allowing OpenWire and the web console to function normally.

Allow only the protocol ports that you are using.

- AMQP: 5671
- MQTT: 8883
- OpenWire: 61617
- STOMP: 61614
- WebSocket: 61619

For more information see:

- Security Groups for your VPC
- Default Security Group for Your VPC
- Working with Security Groups

Logging and monitoring Amazon MQ brokers

Monitoring is an important part of maintaining the reliability, availability, and performance of your AWS solutions. You should collect monitoring data from all of the parts of your AWS solution so that you can more easily debug a multi-point failure if one occurs. AWS provides several tools for monitoring your Amazon MQ resources and responding to potential incidents:

You can use CloudWatch to view and analyze metrics for your Amazon MQ broker. You can view and analyze your broker metrics from the CloudWatch console, the AWS CLI, or the CloudWatch AWS CLI. CloudWatch metrics for Amazon MQ are automatically polled from the broker and then pushed to CloudWatch every minute. For ActiveMQ brokers, CloudWatch monitors only the first 1000 destinations.. For RabbitMQ brokers, CloudWatch monitors only the first 500 destinations, ordered by number of consumers..

For a full list of Amazon MQ metrics, see Available CloudWatch metrics Amazon MQ for ActiveMQ brokers.

For information about creating a CloudWatch alarm for a metrics, see Create or Edit a CloudWatch Alarm in the *Amazon CloudWatch User Guide*.

Accessing CloudWatch metrics for Amazon MQ

You can access CloudWatch metrics using the AWS Management Console, AWS CLI, and API.

You may want to access CloudWatch metrics without using the AWS Management Console.

To access Amazon MQ metrics using the AWS CLI, use the `get-metric-statistics` command. For more information, see Get Statistics for a Metric in the *Amazon CloudWatch User Guide*.

To access Amazon MQ metrics using the CloudWatch API, use the `GetMetricStatistics` action. For more information, see Get Statistics for a Metric in the *Amazon CloudWatch User Guide*.

Accesing CloudWatch metrics using the AWS Management Console

The following example shows you how to access CloudWatch metrics for Amazon MQ using the AWS Management Console.If you're already signed into the Amazon MQ console, on the broker **Details** page, choose **Actions**, **View CloudWatch metrics**.

1. Sign in to the CloudWatch console.
2. On the navigation panel, choose **Metrics**.
3. Select the **AmazonMQ** metric namespace.
4. Select one of the following metric dimensions:

   - **Broker Metrics**
   - **Queue Metrics by Broker**
   - **Topic Metrics by Broker**

   In this example, **Broker Metrics** is selected.
5. You can now examine your Amazon MQ metrics:

   - To sort the metrics, use the column heading.
   - To graph the metric, select the check box next to the metric.
   - To filter by metric, choose the metric name and then choose **Add to search**.

Available CloudWatch metrics Amazon MQ for ActiveMQ brokers

Amazon MQ for ActiveMQ metrics

Metric	Unit	Description
`AmqpMaximumConnections`	Count	The maximum number of clients you can connect to your broker using AMQP. For more information on connection quotas, see Quotas in Amazon MQ.
`BurstBalance`	Percent	The percentage of burst credits remaining on the Amazon EBS volume used

Metric	Unit	Description
		to persist message data for throughput-optimized brokers. If this balance reaches zero, the IOPS provided by the Amazon EBS volume will decrease until the Burst Balance refills. For more information on how Burst Balances work in Amazon EBS, see: I/O Credits and Burst Performance.

Metric	Unit	Description
CpuCreditBalance	Credits (vCPU-minutes)	**Important** This metric is available only for the mq.t2.micro broker instance type. CPU credit metrics are available only at five-minute intervals. The number of earned CPU credits that an instance has accrued since it was launched or started (including the number of launch credits). The credit balance is available for the broker instance to spend on bursts beyond the baseline CPU utilization. Credits are accrued in the credit balance after they're earned and removed from the credit balance after they're spent. The credit balance has a maximum limit. Once the limit is reached, any newly earned credits are discarded.
CpuUtilization	Percent	The percentage of allocated Amazon EC2 compute units that the broker currently uses.

Metric	Unit	Description
CurrentConnectionsCount	Count	The current number of active connections on the current broker.
EstablishedConnectionsCount	Count	The total number of connections, active and inactive, that have been established on the broker.
HeapUsage	Percent	The percentage of the ActiveMQ JVM memory limit that the broker currently uses.
InactiveDurableTopicSubscribersCount	Count	The number of inactive durable topic subscribers, up to a maximum of 2000.
JobSchedulerStorePercentUsage	Percent	The percentage of disk space used by the job scheduler store.
JournalFilesForFastRecovery	Count	The number of journal files that will be replayed after a clean shutdown.
JournalFilesForFullRecovery	Count	The number of journal files that will be replayed after an unclean shutdown.
MqttMaximumConnections	Count	The maximum number of clients you can connect to your broker using MQTT. For more information on connection quotas, see Quotas in Amazon MQ.

Metric	Unit	Description
NetworkConnectorConnectionCount	Count	The number of nodes connected to the broker in a network of brokers using NetworkConnector.
NetworkIn	Bytes	The volume of incoming traffic for the broker.
NetworkOut	Bytes	The volume of outgoing traffic for the broker.
OpenTransactionCount	Count	The total number of transactions in progress.
OpenwireMaximumConnections	Count	The maximum number of clients you can connect to your broker using OpenWire. For more information on connection quotas, see Quotas in Amazon MQ.
StompMaximumConnections	Count	The maximum number of clients you can connect to your broker using STOMP. For more information on connection quotas, see Quotas in Amazon MQ.
StorePercentUsage	Percent	The percent used by the storage limit. If this reaches 100, the broker will refuse messages.
TempPercentUsage	Percent	The percentage of available temporary storage used by non-persistent messages.

Metric	Unit	Description
TotalConsumerCount	Count	The number of message consumers subscribed to destinations on the current broker.
TotalMessageCount	Count	The number of messages stored on the broker.
TotalProducerCount	Count	The number of message producers active on destinati ons on the current broker.
VolumeReadOps	Count	The number of read operation s performed on the Amazon EBS volume.
VolumeWriteOps	Count	The number of write operations performed on the Amazon EBS volume.
WsMaximumConnections	Count	The maximum number of clients you can connect to your broker using WebSocket . For more information on connection quotas, see Quotas in Amazon MQ.

Dimensions for ActiveMQ broker metrics

Dimension	Description
Broker	The name of the broker

Dimension	Description
	Note A single-instance broker has the suffix -1. An active/standby broker for high availability has the suffixes -1 and -2 for its redundant pair.

ActiveMQ destination (queue and topic) metrics

Important

The following metrics include per-minute counts for the CloudWatch polling period.

- `EnqueueCount`
- `ExpiredCount`
- `DequeueCount`
- `DispatchCount`
- `InFlightCount`

For example, in a five-minute CloudWatch period, `EnqueueCount` has five count values, each for a one-minute portion of the period. The `Minimum` and `Maximum` statistics provide the lowest and highest per-minute value during the specified period.

Metric	Unit	Description
`ConsumerCount`	Count	The number of consumers subscribed to the destination.
`EnqueueCount`	Count	The number of messages sent to the destination, per minute.

Metric	Unit	Description
EnqueueTime	Time (milliseconds)	The end-to-end latency from when a message arrives at a broker until it is delivered to a consumer. **Note** EnqueueTime does not measure the end-to-end latency from when a message is sent by a producer until it reaches the broker, nor the latency from when a message is received by a broker until it is acknowled ged by the broker. Rather, EnqueueTi me is the number of milliseconds from the moment a message is received by the broker until it is successfu lly delivered to a consumer.
ExpiredCount	Count	The number of messages that couldn't be delivered because they expired, per minute.
DispatchCount	Count	The number of messages sent to consumers, per minute.

Metric	Unit	Description
DequeueCount	Count	The number of messages acknowledged by consumers, per minute.
InFlightCount	Count	The number of messages sent to consumers that have not been acknowledged.
ReceiveCount	Count	The number of messages that have been received from the remote broker for a duplex network connector.
MemoryUsage	Percent	The percentage of the memory limit that the destination currently uses.
ProducerCount	Count	The number of producers for the destination.
QueueSize	Count	The number of messages in the queue. **Important** This metric applies only to queues.
TotalEnqueueCount	Count	The total number of messages that have been sent to the broker.
TotalDequeueCount	Count	The total number of messages that have been consumed by clients.

> **Note**
>
> `TotalEnqueueCount` and `TotalDequeueCount` metrics include messages for advisory topics. For more information about advisory topic messages, see the ActiveMQ documentation.

Dimensions for ActiveMQ destination (queue and topic) metrics

Dimension	Description
`Broker`	The name of the broker. **Note** A single-instance broker has the suffix `-1`. An active/standby broker for high availability has the suffixes `-1` and `-2` for its redundant pair.
`Topic` or `Queue`	The name of the topic or queue.
`NetworkConnector`	The name of the network connector.

Available CloudWatch metrics for Amazon MQ for RabbitMQ brokers

RabbitMQ broker metrics

Metric	Unit	Description
`ExchangeCount`	Count	The total number of exchanges configured on the broker.

Metric	Unit	Description
QueueCount	Count	The total number of queues configured on the broker.
ConnectionCount	Count	The total number of connections established on the broker.
ChannelCount	Count	The total number of channels established on the broker.
ConsumerCount	Count	The total number of consumers connected to the broker.
MessageCount	Count	The total number of messages in the queues. **Note** The number produced is the total sum of ready and unacknowl edged messages on the broker.
MessageReadyCount	Count	The total number of ready messages in the queues.
MessageUnacknowled gedCount	Count	The total number of unacknowledged messages in the queues.

Metric	Unit	Description
PublishRate	Count	The rate at which messages are published to the broker. The number produced represents the number of messages per second at the time of sampling.
ConfirmRate	Count	The rate at which the RabbitMQ server is confirming published messages. You can compare this metric with PublishRate to better understand how your broker is performing. The number produced represents the number of messages per second at the time of sampling.
AckRate	Count	The rate at which messages are being acknowledged by consumers. The number produced represents the number of messages per second at the time of sampling.

Metric	Unit	Description
SystemCpuUtilization	Percent	The percentage of allocated Amazon EC2 compute units that the broker currently uses. For cluster deployments, this value represents the aggregate of all three RabbitMQ nodes' corresponding metric values.
RabbitMQMemLimit	Bytes	The RAM limit for a RabbitMQ broker. For cluster deployments, this value represents the aggregate of all three RabbitMQ nodes' corresponding metric values.
RabbitMQMemUsed	Bytes	The volume of RAM used by a RabbitMQ broker. For cluster deployments, this value represents the aggregate of all three RabbitMQ nodes' corresponding metric values.
RabbitMQDiskFreeLimit	Bytes	The disk limit for a RabbitMQ broker. For cluster deployments, this value represents the aggregate of all three RabbitMQ nodes' corresponding metric values. This metric is different per instance size.

Metric	Unit	Description
RabbitMQDiskFree	Bytes	The total volume of free disk space available in a RabbitMQ broker. When disk usage goes above its limit, the cluster will block all producer connections. For cluster deployments, this value represents the aggregate of all three RabbitMQ nodes' corresponding metric values.
RabbitMQFdUsed	Count	Number of file descriptors used. For cluster deployments, this value represents the aggregate of all three RabbitMQ nodes' corresponding metric values.
RabbitMQIOReadAverageTime	Count	The average time (in milliseconds) for RabbitMQ to perform one read operation. The value is proportional to the message size.
RabbitMQIOWriteAverageTime	Count	The average time (in milliseconds) for RabbitMQ to perform one write operation. The value is proportional to the message size.

Dimensions for RabbitMQ broker metrics

Dimension	Description
Broker	The name of the broker.

RabbitMQ node metrics

Metric	Unit	Description
SystemCpuUtilization	Percent	The percentage of allocated Amazon EC2 compute units that the broker currently uses.
RabbitMQMemLimit	Bytes	The RAM limit for a RabbitMQ node.
RabbitMQMemUsed	Bytes	The volume of RAM used by a RabbitMQ node. When memory use goes above the limit, the cluster will block all producer connections.
RabbitMQDiskFreeLimit	Bytes	The disk limit for a RabbitMQ node. This metric is different per instance size.
RabbitMQDiskFree	Bytes	The total volume of free disk space available in a RabbitMQ node. When disk usage goes above its limit, the cluster will block all producer connections.
RabbitMQFdUsed	Count	Number of file descriptors used.

Dimensions for RabbitMQ node metrics

Dimension	Description
Node	The name of the node. **Note** A node name consists of two parts: a prefix (usuallly `rabbit`) and a hostname. For example, `rabbit@ip-10-0-0-230.us-west-2.compute.internal` is a node name with the prefix `rabbit` and the hostname `ip-10-0-0-230.us-west-2.compute.internal`.
Broker	The name of the broker.

RabbitMQ queue metrics

Metric	Unit	Description
ConsumerCount	Count	The number of consumers subscribed to the queue.
MessageReadyCount	Count	The number of messages that are currently available to be delivered.
MessageUnacknowledgedCount	Count	The number of messages for which the server is awaiting acknowledgement.

Metric	Unit	Description
MessageCount	Count	The total number of MessageReadyCount and MessageUnacknowledgedCount (also known as queue depth).

Dimensions for RabbitMQ queue metrics

> **Note**
>
> Amazon MQ for RabbitMQ will not publish metrics for virtual hosts and queues with names containing blank spaces, tabs or other non-ASCII characters.
> For more information about dimension names, see Dimension in the *Amazon CloudWatch API Reference*.

Dimension	Description
Queue	The name of the queue.
VirtualHost	The name of the virtual host.
Broker	The name of the broker.

RabbitMQ network metrics

Metric	Unit	Description
NetworkOut	Bytes	The number of bytes sent out by the instance on all network interfaces. This metric identifies the volume of outgoing network traffic from a single instance. The number reported is the number of bytes sent during the period. If you are using basic (5-minute) monitoring

Metric	Unit	Description
		and the statistic is Sum, you can divide this number by 300 to find Bytes/second. If you have detailed (1-minute) monitoring and the statistic is Sum, divide it by 60. You can also use the CloudWatch metric math function `DIFF_TIME` to find the bytes per second. For example, if you have graphed NetworkOut in CloudWatch as `m1`, the metric math formula `m1/(DIFF_TIME(m1))` returns the metric in bytes/second. For more information about `DIFF_TIME` and other metric math functions, see Using metric math. **Meaningful Statistics:** Sum, Average, Minimum, Maximum
`NetworkIn`	Bytes	The number of bytes received by the instance on all network interfaces. This metric identifies the volume of incoming network traffic to a single instance. The number reported is the number of bytes received during the period. If you are using basic (5-minute) monitoring and the statistic is Sum, you can divide this number by 300 to find Bytes/second. If you have detailed (1-minute) monitoring and the statistic is Sum, divide it by 60. You can also use the CloudWatch metric math function `DIFF_TIME` to find the bytes per second. For example, if you have graphed NetworkIn in CloudWatch as `m1`, the metric math formula `m1/(DIFF_TIME(m1))` returns the metric in bytes/second. For more information about `DIFF_TIME` and other metric math functions, see Using metric math. **Meaningful Statistics:** Sum, Average, Minimum, Maximum

Dimensions for RabbitMQ brokers

Dimension	Description
`BrokerId`	Id of the broker

Configuring Amazon MQ for RabbitMQ logs

When you enable CloudWatch logging for your RabbitMQ brokers, Amazon MQ uses a service-linked role to publish general logs to CloudWatch. If no Amazon MQ service-linked role exists when you first create a broker, Amazon MQ will automatically create one. All subsequent RabbitMQ brokers will use the same service-linked role to publish logs to CloudWatch.

For more information about service-linked roles, see Using service-linked roles in the *AWS Identity and Access Management User Guide*. For more information about how Amazon MQ uses service-linked roles, see the section called “Using service-linked roles”.

Logging Amazon MQ API calls using AWS CloudTrail

Amazon MQ is integrated with AWS CloudTrail, a service that provides a record of the Amazon MQ calls that a user, role, or AWS service makes. CloudTrail captures API calls related to Amazon MQ brokers and configurations as events, including calls from the Amazon MQ console and code calls from Amazon MQ APIs. For more information about CloudTrail, see the *AWS CloudTrail User Guide*.

> **Note**
>
> CloudTrail doesn't log API calls related to ActiveMQ operations (for example, sending and receiving messages) or to the ActiveMQ Web Console. To log information related to ActiveMQ operations, you can configure Amazon MQ to publish general and audit logs to Amazon CloudWatch Logs.

Using the information that CloudTrail collects, you can identify a specific request to an Amazon MQ API, the IP address of the requester, the requester's identity, the date and time of the request, and so on. If you configure a *trail*, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket. If you don't configure a trail, you can view the most recent events in the event

history in the CloudTrail console. For more information, see Overview for Creating a Trail in the *AWS CloudTrail User Guide*.

Amazon MQ Information in CloudTrail

When you create your AWS account, CloudTrail is enabled. When a supported Amazon MQ event activity occurs, it is recorded in a CloudTrail event with other AWS service events in the event history. You can view, search, and download recent events for your AWS account. For more information, see Viewing Events with CloudTrail Event History in the *AWS CloudTrail User Guide*.

A trail allows CloudTrail to deliver log files to an Amazon S3 bucket. You can create a trail to keep an ongoing record of events in your AWS account. By default, when you create a trail using the AWS Management Console, the trail applies to all AWS Regions. The trail logs events from all AWS Regions and delivers log files to the specified Amazon S3 bucket. You can also configure other AWS services to further analyze and act on the event data collected in CloudTrail logs. For more information, see the following topics in the *AWS CloudTrail User Guide*:

- CloudTrail Supported Services and Integrations
- Configuring Amazon SNS Notifications for CloudTrail
- Receiving CloudTrail Log Files from Multiple Regions
- Receiving CloudTrail Log Files from Multiple Accounts

Amazon MQ supports logging both the request parameters and the responses for the following APIs as events in CloudTrail log files:

- `CreateConfiguration`
- `DeleteBroker`
- `DeleteUser`
- `RebootBroker`
- `UpdateBroker`

> **Note**
>
> RebootBroker log files are logged when you reboot the broker. During the maintenance window, the service automatically reboots, and RebootBroker log files are not logged.

Important

For the GET methods of the following APIs, the request parameters are logged, but the responses are redacted:

- DescribeBroker
- DescribeConfiguration
- DescribeConfigurationRevision
- DescribeUser
- ListBrokers
- ListConfigurationRevisions
- ListConfigurations
- ListUsers

For the following APIs, the `data` and `password` request parameters are hidden by asterisks (***):

- CreateBroker (POST)
- CreateUser (POST)
- UpdateConfiguration (PUT)
- UpdateUser (PUT)

Every event or log entry contains information about the requester. This information helps you determine the following:

- Was the request made with root or user credentials?
- Was the request made with temporary security credentials for a role or a federated user?
- Was the request made by another AWS service?

For more information, see CloudTrail userIdentity Element in the *AWS CloudTrail User Guide*.

Example Amazon MQ Log File Entry

A *trail* is a configuration that allows the delivery of events as log files to the specified Amazon S3 bucket. CloudTrail log files contain one or more log entries.

An *event* represents a single request from any source and includes information about the request to an Amazon MQ API, the IP address of the requester, the requester's identity, the date and time of the request, and so on.

The following example shows a CloudTrail log entry for a `CreateBroker` API call.

> **Note**
>
> Because CloudTrail log files aren't an ordered stack trace of public APIs, they don't list information in any specific order.


```
{
    "eventVersion": "1.06",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::111122223333:user/AmazonMqConsole",
        "accountId": "111122223333",
        "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
        "userName": "AmazonMqConsole"
    },
    "eventTime": "2018-06-28T22:23:46Z",
    "eventSource": "amazonmq.amazonaws.com",
    "eventName": "CreateBroker",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "203.0.113.0",
    "userAgent": "PostmanRuntime/7.1.5",
    "requestParameters": {
        "engineVersion": "5.15.9",
        "deploymentMode": "ACTIVE_STANDBY_MULTI_AZ",
        "maintenanceWindowStartTime": {
            "dayOfWeek": "THURSDAY",
            "timeOfDay": "22:45",
            "timeZone": "America/Los_Angeles"
        },
        "engineType": "ActiveMQ",
```

```
        "hostInstanceType": "mq.m5.large",
        "users": [
            {
                "username": "MyUsername123",
                "password": "***",
                "consoleAccess": true,
                "groups": [
                    "admins",
                    "support"
                ]
            },
            {
                "username": "MyUsername456",
                "password": "***",
                "groups": [
                    "admins"
                ]
            }
        ],
        "creatorRequestId": "1",
        "publiclyAccessible": true,
        "securityGroups": [
            "sg-a1b234cd"
        ],
        "brokerName": "MyBroker",
        "autoMinorVersionUpgrade": false,
        "subnetIds": [
            "subnet-12a3b45c",
            "subnet-67d8e90f"
        ]
    },
    "responseElements": {
        "brokerId": "b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9",
        "brokerArn": "arn:aws:mq:us-
east-2:123456789012:broker:MyBroker:b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9"
    },
    "requestID": "a1b2c345-6d78-90e1-f2g3-4hi56jk7l890",
    "eventID": "a12bcd3e-fg45-67h8-ij90-12k34d5l16mn",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

Configuring Amazon MQ for ActiveMQ logs

To allow Amazon MQ to publish logs to CloudWatch Logs, you must add a permission to your Amazon MQ user and also configure a resource-based policy for Amazon MQ before you create or restart the broker.

> **Note**
>
> When you turn on logs and publish messages from the ActiveMQ web console, the content of the message is sent to CloudWatch and displayed in the logs.

The following describes the steps to configure CloudWatch logs for your ActiveMQ brokers.

Topics

Understanding the structure of logging in CloudWatch Logs

You can enable *general* and *audit* logging when you configure advanced broker settings when you create a broker, or when you edit a broker.

General logging enables the default `INFO` logging level (`DEBUG` logging isn't supported) and publishes `activemq.log` to a log group in your CloudWatch account. The log group has a format similar to the following:

```
/aws/amazonmq/broker/b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9/general
```

Audit logging enables logging of management actions taken using JMX or using the ActiveMQ Web Console and publishes `audit.log` to a log group in your CloudWatch account. The log group has a format similar to the following:

```
/aws/amazonmq/broker/b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9/audit
```

Depending on whether you have a single-instance broker or an active/standby broker, Amazon MQ creates either one or two log streams within each log group. The log streams have a format similar to the following.

```
activemq-b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.log
               activemq-b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-2.log
```

The `-1` and `-2` suffixes denote individual broker instances. For more information, see Working with Log Groups and Log Streams in the *Amazon CloudWatch Logs User Guide*.

Add the `CreateLogGroup` permission to your Amazon MQ user

To allow Amazon MQ to create a CloudWatch Logs log group, you must ensure that the user who creates or reboots the broker has the `logs:CreateLogGroup` permission.

> **Important**
>
> If you don't add the `CreateLogGroup` permission to your Amazon MQ user before the user creates or reboots the broker, Amazon MQ doesn't create the log group.

The following example IAM-based policy grants permission for `logs:CreateLogGroup` for users to whom this policy is attached.

JSON

```
{
                    "Version":"2012-10-17",
                    "Statement": [
                        {
                            "Effect": "Allow",
                            "Action": "logs:CreateLogGroup",
                            "Resource": "arn:aws:logs:*:*:log-group:/aws/
amazonmq/*"
                        }
                    ]
                    }
```

Note

Here, the term user refers to *Users* and not *Amazon MQ users*, which are created when a new broker is configured. For more information regarding setting up users and configuring IAM policies, please refer to the Identity Management Overview section of the IAM User Guide.

For more information, see `CreateLogGroup` in the *Amazon CloudWatch Logs API Reference*.

Configure a resource-based policy for Amazon MQ

Important

If you don't configure a resource-based policy for Amazon MQ, the broker can't publish the logs to CloudWatch Logs.

To allow Amazon MQ to publish logs to your CloudWatch Logs log group, configure a resource-based policy to give Amazon MQ access to the following CloudWatch Logs API actions:

- `CreateLogStream` – Creates a CloudWatch Logs log stream for the specified log group.
- `PutLogEvents` – Delivers events to the specified CloudWatch Logs log stream.

The following resource-based policy grants permission for `logs:CreateLogStream` and `logs:PutLogEvents` to AWS.

JSON

```
{
                          "Version":"2012-10-17",
                          "Statement": [
                              {
                                  "Effect": "Allow",
                                  "Principal": { "Service":
 "mq.amazonaws.com" },
                                  "Action": [ "logs:CreateLogStream",
 "logs:PutLogEvents" ],
```

```
                                    "Resource": "arn:aws:logs:*:*:log-group:/aws/
amazonmq/*"
                                }
                            ]
                        }
```

This resource-based policy *must* be configured by using the AWS CLI as shown by the following command. In the example, replace *us-east-1* with your own information.

```
aws --region us-east-1 logs put-resource-policy --policy-name AmazonMQ-logs \
                        --policy-document "{\"Version\": \"2012-10-17\", \"Statement\":
[{ \"Effect\": \"Allow\", \"Principal\": { \"Service\": \"mq.amazonaws.com\" },
                        \"Action\": [\"logs:CreateLogStream\", \"logs:PutLogEvents\"],
 \"Resource\": \"arn:aws:logs:*:*:log-group:\/aws\/amazonmq\/*\" }]}"
```

> **Note**
>
> Because this example uses the `/aws/amazonmq/` prefix, you need to configure the resource-based policy only once per AWS account, per region.

Cross-service confused deputy prevention

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

We recommend using the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in your Amazon MQ resource-based policy to limit CloudWatch Logs access to one or more specified brokers.

> **Note**
>
> If you use both global condition context keys, the `aws:SourceAccount` value and the account in the `aws:SourceArn` value must use the same account ID when used in the same policy statement.

The following example demonstrates a resource-based policy that limits CloudWatch Logs access to a single Amazon MQ broker.

JSON

```
{
                        "Version":"2012-10-17",
                        "Statement": [
                            {
                            "Effect": "Allow",
                            "Principal": {
                                "Service": "mq.amazonaws.com"
                            },
                            "Action": [
                                "logs:CreateLogStream",
                                "logs:PutLogEvents"
                            ],
                            "Resource": "arn:aws:logs:*:*:log-group:/aws/
amazonmq/*",
                            "Condition": {
                                "StringEquals": {
                                "aws:SourceAccount": "123456789012",
                                "aws:SourceArn": "arn:aws:mq:us-
west-1:123456789012:broker:my-broker:123456789012"
                                }
                            }
                            }
                        ]
                        }
```

You can also configure your resource-based policy to limit CloudWatch Logs access to all brokers in an account, as shown in the following.

JSON

```
{
                            "Version":"2012-10-17",
                            "Statement": [
                            {
                                "Effect": "Allow",
                                "Principal": {
                                "Service": [
                                    "mq.amazonaws.com"
                                ]
                                },
                                "Action": [
                                "logs:CreateLogStream",
                                "logs:PutLogEvents"
                                ],
                                "Resource": "arn:aws:logs:*:*:log-group:/aws/
amazonmq/*",
                                "Condition": {
                                "ArnLike": {
                                    "aws:SourceArn":
 "arn:aws:mq:*:123456789012:broker:*"
                                },
                                "StringEquals": {
                                    "aws:SourceAccount": "123456789012"
                                }
                                }
                            }
                            ]
                        }
```

For more information about the confused deputy security issue, see The confused deputy problem in the *User Guide*.

Troubleshooting CloudWatch Logs Configuration with Amazon MQ

In some cases, CloudWatch Logs might not always behave as expected. This section gives an overview of common issues and shows how to resolve them.

Log Groups Don't Appear in CloudWatch

Add the `CreateLogGroup` permission to your Amazon MQ user and reboot the broker. This allows Amazon MQ to create the log group.

Log Streams Don't Appear in CloudWatch Log Groups

Configure a resource-based policy for Amazon MQ. This allows your broker to publish its logs.

Quotas in Amazon MQ

This topic lists limits within Amazon MQ. Many of the following limits can be changed for specific AWS accounts. To request an increase for a limit, see AWS Service Quotas in the *Amazon Web Services General Reference*. Updated limits will not be visible even after the limit increase has been applied. For more information on viewing current connection limits in Amazon CloudWatch, see Monitoring Amazon MQ brokers using Amazon CloudWatch.

Topics

Brokers

The following table lists quotas related to Amazon MQ brokers.

Limit	Description
Broker name	• Must be unique in your AWS account. • Must be 1-50 characters long. • Must contain only characters specified in the ASCII Printable Character Set. • Can contain only alphanumeric character s, dashes, periods, underscores, and tildes (- . _ ~).
Number of brokers, per region	200

Limit	Description
Wire-level connections per protocol for smaller broker	**Important** Does not apply to RabbitMQ brokers. 300 for `mq.*.micro` instance type brokers.
Wire-level connections per protocol for larger broker	**Important** Does not apply to RabbitMQ brokers. 2,000 for `mq.*.*large` instance type brokers.
Security groups per broker	5
ActiveMQ destinations (queues, and topics) monitored in CloudWatch	CloudWatch monitors only the first 1000 destinations.
RabbitMQ destinations (queues) monitored in CloudWatch	CloudWatch monitors only the first 500 destinations, ordered by number of consumers .
Tags per broker	50

Configurations

The following table lists quotas related to Amazon MQ configurations.

Limit	Description
Configuration name	• Must be 1-150 characters long. •

Limit	Description
	Must contain only characters specified in the ASCII Printable Character Set. • Can contain only alphanumeric characters, dashes, periods, underscores, and tildes (- . _ ~).
Revisions per configuration	300

Users

The following table lists quotas related to Amazon MQ ActiveMQ broker users.

Limit	Description
Username	• Must be 1-100 characters long. • Must contain only characters specified in the ASCII Printable Character Set. • Can contain only alphanumeric characters, dashes, periods, underscores, and tildes (- . _ ~). • Must not contain commas (,).
Password	• Must be 12-250 characters long. • Must contain only characters specified in the ASCII Printable Character Set. • Must contain at least 4 unique characters. •

Limit	Description
	Must not contain commas (`,`).
Users per broker (simple auth)	250
Groups per user (simple auth)	20

Data Storage

The following table lists quotas related to Amazon MQ data storage.

Limit	Description
Storage capacity per smaller broker	20 GB for `mq.*.micro` instance type brokers. For more information regarding A mazon MQ instance types, see Broker instance types.
Storage capacity per larger broker	200 GB for `mq.m5.*` instance type brokers. For more information regarding Amazon MQ instance types, see Broker instance types.
Job scheduler usage limit per broker backed by Amazon EBS	**Important** Does not apply to RabbitMQ brokers. 50 GB. For more information about job scheduler usage, see `JobSchedulerUsage` in the *Apache ActiveMQ API Documentation*.
Temporary storage capacity per smaller broker.	**Important** Does not apply to RabbitMQ brokers.

Limit	Description
	5 GB for `mq.*.micro` instance type brokers.
Temporary storage capacity per larger broker.	**Important** Does not apply to RabbitMQ brokers. 50 GB for `mq.m5.*` instance type brokers.

API Throttling

The following throttling quotas are aggregated per AWS account, *across all Amazon MQ APIs* to maintain service bandwidth. For more information about Amazon MQ APIs, see the *Amazon MQ REST API Reference*.

Important

These quotas don't apply to Amazon MQ for ActiveMQ or Amazon MQ for RabbitMQ broker messaging APIs. For example, Amazon MQ doesn't throttle the sending or receiving of messages.

API burst limit	API rate limit
100	15

Troubleshooting Amazon MQ

This section describes common issues you might encounter when using Amazon MQ brokers, and the steps you can take to resolve them. For general troubleshooting, see the section called "Troubleshooting: General Amazon MQ". For troubleshooting your specific engine version, see the following sections.

Troubleshooting ActiveMQ on Amazon MQ

Troubleshooting topic	Description
General troubleshooting	Use the information in this section to help you diagnose and resolve common issues you might encounter when working with ActiveMQ on Amazon MQ brokers.
BROKER_ENI_DELETED	ActiveMQ on Amazon MQ will raise a BROKER_ENI_DELETED alarm when you delete a broker's Elastic Network Interface (ENI).
BROKER_OOM	ActiveMQ on Amazon MQ will raise a BROKER_OOM alarm when the broker undergoes a restart loop due to the insuffici ent memory capacity

Troubleshooting RabbitMQ on Amazon MQ

Troubleshooting topic	Description
General troubleshooting	Diagnose common issues you might encounter when working with RabbitMQ brokers.

Troubleshooting topic	Description
RABBITMQ_MEMORY_ALARM	RabbitMQ will raise a high memory alarm when the broker's memory usage, identified by CloudWatch metric `RabbitMQMemUsed` , exceeds the memory limit, identified by `RabbitMQMemLimit` .
RABBITMQ_INVALID_KMS_KEY	RabbitMQ on Amazon MQ will raise an INVALID_KMS_KEY critical action required code when a broker created with a customer managed AWS KMS key(CMK) detects that the AWS Key Management Service (KMS) key is disabled.
RABBITMQ_INVALID_ASSUMEROLE	RabbitMQ on Amazon MQ will raise an INVALID_ASSUMEROLE critical action required code when the IAM role ARN specified in `aws.arns.assume_role_arn` cannot be assumed by Amazon MQ.
RABBITMQ_INVALID_ARN_LDAP	RabbitMQ on Amazon MQ will raise an INVALID_ARN_LDAP critical action required code when the LDAP service account password ARN is invalid or inaccessible.

Troubleshooting topic	Description
RABBITMQ_INVALID_ARN_HTTP	RabbitMQ on Amazon MQ will raise an INVALID_ARN_HTTP critical action required code when one or more ARNs of SSL certificates or key file for HTTP auth_backend are invalid or inaccessible.
RABBITMQ_INVALID_ARN_SSL	RabbitMQ on Amazon MQ will raise an INVALID_ARN_SSL critical action required code when one or more ARNs of CA certificate truststore for EXTERNAL auth_mechanism are invalid or inaccessible.
RABBITMQ_INVALID_ARN	RabbitMQ on Amazon MQ will raise an INVALID_ARN critical action required code when one or more ARNs in the broker configuration are invalid or inaccessible.
RABBITMQ_DISK_ALARM	Disk limit alarm is an indicatio n that the volume of disk used by a RabbitMQ node has decreased due to a high number of messages not consumed while new messages were added.

Troubleshooting: General Amazon MQ

Use the information in this section to help you diagnose common issues you might encounter when working with Amazon MQ brokers, such as issues connecting to your broker, and broker reboots.

Contents

I can't connect to my broker web console or endpoints.

If you're experiencing issues connecting to your broker using the web console or wire-level endpoints, we recommend the following steps.

1. Check whether you're attempting to connect to your broker from behind a firewall. You might need to configure the firewall to allow access to your broker.
2. Check whether you're trying to connect to your broker using a FIPS endpoint. Amazon MQ only supports FIPS endpoints when using API operations, and not for wire-level connections to the broker instance itself.
3. Check if the **Public Accessibility** option for your broker is set to **Yes**. If this is set to **No**, check your subnet's network Access Control List (ACL) rules. If you've created custom network ACLs, you might need to change the network ACL rules to provide access to your broker. For more information on Amazon VPC networking, see Enabling internet access in the *Amazon VPC User Guide*
4. Check your broker's Security Group rules. Make sure that you are allowing connections to the following ports:

   > **Note**
   >
   > The following ports are grouped according to engine types because ActiveMQ on Amazon MQ and RabbitMQ on Amazon MQ use different ports for connections.

   ActiveMQ on Amazon MQ

   - Web console – Port 8162
   - OpenWire – Port 61617
   - AMQP – Port 5671

- STOMP – Port 61614
- MQTT – Port 8883
- WSS – Port 61619

RabbitMQ on Amazon MQ

- Web console and management API – Port 443 and 15671
- AMQP – Port 5671

5. Run the following network connectivity tests for your broker engine type.

> **Note**
>
> For brokers without public accessibility, run the tests from an Amazon EC2 instance within the same Amazon VPC as your Amazon MQ broker and evaluate the responses.

ActiveMQ on Amazon MQ

To test your ActiveMQ on Amazon MQ broker's network connectivity

1. Open a new terminal or command line window.
2. Run the following `nslookup` command to query your broker DNS record. For active/standby deployments, test both the active and standby endpoints. The active/standby endpoints are identified with a suffix, `-1` or `-2` added to the unique broker ID. Replace the endpoint with your information.

```
$ nslookup b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-west-2.amazonaws.com
```

If the query succeeds, you will see an output similar to the following.

```
Non-authoritative answer:
Server:  dns-resolver-corp-sfo-1.sfo.corp.amazon.com
Address:  172.10.123.456

Name:    ec2-12-345-123-45.us-west-2.compute.amazonaws.com
Address:  12.345.123.45
Aliases:  b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-west-2.amazonaws.com
```

The resolved IP address should match the IP addresses provided in the Amazon MQ console. This indicates that the domain name is resolving correctly on the DNS server, and you can move on to the next step.

3. Run the following `telnet` command to test the network path for your broker. Replace the endpoint with your information. Replace *`port`* with port number 8162 for the web console, or other wire-level ports to test additional protocols as needed.

> **Note**
>
> For active/standby deployments, you will receive a `Connect failed` error message if you run `telnet` with the standby endpoint. This is expected, as the standby instance itself is running, but the ActiveMQ process is not running and does not have access to the broker's Amazon EFS storage volume. Run the command for both `-1` and `-2` endpoints to ensure you test both the active and the standby instances.

```
$ telnet b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-
west-2.amazonaws.com port
```

For the active instance, you will see an output similar to the following.

```
Connected to b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-
west-2.amazonaws.com.
Escape character is '^]'.
```

4. Do one of the following.

- If the `telnet` command succeeds, check the `EstablishedConnectionsCount` metric and confirm that the broker has not reached the maximum Wire-level connection limit. You can also confirm if the limit has been reached by reviewing the broker `General` logs. If this metric is greater than zero, then there is at least one client currently connected to the broker. If the metric shows zero connections, then perform the `telnet` path test again and wait at least one minute before disconnecting, as broker metrics are published every minute.
- If the `telnet` command fails, check the status of your broker's elastic network interface, and confirm that the status is `in-use`. Create an Amazon VPC flow log for

each instance's network interface, and review the generated flow logs. Look for the broker's IP addresses when you ran the `telnet` command, and confirm the connection packets are `ACCEPTED`, including a return packet. For more information, and to see a flow log example, see Flow log record examples in the *Amazon VPC Developer Guide*.

5. Run the following `curl` command to check connectivity to the ActiveMQ admin web console.

```
$ curl https://b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-
west-2.amazonaws.com:8162/index.html
```

If the command succeeds, the output should be an HTML document similar to the following.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://
www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
 charset=iso-8859-1" />
        <title>Apache ActiveMQ</title>
        ...
```

RabbitMQ on Amazon MQ

To test your RabbitMQ on Amazon MQ broker's network connectivity

1. Open a new terminal or command line window.
2. Run the following `nslookup` command to query your broker DNS record. Replace the endpoint with your information.

```
$ nslookup b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-west-2.amazonaws.com
```

If the query succeeds, you will see an output similar to the following.

```
Non-authoritative answer:
Server:  dns-resolver-corp-sfo-1.sfo.corp.amazon.com
Address:  172.10.123.456
```

```
Name:    rabbit-broker-1c23e456ca78-b9000123b4ebbab5.elb.us-
west-2.amazonaws.com
Addresses:  52.12.345.678
          52.23.234.56
          41.234.567.890
          54.123.45.678
Aliases:  b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-west-2.amazonaws.com
```

3. Run the following `telnet` command to test the network path for your broker. Replace the endpoint with your information. You can replace *`port`* with port 443 for the web console, and 5671 to test the wire-level AMQP connection.

```
$ telnet b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-
west-2.amazonaws.com port
```

If the command succeeds, you'll see an output similar to the following.

```
Connected to b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-
west-2.amazonaws.com.
Escape character is '^]'.
```

> **Note**
>
> The telnet connection will close automatically after a few seconds.

4. Do one of the following.

- If the `telnet` command succeeds, check the `ConnectionCount` metric and confirm that the broker has not reached the value set in the `max-connections` default policy. You can also confirm if the limit has been reached by reviewing the broker `Connection.log` log group. If this metric is greater than zero, there is at least one client currently connected to the broker. If the metric shows zero connections, then perform the `telnet` path test again. You may need to repeat this process if the connection closes before your broker has published new connection metrics to CloudWatch. Metrics are published every minute.
- For brokers without public accessibility, if the `telnet` command fails, check the status of your broker's elastic network interfaces, and confirm that the status is `in-use`. Create an Amazon VPC flow log for each network interface, and review the generated flow logs. Look for the broker's private IP addresses when you the `telnet` command

was invoked, and confirm the connection packets are ACCEPTED, including a return packet. For more information, and to see a flow log example, see Flow log record examples in the *Amazon VPC Developer Guide*.

> **Note**
>
> This step does not apply to RabbitMQ on Amazon MQ brokers with public accessibility.

5. Run the following `curl` command to check connectivity to the RabbitMQ admin web console.

```
$ curl https://b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-west-2.amazonaws.com:443/index.html
```

If the command succeeds, the output should be an HTML document similar to the following.

```
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="X-UA-Compatible" content="IE=edge" />
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
        <title>RabbitMQ Management</title>
        ...
```

My broker is running, and I can verify connectivity using `telnet`, but my clients are unable to connect and are returning SSL exceptions.

Your broker endpoint certificate may have been updated during the broker maintenance window. Amazon MQ broker certificates are rotated periodically to ensure continued availability and security of brokers.

We recommend using the Amazon root certificate authority (CA) in Amazon Trust Services to authenticate against in your clients' trust store. All Amazon MQ broker certificates are signed with this root CA. By using an Amazon root CA, you will no longer need to download the new Amazon MQ broker certificate every time there is a certificate update on the broker.

I created a broker but broker creation failed.

If your broker is in a `CREATION_FAILED` status, do the following.

- Check your IAM permissions. To create a broker must either use the AWS managed IAM policy `AmazonMQFullAccess` or have the correct set of Amazon EC2 permissions in your custom IAM policy. To learn more about the required Amazon EC2 permissions you need, see IAM permissions required to create an Amazon MQ broker.
- Check if the subnet you are choosing for your broker is in a shared Amazon Virtual Private Cloud (VPC). To create an Amazon MQ broker in a shared Amazon VPC, you must create it in the account that owns the Amazon VPC.

My broker restarted and I'm not sure why.

If your broker has restarted automatically, it may be due to one of the following reasons.

- Your broker may have restarted because of a scheduled weekly maintenance window. Periodically, Amazon MQ performs maintenance to the hardware, operating system, or the engine software of a message broker. The duration of the maintenance varies, but can last up to two hours, depending on the operations that are scheduled for your message broker. Brokers might restart at any point during the two hour maintenance window. For more information on broker maintenance windows, see the section called “Scheduling broker maintenance”.
- Your broker instance type might not be suitable to your application workload. For example, running a production workload on a `mq.t3.micro` might result in the broker running out of resources. High CPU utilization, or high broker memory usage can cause a broker to unexpectedly restart. To see how much CPU and memory is being utilized by your broker, use the following CloudWatch metrics for your engine type.
  - **ActiveMQ on Amazon MQ** – Check `CpuUtilization` for the percentage of allocated Amazon EC2 compute units that the broker currently uses. Check `HeapUsage`for the percentage of the ActiveMQ JVM memory limit that the broker currently uses.
  - **RabbitMQ on Amazon MQ** – Check `SystemCpuUtilization` for the percentage of allocated Amazon EC2 compute units that the broker currently uses. Check `RabbitMQMemUsed` for the volume of RAM used in Bytes, and divide by `RabbitMQMemLimit` for the percentage of memory used by the RabbitMQ node.

  For more information on broker instance types and how to choose the right instance type for your workload, see Broker instance types.

Troubleshooting ActiveMQ on Amazon MQ

Use the information in this section to help you diagnose and resolve common issues you might encounter when working with ActiveMQ on Amazon MQ brokers.

Contents

I can't see general or audit logs for my broker in CloudWatch Logs even though I've activated logging.

If you're unable to view logs for your broker in CloudWatch Logs, do the following.

1. Check if the user who creates or reboots the broker has the `logs:CreateLogGroup` permission. If you don't add the `CreateLogGroup` permission to a user before the user creates or reboots the broker, Amazon MQ will not create the log group.
2. Check if you have configured a resource-based policy to allow Amazon MQ to publish logs to CloudWatch Logs. To allow Amazon MQ to publish logs to your CloudWatch Logs log group, configure a resource-based policy to give Amazon MQ access to the following CloudWatch Logs API actions:
   - `CreateLogStream` – Creates a CloudWatch Logs log stream for the specified log group.
   - `PutLogEvents` – Delivers events to the specified CloudWatch Logs log stream.

For more information on configuring ActiveMQ on Amazon MQ to publish logs to CloudWatch Logs, see Configuring logging.

After broker restart or maintenance window, I can't connect to my broker even though the status is `RUNNING`. Why?

You might be encountering connection issues after a broker restart you initiated, after a scheduled maintenance window is completed, or in a failure event, where the standby instance is activated. In either case, connection issues following a broker restart are most likely caused by unusually large numbers of messages persisted in your broker's Amazon EFS or Amazon EBS storage volume. During a restart, Amazon MQ moves persisted messages from storage to broker memory. To confirm this diagnosis, you can monitor the following metrics on CloudWatch for your Amazon MQ for ActiveMQ broker:

- **`StoragePercentUsage`** — Large percentages at or close to 100 percent can cause the broker to refuse connections.
- **`JournalFilesForFullRecovery`** — Indicates the number of of journal files that will be replayed following an unclean shutdown and restart. An increasing, or consistently higher than one, value indicates unresolved transactions that can cause connection issues after restart.
- **`OpenTransactionCount`** — A number larger than zero following a restart indicates that the broker will attempt to store previously consumed messages, as a result causing connection issues.

To resolve this issue, we recommend resolving your XA transactions with either a `rollback()` or a `commit()`. For more information, and to see a code example of resolving XA transactions using `rollback()`, see recovering XA transactions.

I see some of my clients connecting to the broker, while others are unable to connect.

If your broker is in the `RUNNING` status and some clients are able to connect to the broker successfully, while others are unable to do so, you may have reached the wire-level connections limit for the broker. To verify that you've reached the wire-level connections limit, do the following:

- Check the *general* broker logs for your ActiveMQ on Amazon MQ broker in CloudWatch Logs. If the limit has been reached, you will see `Reached Maximum Connections` in the broker logs. For more information on CloudWatch Logs for ActiveMQ on Amazon MQ brokers, see the section called “Understanding the structure of logging in CloudWatch Logs”.

Once the wire-level connections limit is reached, the broker will actively refuse additional incoming connections. To resolve this issue, we recommend upgrading your broker instance type. For more information on choosing the best instance type for your workload, see Broker instance types.

If you've confirmed that the number of your wire-level connections is less than the broker connection limit, the issue might be related to rebooting clients. Check your broker logs for numerous and frequent entries of `... Inactive for longer than 600000 ms - removing ...`. The log entry is indicative of rebooting clients or connectivity issues. This effect is more evident when clients connect to the broker via a Network Load Balancer (NLB) with clients that frequently disconnect and reconnect to the broker. This is more typically observed in container based clients.

Check your client-side logs for further details. The broker will clean up inactive TCP connections after 600000 ms, and free up the connection socket.

I'm seeing exception `org.apache.jasper.JasperException: An exception occurred processing JSP page` on the ActiveMQ console when performing operations.

If you are using simple authentication and configuring `AuthorizationPlugin` for queue and topic authorization, make sure to use the `AuthorizationEntries` element in your XML configuration file, and allow the `activemq-webconsole` group permission to all queues and topics. This ensures that the ActiveMQ web console can communicate with the ActiveMQ broker.

The following example `AuthorizationEntry` grants read and write permissions for all queues and topics to the `activemq-webconsole` group.

```
<authorizationEntries>
    <authorizationEntry admin="activemq-webconsole,admins,users" topic=">"
 read="activemq-webconsole,admins,users" write="activemq-webconsole,admins,users" />
    <authorizationEntry admin="activemq-webconsole,admins,users" queue=">"
 read="activemq-webconsole,admins,users" write="activemq-webconsole,admins,users" />
</authorizationEntries>
```

Similarly when integrating your broker with LDAP, make sure to grant permission for the `amazonmq-console-admins` group. For more information on LDAP integration, see the section called “How LDAP integration works”.

Troubleshooting: RabbitMQ on Amazon MQ

Use the information in this section to help you diagnose and resolve common issues you might encounter when working with RabbitMQ on Amazon MQ brokers.

Contents

I can't see metrics for my queues or virtual hosts in CloudWatch.

If you're unable to view metrics for your queues or virtual hosts in CloudWatch, check if your queue or virtual host names contain any blank spaces, tabs, or other non-ASCII characters.

Amazon MQ cannot publish metrics for virtual hosts and queues with names containing blank spaces, tabs or other non-ASCII characters.

For more information on dimension names, see Dimension in the *Amazon CloudWatch API Reference*.

How do I enable plugins in RabbitMQ on Amazon MQ?

RabbitMQ on Amazon MQ currently only supports the RabbitMQ management, shovel, federation, consistent-hash exchange plugin, which are enabled by default. For more information on using supported plugins, see the section called “Plugins”.

I'm unable to change Amazon VPC configuration for the broker.

Amazon MQ does not support changing Amazon VPC configuration after your broker is created. Please note that you will need to create a new broker with the new Amazon VPC configuration and update the client connection URL with the new broker connection URL.

Cluster deployments have paused my queue synchronizations.

While addressing RabbitMQ's high memory alarms, you may find that messages on one or multiple queues cannot be consumed. These queues may be in the process of synchronizing messages between nodes, during which the respective queues become unavailable for publishing and consuming. Queue synchronizations might become paused due to the high memory alarm, and even contribute to the memory alarm.

For information about stopping and retrying paused queue syncs, see the section called “Resolving paused queue sync”.

My Amazon MQ for RabbitMQ single-instance broker is in a restart loop.

An Amazon MQ for RabbitMQ single-instance broker that raises a high memory alarm is at risk of becoming unavailable if it restarts and doesn't have enough memory to start up. This can cause RabbitMQ to enter a restart loop and prevent any further interactions with the broker until the issue is resolved. If your broker is in a restart loop, you won't be able to apply the Amazon MQ recommended best practices to resolve the high memory alarm.

To recover your broker, we recommend upgrading to a larger instance type with more memory. Unlike in cluster deployments, you can upgrade a single-instance broker while it's experiencing a high memory alarm because there are no queue synchronizations to perform between nodes during a restart.

I've lost access to all administrator accounts on my broker.

You can recover access using IAM authentication. Enable outbound web identity federation for your AWS account, create an IAM role with permissions to obtain web identity tokens, configure your broker to accept IAM authentication via OAuth 2.0, then use IAM credentials to obtain a JWT token and create a new administrator user. For detailed instructions, see the section called “Using IAM authentication and authorization”.

ActiveMQ on Amazon MQ: Deleted Elastic Network Interface alarm

ActiveMQ on Amazon MQ will raise a BROKER_ENI_DELETED alarm when you delete a broker’s Elastic Network Interface (ENI). When you first create an Amazon MQ broker, Amazon MQ

provisions an elastic network interface in the Virtual Private Cloud (VPC) under your account and, thus, requires a number of EC2 permissions.

You must not modify or delete this network interface. Modifying or deleting the network interface can cause a permanent loss of connection between your VPC and your broker. If you wish to delete the network interface, you must delete the broker first.

ActiveMQ on Amazon MQ: Broker Out Of Memory alarm

ActiveMQ on Amazon MQ will raise a BROKER_OOM alarm when the broker undergoes a restart loop due to the insufficient memory capacity. When a broker is in a restart loop, also called a bounce loop, the broker initiates repeated recovery attempts within a short time window. Brokers that cannot complete start-up due to insufficient memory capacity can enter a restart loop, during which interactions with the broker are limited.

Amazon MQ enables metrics for your broker by default. You can view your broker metrics by accessing the Amazon CloudWatch console, or by using the CloudWatch API. The following metrics are useful when diagnosing the ActiveMQ BROKER_OOM alarm:

Amazon MQ CloudWatch metric	Reason for high memory use	
`TotalMessageCount`	Messages are stored in memory until they are consumed or discarded. A high message count might indicate overutilization of resources and can lead to a high memory alarm.	
`HeapUsage`	The percentage of the ActiveMQ JVM memory limit that the broker currently uses. A higher percentage indicates the broker is using significant resources and may lead to an OOM alarm.	

Amazon MQ CloudWatch metric	Reason for high memory use
`ConnectionCount`	Client connections utilize memory, and too many simultaneous connections can lead to a high memory alarm.
`CpuUtilization`	The percentage of allocated EC2 compute units that the broker currently uses.
`TotalConsumerCount`	For every consumer connected to the broker, a set number of messages are loaded from storage into memory before they are delivered to the consumer. A large number of consumer connections might cause high memory usage and lead to a high memory alarm.

To prevent restart loops and avoid the BROKER_OOM alarm, ensure that messages are consumed quickly. You can do this by choosing the most effective broker instance type, and also cleaning your Dead Letter Queue to discard undeliverable or expired messages. You can learn more about ensuring effective performance at ActiveMQ on Amazon MQ best practices.

Amazon MQ for RabbitMQ: High memory alarm

Amazon MQ for RabbitMQ will raise a high memory alarm when the broker's memory usage, identified by CloudWatch metric `RabbitMQMemUsed`, exceeds the memory limit, identified by `RabbitMQMemLimit`.

A RabbitMQ broker that has raised a high memory alarm will block all clients that are publishing messages. Your broker may enter a restart loop, experience paused queue synchronization, or develop other issues that complicate diagnosis and resolution of the alarm.

To diagnose and resolve high memory alarm, first follow all best practices for RabbitMQ, then complete the following steps.

Important

- `RabbitMQMemLimit` is set by Amazon MQ and is specifically tuned considering the memory available for each host instance type.
- Amazon MQ will not restart a broker experiencing a high memory alarm and will return an exception for `RebootBroker` API operations as long as the broker continues to raise the alarm.

Step 1: Diagnose high memory alarm

There are two ways to diagnose high memory alarms on your Amazon MQ for RabbitMQ broker. We recommend that you check both the RabbitMQ web console and Amazon MQ metrics in CloudWatch.

Diagnose high memory alarm using the RabbitMQ web console

The RabbitMQ web console can generate and display detailed memory usage information for each node. You can find this information by doing the following:

1. Sign in to AWS Management Console and open your broker's RabbitMQ web console.
2. On the RabbitMQ console, on the **Overview** page, choose the name of a node from the **Nodes** list.
3. On the node detail page, choose **Memory details** to expand the section to view the node's memory usage information.

The memory usage information that RabbitMQ provides in the web console can help you determine which resources might be consuming too much memory and contributing to the high memory alarm. For more information on the memory usage details available via the RabbitMQ web console, see Reasoning About Memory Use on the RabbitMQ Server Documentation website.

Diagnose high memory alarm using Amazon MQ metrics

Amazon MQ enables metrics for your broker by default. You can view your broker metrics by accessing the CloudWatch console, or by using the CloudWatch API. The following metrics are useful when diagnosing the RabbitMQ high memory alarm.

Amazon MQ CloudWatch metric	Reason for high memory use
`MessageCount`	Messages are stored in memory until they are consumed or discarded. A high message count might indicate overutilization of resources and can lead to a high memory alarm.
`QueueCount`	Queues are stored in memory, and a high number of queues can lead to a high memory alarm.
`ConnectionCount`	Client connections utilize memory, and too many simultaneous connections can lead to a high memory alarm.
`ChannelCount`	Similar to connections, channels established using each connection are also stored in node memory, and a high number of channels can lead to a high memory alarm.
`ConsumerCount`	For every consumer connected to the broker, a set number of messages are loaded from storage into

Amazon MQ CloudWatch metric	Reason for high memory use	
	memory before they are delivered to the consumer. A large number of consumer connections might cause high memory usage and lead to a high memory alarm.	
`PublishRate`	Publishing messages utilizes the broker' memory. If the rate at which messages are published to the broker is too high and significantly outpaces the rate at which the broker delivers messages to consumers, the broker might raise a high memory alarm.	

Step 2: Address and prevent high memory alarm

> **Note**
>
> It may take up to several hours for the RABBITMQ_MEMORY_ALARM status to clear after you take the required actions.

Follow all best practices for RabbitMQ as a general method of prevention. For each specific contributor that you identify, we recommend the following set of actions to address and prevent RabbitMQ high memory alarms.

Source of high memory use	Amazon MQ recommend ation for addressing	Amazon MQ recommend ation for preventing
Number of messages	Consume messages published to the queues, purge messages from queues, or delete the queues from your broker.	Enable lazy queues, and set or reduce the queue depth limit.
Number of queues	Reduce the number of queues.	Set or reduce the queue count limit.
Number of connections	Reduce the number of connections.	Set or reduce the connection count limit.
Number of channels	Reduce the number of channels.	Set a maximum number of channels per connection on client applications.
Number of consumers	Reduce the number of consumers connected to the broker.	Set a small consumer pre-fetch limit.
Message publishing rate	Reduce the rate at which publishers send messages to the broker.	Turn on publisher confirms.
Client connection attempt rate	Reduce the frequency at which clients attempt to connect to the broker in order to publish or consume messages, or configure the broker.	Use longer-lived connectio ns to reduce the number and frequency of connection attempts.

After your broker's memory alarm is resolved, you can upgrade your host instance type to an instance with additional resources. For information on how to update your broker's instance type, see `UpdateBrokerInput` in the *Amazon MQ REST API Reference*.

Note

You can't downgrade a broker from an `mq.m5.x` instance type to an `mq.t3.micro` instance type. To downgrade, you must delete your broker and create a new one.

RabbitMQ on Amazon MQ: Invalid AWS Key Management Service Key

RabbitMQ on Amazon MQ will raise an INVALID_KMS_KEY critical action required code when a broker created with a customer managed AWS KMS key(CMK) detects that the AWS Key Management Service (KMS) key is disabled. A RabbitMQ broker with a CMK periodically verifies that the KMS key is enabled and the broker has all necessary grants. If RabbitMQ cannot verify that the key is enabled, the broker is quarantined and RabbitMQ will return INVALID_KMS_KEY.

Without an active KMS key, the broker does not have basic permissions for customer managed KMS keys. The broker cannot perform cryptographic operations using your key until you re-enable your key and the broker restarts. A RabbitMQ broker with a disabled KMS key is quarantined to prevent deterioration. After RabbitMQ determines the KMS key is active again, your broker is removed from quarantine. Amazon MQ does not restart a broker with a disabled KMS key and returns an exception for `RebootBroker` API operations as long as the broker continues to have an invalid KMS key.

Diagnosing and addressing INVALID_KMS_KEY

To diagnose and address the INVALID_KMS_KEY action required code, you must use the AWS Command Line Interface (CLI) and the AWS Key Management Service console.

To re-enable your KMS key

1. Call the `DescribeBroker` method to retrieve the `kmsKeyId` for your CMK broker.
2. Sign in to the AWS Key Management Service console.
3. On the **Customer managed keys** page, locate the KMS Key ID of the problematic broker and verify the status is **Enabled**.
4. If your KMS key has been disabled, re-enable the key by choosing **Key Actions**, then choose **Enable**. After your key has been re-enabled, you must wait for RabbitMQ to remove the broker from quarantine.

To verify that the necessary grants are still associated with the broker's KMS key, call the `ListGrantListGrant` method to verify that `mq_rabbit_grant` and `mq_grant` are present. If the KMS grant or key has been deleted, you must delete the broker and create a new one with all necessary grants. For steps on deleting a broker, see Deleting a broker.

To prevent the INVALID_KMS_KEY critical action required code, do not manually delete or disable a KMS key or CMK grant. If you wish to delete the key, delete the broker first.

RabbitMQ on Amazon MQ: Disk limit alarm

Disk limit alarm is an indication that the volume of disk used by a RabbitMQ node has decreased due to a high number of messages not consumed while new messages were added. RabbitMQ will raise a disk limit alarm when the broker's free disk space, identified by Amazon CloudWatch metric `RabbitMQDiskFree`, reaches the disk limit, identified by `RabbitMQDiskFreeLimit`. `RabbitMQDiskFreeLimit` is set by Amazon MQ and has been defined considering the disk space available for each broker instance type.

An RabbitMQ on Amazon MQ broker that has raised a disk limit alarm will become unavailable for new messages being published. If you have a publisher and consumer on the same connection, the consumer will also be unavailable to receive messages. When running RabbitMQ in a cluster, the disk alarm is cluster-wide. If one node goes under the limit, all other nodes will block incoming messages. Due to the lack of disk space, your broker might also experience other issues that complicate diagnosis and resolution of the alarm.

Amazon MQ will not restart a broker experiencing a disk alarm and will return an exception for `RebootBroker` API operations as long as the broker continues to raise the alarm.

> **Note**
>
> You cannot downgrade a broker from an `mq.m5` instance type to an `mq.t3.micro` instance type. If you wish to downgrade, you must delete your broker and create a new one.

Diagnosing and addressing disk limit alarm

Amazon MQ enables metrics for your broker by default. You can view your broker metrics by accessing the Amazon CloudWatch console, or by using the CloudWatch API.`MessageCount` is a useful metric when diagnosing the RabbitMQ disk limit alarm. Messages are stored in memory until

they are consumed or discarded. A high message count indicates overutilization of disk storage and can lead to a disk alarm.

To diagnose the disk limit alarm, use the Amazon MQ Management Console to:

- Create a new connection to consume messages published to the queues.
- Purge messages from queues.
- Delete the queues from your broker.

> **Note**
>
> It may take up to several hours for the RABBITMQ_DISK_ALARM status to clear after you take the required actions.

To prevent the disk limit alarm from reoccurring, you can upgrade your host instance type to an instance with additional resources. For information on how to update your broker's instance type see `UpdateBrokerInput` in the Amazon MQ REST API Reference. We also recommend keeping your publishers and consumers on different connections.

Amazon MQ for RabbitMQ: Instance type change alarm

`RABBITMQ_CLUSTER_DISK_USAGE_TOO_HIGH_FOR_INSTANCE_CHANGE` is an indication that a requested broker instance type change cannot proceed due to high disk usage on the current RabbitMQ node. Amazon MQ for RabbitMQ will raise this alarm when when the current disk usage exceeds what would be available on the requested instance type, as identified by the CloudWatch metric `RabbitMQDiskFree`.

RabbitMQ brokers that enter `RABBITMQ_CLUSTER_DISK_USAGE_TOO_HIGH_FOR_INSTANCE_CHANGE` state will continue to be available for your applications, but the requested instance type change will not proceed. Amazon MQ allows broker restarts in this state, but you cannot change the instance type while the disk usage remains above the threshold for the requested instance type. The broker will return an exception for `ModifyBroker` API operations that attempt to change the instance type while in this state.

Diagnosing and addressing instance type change alarm

Amazon MQ enables metrics for your broker by default. You can view your broker metrics by accessing the CloudWatch console or by using the CloudWatch API. `MessageCount` and `RabbitMQDiskFree` metrics can be used to diagnose `RABBITMQ_CLUSTER_DISK_USAGE_TOO_HIGH_FOR_INSTANCE_CHANGE`.

To resolve the quarantine state and allow your instance type change to proceed, use the Amazon MQ Management Console to:

- Create a new connection to consume messages published to the queues.
- Purge messages from queues.
- Delete the queues from your broker.

> **Note**
>
> It may take up to several hours for the `RABBITMQ_CLUSTER_DISK_USAGE_TOO_HIGH_FOR_INSTANCE_CHANGE` status to clear after you take the required actions.

RabbitMQ on Amazon MQ: Invalid IAM Assume Role

RabbitMQ on Amazon MQ will raise an INVALID_ASSUMEROLE critical action required code when the IAM role ARN specified in `aws.arns.assume_role_arn` is invalid or cannot be assumed by Amazon MQ. This can occur when the role does not exist, is in a different AWS account than the broker, or lacks the necessary trust relationship with mq.amazonaws.com.

A broker in RABBITMQ_INVALID_ASSUMEROLE quarantine cannot retrieve credentials or certificates required for LDAP authentication, rendering LDAP authentication unavailable. If LDAP is the only configured authentication method, users will be unable to connect to the broker. The IAM role is required by Amazon MQ to access AWS resources referenced by ARNs in the broker configuration, such as AWS Secrets Manager secrets or Amazon S3 objects used for LDAP authentication.

Diagnosing and addressing RABBITMQ_INVALID_ASSUMEROLE

To diagnose and address the RABBITMQ_INVALID_ASSUMEROLE action required code, you must use Amazon CloudWatch Logs and the AWS Identity and Access Management console.

To resolve the invalid assume role issue

1. Navigate to Amazon CloudWatch Logs Insights and run the following query against your broker's log group `/aws/amazonmq/broker/<broker-id>/general`:

```
fields @timestamp, @message
| sort @timestamp desc
| filter @message like /error.*aws_arn_config/
| limit 10000
```

2. Look for error messages similar to:

```
[error] <0.254.0> aws_arn_config: {handle_assume_role,{error,
{assume_role_failed,"AWS service is unavailable"}}}
```

3. Check the IAM role configuration and fix any issues such as:
   - Ensure the role exists in the same AWS account as the broker
   - Verify the trust policy allows mq.amazonaws.com to assume the role
   - Confirm the role has appropriate permissions to access the required AWS resources
4. Validate the fix using the ARN access validation API endpoint before updating the broker configuration.
5. Update the broker configuration and reboot the broker.

RabbitMQ on Amazon MQ: Invalid LDAP ARN

RabbitMQ on Amazon MQ will raise an INVALID_ARN_LDAP critical action required code when the ARN configured for the LDAP service account password is invalid or inaccessible. This applies to ARNs specified in `aws.arns.auth_ldap.dn_lookup_bind.password` or

`aws.arns.auth_ldap.other_bind.password`, which must reference AWS Secrets Manager secrets containing plaintext passwords.

A broker in RABBITMQ_INVALID_ARN_LDAP quarantine cannot authenticate with the LDAP service account, making LDAP authentication unavailable. If LDAP is the only configured authentication method, users will be unable to connect to the broker. Invalid ARNs can be caused by malformed ARN syntax, references to non-existent secrets, secrets located in a different AWS region than the broker, or insufficient secretsmanager:GetSecretValue permissions in the IAM role.

Diagnosing and addressing RABBITMQ_INVALID_ARN_LDAP

To diagnose and address the RABBITMQ_INVALID_ARN_LDAP action required code, you must use Amazon CloudWatch Logs and the console.

To resolve the invalid LDAP ARN issue

1. Navigate to Amazon CloudWatch Logs Insights and run the following query against your broker's log group `/aws/amazonmq/broker/<broker-id>/general`:

```
fields @timestamp, @message
| sort @timestamp desc
| filter @message like /error.*aws_arn_config/
| limit 10000
```

2. Look for error messages similar to:

```
[error] <0.254.0> aws_arn_config: {<<"could not resolve
 ARN 'arn:aws:secretsmanager:xxx' for configuration
 'aws.arns.auth_ldap.dn_lookup_bind.password', error: \"AWS service is unavailable
\"">>,{error,"AWS service is unavailable"}}
```

3. Check the Secrets Manager secret and fix any issues such as:

   - Verify the secret exists in the same AWS region as the broker
   - Confirm the ARN syntax is correct
   - Ensure the IAM role has secretsmanager:GetSecretValue permissions

4. Validate the fix using the ARN access validation API endpoint before updating the broker configuration.
5. Update the broker configuration and reboot the broker.

RabbitMQ on Amazon MQ: Invalid HTTP ARN

RabbitMQ on Amazon MQ will raise an INVALID_ARN_HTTP critical action required code when one or more ARNs of SSL certificates or key file for HTTP auth_backend are invalid or inaccessible. This applies to ARNS specified in `aws.arns.auth_http.ssl_options.cacertfile`, `aws.arns.auth_http.ssl_options.certfile` or `aws.arns.auth_http.ssl_options.keyfile`, which must reference Amazon S3 objects and AWS Secrets Manager secrets containing certificates and private key.

A broker in RABBITMQ_INVALID_ARN_HTTP quarantine cannot authenticate via the HTTP server. If HTTP is the only configured authentication method, users will be unable to connect to the broker. Invalid ARNs can be caused by malformed ARN syntax, references to non-existent secrets, secrets located in a different AWS region than the broker, or insufficient s3:GetObject/ secretsmanager:GetSecretValue permissions in the IAM role.

Diagnosing and addressing RABBITMQ_INVALID_ARN_HTTP

To diagnose and address the RABBITMQ_INVALID_ARN_HTTP action required code, you must use Amazon CloudWatch Logs and the console.

To resolve the invalid HTTP ARN issue

1. Navigate to Amazon CloudWatch Logs Insights and run the following query against your broker's log group `/aws/amazonmq/broker/<broker-id>/general`:

```
fields @timestamp, @message
| sort @timestamp desc
| filter @message like /error.*aws_arn_config/
| limit 10000
```

2. Look for error messages similar to:

```
[error] <0.209.0> aws_arn_config: {<<"could not resolve ARN 'arn:aws:s3:::xxxx' for
 configuration 'aws.arns.auth_http.ssl_options.certfile', error: \"AWS service is
 unavailable\"">>,{error,"AWS service is unavailable"}}
```

3. Check the S3 Object/Secrets Manager secret and fix any issues such as:
   - Verify the resource exists in the same AWS region as the broker
   - Confirm the ARN syntax is correct
   - Ensure the IAM role has s3:GetObject and secretsmanager:GetSecretValue permissions
4. Validate the fix using the ARN access validation API endpoint before updating the broker configuration.
5. Update the broker configuration and reboot the broker.

RabbitMQ on Amazon MQ: Invalid SSL ARN

RabbitMQ on Amazon MQ will raise an INVALID_ARN_SSL critical action required code when one or more ARNs of CA certificate truststore for EXTERNAL auth_mechanism are invalid or inaccessible. This applies to ARNS specified in `aws.arns.ssl_options.cacertfile` or `aws.arns.management.ssl.cacertfile`, which must reference Amazon S3 or ACM PCA object containing the certificate.

A broker in RABBITMQ_INVALID_ARN_SSL quarantine cannot authenticate client certificates during mutual TLS handshakes because no valid truststore is configured. If EXTERNAL auth mechanism is the only configured authentication method, users will be unable to connect to the broker. Invalid ARNs can be caused by malformed ARN syntax, references to non-existent S3 objects, S3 objects located in a different AWS region than the broker, or insufficient s3:GetObject/acm-pca:GetCertificateAuthorityCertificate permissions in the IAM role.

Diagnosing and addressing RABBITMQ_INVALID_ARN_SSL

To diagnose and address the RABBITMQ_INVALID_ARN_SSL action required code, you must use Amazon CloudWatch Logs and the console.

To resolve the invalid SSL ARN issue

1. Navigate to Amazon CloudWatch Logs Insights and run the following query against your broker's log group `/aws/amazonmq/broker/<broker-id>/general`:

```
fields @timestamp, @message
| sort @timestamp desc
| filter @message like /error.*aws_arn_config/
| limit 10000
```

2. Look for error messages similar to:

```
[error] <0.209.0> aws_arn_config: {<<"could not resolve ARN 'arn:aws:acm-pca:xxxx'
 for configuration 'aws.arns.ssl_options.cacertfile', error: \"AWS service is
 unavailable\"">>,{error,"AWS service is unavailable"}}
```

3. Check the S3/ACM-PCA Object and fix any issues such as:
   - Verify the secret exists in the same AWS region as the broker
   - Confirm the ARN syntax is correct
   - Ensure the IAM role has s3:GetObject/acm-pca:GetCertificateAuthorityCertificate permissions
4. Validate the fix using the ARN access validation API endpoint before updating the broker configuration.
5. Update the broker configuration and reboot the broker.

RabbitMQ on Amazon MQ: Invalid ARN

RabbitMQ on Amazon MQ will raise an INVALID_ARN critical action required code when one or more ARNs configured in the broker are invalid or inaccessible. This applies to ARNs used for SSL certificates, AWS Secrets Manager secrets, Amazon S3 objects, or other AWS resource references not covered by more specific quarantine codes such as RABBITMQ_INVALID_ARN_LDAP or RABBITMQ_INVALID_ASSUMEROLE.

A broker in RABBITMQ_INVALID_ARN quarantine may experience degraded functionality depending on which ARNs are invalid. Features that depend on the inaccessible resources will be unavailable, and the broker will log errors indicating which ARN failed to resolve. The impact on broker availability depends on whether the invalid ARN is required for critical broker operations.

Diagnosing and addressing RABBITMQ_INVALID_ARN

To diagnose and address the RABBITMQ_INVALID_ARN action required code, you must use Amazon CloudWatch Logs and the appropriate AWS service console for the affected resource.

To resolve the invalid ARN issue

1. Navigate to Amazon CloudWatch Logs Insights and run the following query against your broker's log group `/aws/amazonmq/broker/<broker-id>/general`:

```
fields @timestamp, @message
| sort @timestamp desc
| filter @message like /error.*aws_arn_config/
| limit 10000
```

2. Look for error messages similar to:

```
[error] <0.254.0> aws_arn_config: {<<"could not resolve ARN
 'arn:aws:s3:::bucket-name/certificate.pem' for configuration
 'aws.arns.auth_ldap.ssl_options.cacertfile', error: \"AWS service is unavailable
\"">>,{error,"AWS service is unavailable"}}
```

3. Check the AWS resource and fix any issues such as:

   - Verify the resource exists in the same AWS region as the broker
   - Confirm the ARN syntax is correct
   - Ensure the IAM role has appropriate permissions to access the resource

4. Validate the fix using the ARN access validation API endpoint before updating the broker configuration.
5. Update the broker configuration and reboot the broker.

Related resources

Amazon MQ resources

The following table lists useful resources for working with Amazon MQ.

Resource	Description
Amazon MQ REST API Reference	Descriptions of REST resources, example requests, HTTP methods, schemas, parameters, and the errors that the service returns.
Amazon MQ in the *AWS CLI Command Reference*	Descriptions of the AWS CLI commands that you can use to work with message brokers.
Amazon MQ in the *AWS CloudFormation User Guide*	The `AWS::AmazonMQ::Broker` resource lets you create Amazon MQ brokers, add configuration changes or modify users for the specified broker, return information about the specified broker, and delete the specified broker. The `AWS::AmazonMQ::Configuration` resource lets you create Amazon MQ configurations, add configuration changes or modify users, and return information about the specified configuration.
Regions and Endpoints	Information about Amazon MQ regions and endpoints
Product Page	The primary web page for information about Amazon MQ.
Discussion Forum	A community-based forum for developers to discuss technical questions related to Amazon MQ.

Resource	Description
AWS Premium Support Information	The primary web page for information about AWS Premium Support, a one-on-one, fast-response support channel to help you build and run applications on AWS infrastructure services

Amazon MQ for ActiveMQ resources

The following table lists useful resources for working with Apache ActiveMQ.

Resource	Description
Apache ActiveMQ Getting Started Guide	The official documentation of Apache ActiveMQ.
ActiveMQ in Action	A guide to Apache ActiveMQ that covers the anatomy of JMS messages, connectors, message persistence, authentication, and authorization.
Cross-Language Clients	A list of programming languages and corresponding Apache ActiveMQ libraries. See also ActiveMQ Client and QpidJMS Client.

Amazon MQ for RabbitMQ resources

The following table lists useful resources for working with RabbitMQ.

Resource	Description
The RabbitMQ Getting Started Guide	The official documentation of RabbitMQ.
RabbitMQ Client Libraries and Developer Tools	A guide to the officially supported client libraries and devloper tools for working with

Resource	Description
	RabbitMQ using a variety of programming languages and platforms.
RabbitMQ Best Practices	CloudAMQP's guide on best practices and recommendations for working with RabbitMQ.

Amazon MQ release notes

The following table lists Amazon MQ feature releases and improvements.

Date	Documentation Update
February 19, 2026	Amazon MQ Amazon MQ now supports ActiveMQ 5.19, a new minor engine version release. For more information, see • ActiveMQ 5.19 Release Page • Managing Amazon MQ for ActiveMQ engine versions • Upgrading an Amazon MQ broker engine version • Using Spring XML configuration files
January 22, 2026	Amazon MQ now supports JMS topic exchange plugin for brokers on RabbitMQ 4.2 and above. You can use the official RabbitMQ JMS client to run JMS workloads on Amazon MQ for RabbitMQ broker. It supports JMS 1.1, 2.0 and 3.1. For more information, see • Official JMS 2.0 spec (backward compatible with and extended JMS 1.1) • Official JMS 3.1 spec • Limitation of RabbitMQ JMS client • Connecting your JMS application to Amazon MQ for RabbitMQ broker
January 8, 2026	Amazon MQ now supports SSL certificate authentication for brokers on RabbitMQ 4.2 and above using X.509 client certificates, and mutual TLS (mTLS) configuration. You can configure SSL certificate authentication and mTLS through AWS Management Console, AWS CloudFormation, AWS CLI, or AWS CDK in all AWS Regions where Amazon MQ is available. For more information, see SSL certificate authentication and Configuring mTLS.

Date	Documentation Update
January 6, 2026	Amazon MQ now supports HTTP authentication and authorization for brokers on RabbitMQ 4.2 and above with external HTTP servers. You can configure HTTP authentication through AWS Management Console, AWS CloudFormation, AWS CLI, or AWS CDK in all AWS Regions where Amazon MQ is available. For more information, see HTTP authentication and authorization.
November 20, 2025	Amazon MQ now supports RabbitMQ 4.2, a new major version release which introduces native support for the AMQP 1.0 protocol, a new Raft based metadata store Khepri, local shovels, and message priorities for quorum queues. RabbitMQ 4.2 also includes various bug fixes and performance improvements for throughput and memory management. While this version introduces new features, there are some breaking changes. For more information, see • RabbitMQ 4 • Open-source RabbitMQ release notes • Configuring resource limits • Supported Protocols • Amazon MQ Version upgrades
November 18, 2024	Amazon MQ now supports Graviton3 powered m7g instances for RabbitMQ in a range of sizes from medium to 16xlarge in Africa (Cape Town). For more information, see Amazon MQ for RabbitMQ broker instance types.
November 17, 2025	Amazon MQ now supports LDAP authentication and authorization for RabbitMQ brokers with external LDAP directory services. You can configure LDAP through AWS Management Console, AWS CloudFormation, AWS CLI, or AWS CDK in all AWS Regions where Amazon MQ is available. For more information, see LDAP authentication and authorization for Amazon MQ for RabbitMQ.

Date	Documentation Update
October 22, 2025	Amazon MQ is now available in the Asia Pacific (New Zealand) Region. For information on available regions, see AWS Regions and Endpoints in the *AWS General Reference guide*.
September 3, 2025	Amazon MQ now supports OAuth 2.0 authentication and authorization for RabbitMQ brokers with public identity providers (IdPs). You can configure OAuth 2.0 through AWS Management Console, AWS CloudFormation, AWS CLI, or AWS CDK in all AWS Regions where Amazon MQ is available. For more information, see OAuth 2.0 authentication and authorization for Amazon MQ for RabbitMQ.
July 22, 2025	Amazon MQ now supports Graviton3 powered `m7g` instances for RabbitMQ in a range of sizes from medium to 16xlarge. RabbitMQ clusters running on `m7g` instances deliver up to 50% higher workload capacity and up to 85% throughput improvements over comparable Amazon MQ for RabbitMQ clusters running on `m5` instances. `M7g`instances also have optimized disk volume sizes that vary by instance size. For more information, see Broker instance types. `M7g` instances on Amazon MQ are available today across all generally available Regions except Africa (Cape Town), Canada West (Calgary), and Europe (Milan) Regions.
July 8, 2025	Amazon MQ is now available in the Asia Pacific (Taipei) Region. For information on available regions, see AWS Regions and Endpoints in the *AWS General Reference guide*.
April 22, 2025	You can now delete Amazon MQ broker configurations using the `DeleteConfiguration` API. For more information, see Configurations in the *Amazon MQ API Reference*.

Date	Documentation Update
April 16, 2025	Amazon MQ for RabbitMQ now supports using dual-stack (IPv4 and IPv6) endpoints to connect to public and private brokers. For more informati on, see Connecting to Amazon MQ and Configuring a private Amazon MQ broker.
April 7, 2025	Amazon MQ is now available in Asia Pacific (Thailand) and Mexico (Central) Regions. For information on available regions, see AWS Regions and Endpoints in the *AWS General Reference guide*.
February 13, 2025	Amazon MQ API FIPS endpoints are now available in the Canada (Central) and Canada West (Calgary) Regions. For more information on using FIPS endpoints with the Amazon MQ API, see Connecting to Amazon MQ. For information on available regions, see AWS Regions and Endpoints in the *AWS General Reference guide*.
February 12, 2025	Amazon MQ is announcing the following instance type end of support dates: Broker instance types • ActiveMQ `mq.t2.micro` : May 12, 2025 • ActiveMQ `mq.m4.large` : May 12, 2025 You cannot create brokers on `mq.t2.micro` or `mq.m4.large` after March 17, 2025.
December 10, 2024	Amazon MQ now supports using AWS PrivateLink to connect between your virtual private clouds (VPCs) and the Amazon MQ API without exposing your traffic to the public internet. For more information, see the section called "Connect to Amazon MQ using AWS PrivateLink".

Date	Documentation Update
November 18, 2024	Amazon MQ is now available in the Asia Pacific (Malaysia) Region. For information on available regions, see AWS Regions and Endpoints in the *AWS General Reference guide*.
November 14, 2024	Amazon MQ is announcing the following engine version end of support dates: Managing Amazon MQ for ActiveMQ engine versions • ActiveMQ 5.17: June 16, 2025 Managing Amazon MQ for RabbitMQ engine versions • RabbitMQ 3.11: February 17, 2025 • RabbitMQ 3.12: March 17, 2025 For more information on upgrading to the latest version, see Upgrading an Amazon MQ broker engine version
November 13, 2024	Amazon MQ now supports dual-stack service endpoints which you can connect to using either IPv4 or IPv6. Amazon MQ dual-stack Regional service endpoints can be resolved with both A and AAAA DNS records. For more information, see ???.
July 25, 2024	Amazon MQ now supports ActiveMQ 5.18, a new minor engine version release. For more information, see the following: • ActiveMQ 5.18 Release Page • Managing Amazon MQ for ActiveMQ engine versions • Upgrading an Amazon MQ broker engine version • Using Spring XML configuration files

Date	Documentation Update
July 22, 2024	Amazon MQ now supports quorum queues only on brokers using version 3.13 and above. Quorum queues are a replicated FIFO queue type that use the Raft consensus algorithm to maintain data consistency. Quorum queues provide poison message handling, which can help you manage unprocessed messages. To get started with quorum queues, see Quorum queues for RabbitMQ on Amazon MQ.
July 2, 2024	Amazon MQ for RabbitMQ now supports RabbitMQ 3.13, a minor version release. For all brokers using engine version 3.13 and above, Amazon MQ manages upgrades to the latest supported patch version during the maintenance window. For more information, see Upgrading an Amazon MQ broker engine version. Amazon MQ for RabbitMQ sizing guidelines have been updated to include new limits for queues, consumers per channel, and shovels for brokers using engine version 3.13. For more information about the fixes and features in this release, see the RabbitMQ 3.13 release notes on the RabbitMQ server GitHub repository. For more information about supported Amazon MQ for RabbitMQ versions and broker upgrades, see Managing Amazon MQ for RabbitMQ engine versions.
June 10, 2024	Amazon MQ is now available in the Canada West (Calgary) Region. For information on available regions, see AWS Regions and Endpoints in the *AWS General Reference guide*.

Date	Documentation Update
May 10, 2024	The Amazon MQ version support calendar indicates when a broker engine version reaches end of support. When an engine version reaches end of support, Amazon MQ updates all brokers on the version to the next supported minor version automatically. Amazon MQ provides at least a 90 day notice before an engine version reaches end of support. To view the version support calendar and end of support, see the following: • Managing Amazon MQ for ActiveMQ engine versions • Managing Amazon MQ for RabbitMQ engine versions You can also enable automatic minor version upgrades for your broker to update to the next patch version during a maintenance window. For more information, see Upgrading an Amazon MQ broker engine version
May 9, 2024	Amazon MQ for RabbitMQ now supports RabbitMQ 3.12, a minor version release. All brokers on 3.12.13 and above use Classic Queues version 2 (CQv2), and all queues on 3.12.13 and above behave as lazy queues. We recommend brokers on versions prior to 3.12.13 enable CQv2 and lazy queues, or upgrade to the newest version of Amazon MQ for RabbitMQ. For more information about the fixes and features in this release, see the following: • RabbitMQ 3.12 release notes on the RabbitMQ server GitHub repository. For more information about supported Amazon MQ for RabbitMQ versions and broker upgrades, see Managing Amazon MQ for RabbitMQ engine versions.

Date	Documentation Update
March 4, 2024	Amazon MQ for RabbitMQ now supports RabbitMQ 3.11.28. For more information about the fixes and features in this release, see the following: • RabbitMQ 3.11.28 release notes on the RabbitMQ server GitHub repository • RabbitMQ changelog For more information about supported Amazon MQ for RabbitMQ versions and broker upgrades, see Managing Amazon MQ for RabbitMQ engine versions.
January 19, 2024	Amazon MQ for RabbitMQ does not support the username "guest", and will delete the default guest account when you create a new broker. Amazon MQ will also periodically delete any customer created account called "guest".
December 15, 2023	Amazon MQ is now available in the Israel (Tel Aviv) Region. For information on available regions, see AWS Regions and Endpoints in the *AWS General Reference guide*.
December 11, 2023	Amazon MQ for RabbitMQ now supports RabbitMQ 3.10.25. For more information about the fixes and features in this release, see the following: • RabbitMQ 3.10.25 release notes on the RabbitMQ server GitHub repository • RabbitMQ changelog For more information about supported Amazon MQ for RabbitMQ versions and broker upgrades, see Managing Amazon MQ for RabbitMQ engine versions.

Date	Documentation Update
October 26, 2023	Amazon MQ has released the latest ActiveMQ minor versions 5.15.16, 5.16.7, 5.17.6 with a critical update. We have deprecated the older minor versions of ActiveMQ and will be updating all brokers on any version of 5.15 to 5.15.16, or 5.16 to 5.16.7 and 5.17 to 5.17.6. For more information on updating your ActiveMQ broker, see Managing Amazon MQ for ActiveMQ engine versions.
September 27, 2023	Amazon MQ for RabbitMQ now supports RabbitMQ 3.11.20. For more information about the fixes and features in this release, see the following: • RabbitMQ 3.11.20 release notes on the RabbitMQ server GitHub repository • RabbitMQ changelog For more information about supported Amazon MQ for RabbitMQ versions and broker upgrades, see Managing Amazon MQ for RabbitMQ engine versions.
July 27, 2023	Amazon MQ for RabbitMQ now supports RabbitMQ 3.11.16 For more information about the fixes and features in this release, see the following: • RabbitMQ 3.11.16 release notes on the RabbitMQ server GitHub repository • RabbitMQ changelog For more information about supported Amazon MQ for RabbitMQ versions and broker upgrades, see Managing Amazon MQ for RabbitMQ engine versions.

Date	Documentation Update
July 27, 2023	Amazon MQ for RabbitMQ now supports creating and applying configura tions to your RabbitMQ broker. For more information on adding confgurations to your broker, see RabbitMQ Broker Configurations. For more information about this feature, see: • Operator policies • Changes to the operator policies
June 23, 2023	Amazon MQ now supports ActiveMQ 5.17.3, a new minor engine version release. This release supports the new cross-Region data replication (CRDR) feature from Amazon MQ. For more information, see the following: • To get started with CRDR, see Cross-Region data replication for Amazon MQ for ActiveMQ in the Developer Guide. • ActiveMQ 5.17.3 Release Page • Managing Amazon MQ for ActiveMQ engine versions • Upgrading an Amazon MQ broker engine version • Using Spring XML configuration files
June 21, 2023	Amazon MQ for ActiveMQ now offers a cross-Region data replication (CRDR) feature that allows for asynchronous message replication from the primary broker in a primary AWS Region to the replica broker in a replica Region. If the primary broker in the primary Region fails, you can promote the replica broker in the secondary Region to primary by initiating a switchover or failover. To get started with CRDR, see Cross-Region data replication for Amazon MQ for ActiveMQ in the Developer Guide.

Date	Documentation Update
May 18, 2023	Amazon MQ is now available in the following regions: • Asia Pacific (Melbourne) • Asia Pacific (Hyderabad) • Europe (Spain) • Europe (Zurich) For information on available regions, see AWS Regions and Endpoints in the *AWS General Reference guide*.
April 14, 2023	Amazon MQ for RabbitMQ now supports RabbitMQ version 3.9.27. For more information about the fixes and features in this release, see the following: • RabbitMQ 3.9.27 release notes on the RabbitMQ server GitHub repository • RabbitMQ changelog For more information about supported Amazon MQ for RabbitMQ versions and broker upgrades, see Managing Amazon MQ for RabbitMQ engine versions.
April 14, 2023	Amazon MQ for RabbitMQ now supports RabbitMQ version 3.10.20. For more information about the fixes and features in this release, see the following: • RabbitMQ 3.10.20 release notes on the RabbitMQ server GitHub repository • RabbitMQ changelog For more information about supported Amazon MQ for RabbitMQ versions and broker upgrades, see Managing Amazon MQ for RabbitMQ engine versions.

Date	Documentation Update
March 31, 2023	Amazon MQ for RabbitMQ has disabled RabbitMQ engine version 3.10.17 The Amazon MQ for RabbitMQ team, and the open source maintainers of RabbitMQ, have identified an issue with the RabbitMQ management console on version 3.10.17. Amazon MQ has retracted this version. To mitigate the impacts of this issue, create new brokers with version 3.10.10 while we work to support a new patch version of RabbitMQ. We recommend activating the Version upgrade option to automatically get the latest bug fixes, security updates and performance enhancements. For more information on available Amazon MQ for RabbitMQ versions, see Amazon MQ for RabbitMQ engine versions.
March 1, 2023	Amazon MQ for RabbitMQ now supports RabbitMQ version 3.10.17. For more information about the fixes and features in this release, see the following: • RabbitMQ 3.10.17 release notes on the RabbitMQ server GitHub repository • RabbitMQ changelog For more information about supported Amazon MQ for RabbitMQ versions and broker upgrades, see Managing Amazon MQ for RabbitMQ engine versions.

Date	Documentation Update
February 21, 2023	Amazon MQ for RabbitMQ now integrates with AWS Key Management Service (KMS) to offer server-side encryption. You can now select your own customer managed CMK, or use an AWS managed KMS key in your AWS KMS account. For more information, see Encryption at rest. Amazon MQ supports using AWS KMS keys in the following ways. • **Amazon MQ owned KMS key (default)** — The key is owned and managed by Amazon MQ and is not in your account. • **AWS managed KMS key** — The AWS managed KMS key (`aws/mq`) is a KMS key in your account that is created, managed, and used on your behalf by Amazon MQ. • **Select existing customer managed KMS key** — Customer managed KMS keys are created and managed by you in AWS Key Management Service (KMS).
January 13, 2023	Amazon MQ for RabbitMQ now supports RabbitMQ version 3.8.34. For more information about the fixes and features in this release, see the following: • RabbitMQ 3.8.34 release notes on the RabbitMQ server GitHub repository • RabbitMQ changelog For more information about supported Amazon MQ for RabbitMQ versions and broker upgrades, see Managing Amazon MQ for RabbitMQ engine versions.

Date	Documentation Update
December 15, 2022	Amazon MQ for RabbitMQ now supports RabbitMQ version 3.9.24. For more information about the fixes and features in this release, see the following: • RabbitMQ 3.9.24 release notes on the RabbitMQ server GitHub repository • RabbitMQ changelog For more information about supported Amazon MQ for RabbitMQ versions and broker upgrades, see Managing Amazon MQ for RabbitMQ engine versions.
December 13, 2022	Amazon MQ is now available in the Middle East (UAE) Region. For informati on on available regions, see AWS Regions and Endpoints in the *AWS General Reference guide*.
November 14, 2022	Amazon MQ for RabbitMQ now supports 3.10, a major engine version release. You can now enable classic Queues version 2 (CQv2) on your RabbitMQ queues. Direct updates from 3.8 to 3.10 are not supported. For more information, see the following: • RabbitMQ 3.10.10 release notes • RabbitMQ changelog For more information about supported Amazon MQ for RabbitMQ versions and broker upgrades, see Managing Amazon MQ for RabbitMQ engine versions.
November 9, 2022	Amazon MQ now supports ActiveMQ 5.17.2, a new minor engine version release. For more information, see the following: • ActiveMQ 5.17.2 Release Page • Managing Amazon MQ for ActiveMQ engine versions • Upgrading an Amazon MQ broker engine version • Using Spring XML configuration files

Date	Documentation Update
August 17, 2022	Amazon MQ now supports ActiveMQ 5.17.1, a new major engine version release. For more information, see the following: • ActiveMQ 5.17.1 Release Page • Managing Amazon MQ for ActiveMQ engine versions • Upgrading an Amazon MQ broker engine version • Using Spring XML configuration files
July 14, 2022	Amazon MQ now supports ActiveMQ 5.16.5, a minor engine version release. For more information, see the following: • ActiveMQ 5.16.5 Release Page • Managing Amazon MQ for ActiveMQ engine versions • Using Spring XML configuration files • Upgrading an Amazon MQ broker engine version
May 4, 2022	Amazon MQ adds inclusive language for `networkConnector` element in broker configuration. • Creating and configuring an Amazon MQ network of brokers
April 25, 2022	Amazon MQ This release adds the `CRITICAL_ACTION_REQUIRED` broker state and the `ActionRequired` API property. `CRITICAL_ACTION_REQUIRED` informs you when your broker is degraded. `ActionRequired` provides you with a code which you can use to find instructions in the Developer Guide on how to resolve the issue. • *Troubleshooting* • `ActionRequired` documentation in the *Amazon MQ API Reference*.

Date	Documentation Update
April 20, 2022	Amazon MQ now supports ActiveMQ 5.16.4, a minor engine version release. For more information, see the following: • ActiveMQ 5.16.4 Release Page • Managing Amazon MQ for ActiveMQ engine versions • Using Spring XML configuration files • Upgrading an Amazon MQ broker engine version
March 1, 2022	Amazon MQ is now available in the Asia Pacific (Jakarta) Region. For information on available regions, see AWS Regions and Endpoints in the *AWS General Reference guide*.
February 25, 2022	Amazon MQ for RabbitMQ now supports RabbitMQ version 3.8.27. For more information about the fixes and features in this release, see the following: • RabbitMQ 3.8.27 release notes on the RabbitMQ server GitHub repository • RabbitMQ changelog For more information about supported Amazon MQ for RabbitMQ versions and broker upgrades, see Managing Amazon MQ for RabbitMQ engine versions.
February 16, 2022	Amazon MQ is now available in the Africa (Cape Town) Region. For informati on on available regions, see AWS Regions and Endpoints in the *AWS General Reference guide*.

Date	Documentation Update
February 14, 2022	Amazon MQ for RabbitMQ now supports RabbitMQ version 3.9.13. Automatic minor version upgrades cannot be used to upgrade from Rabbit 3.8 to 3.9. To do so, manually upgrade your broker. For more information on new features introduced in RabbitMQ 3.9, see the release notes page for version 3.9.0 on the GitHub website. **Note** Currently, Amazon MQ does not support streams, or using structure d logging in JSON, introduced in RabbitMQ 3.9. For more information about the fixes and features in this release, see the following: • RabbitMQ 3.9.13 release notes on the RabbitMQ server GitHub repository • RabbitMQ changelog For more information about supported Amazon MQ for RabbitMQ versions and broker upgrades, see Managing Amazon MQ for RabbitMQ engine versions.
February 07, 2022	Amazon MQ for RabbitMQ introduces new broker metrics, allowing you to monitor average resource utilization across all three nodes in a cluster deployment. For more information, see the following: • the section called “Metrics for RabbitMQ”

Date	Documentation Update
January 18, 2022	Amazon MQ for RabbitMQ now supports RabbitMQ version 3.8.26. For more information about the fixes and features in this release, see the following: • RabbitMQ 3.8.26 release notes on the RabbitMQ server GitHub repository • RabbitMQ changelog For more information about supported Amazon MQ for RabbitMQ versions and broker upgrades, see Managing Amazon MQ for RabbitMQ engine versions.
January 13, 2022	Amazon MQ introduces the `RABBITMQ_MEMORY_ALARM` status code to inform you when your broker has raised a high memory alarm and is in an unhealthy state. Amazon MQ provides detailed information and recommend ations to help you diagnose, resolve and prevent high memory alarms. For more information, see the following. • the section called “ `RABBITMQ_MEMORY_ALARM` ”
January 6, 2022	When you configure CloudWatch Logs for Amazon MQ for ActiveMQ brokers, Amazon MQ supports using the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in IAM resource-based policies to prevent the confused deputy problem. For more information, see the following. • the section called “Cross-service confused deputy prevention”
December 20, 2021	Amazon MQ for ActiveMQ introduces a set of new metrics, allowing you to monitor the maximum number of connections you can make to your broker using different supported transport protocols, as well as an additional new metric that allows you to monitor the number of nodes connected to your broker in a network of brokers. For more information, see the following. • the section called “Metrics for ActiveMQ”

Date	Documentation Update
November 16, 2021	Amazon MQ for RabbitMQ now supports RabbitMQ version 3.8.23. For more information about the fixes and features in this release, see the following: • RabbitMQ 3.8.23 release notes on the RabbitMQ server GitHub repository • RabbitMQ changelog For more information about supported Amazon MQ for RabbitMQ versions and broker upgrades, see Managing Amazon MQ for RabbitMQ engine versions.
October 12, 2021	Amazon MQ now supports ActiveMQ 5.16.3, a minor engine version release. For more information, see the following: • ActiveMQ 5.16.3 Release Page • Managing Amazon MQ for ActiveMQ engine versions • Upgrading an Amazon MQ broker engine version • Using Spring XML configuration files
September 8, 2021	Amazon MQ for RabbitMQ now supports RabbitMQ version 3.8.22. This release includes a fix for an issue with queues using per-message TTL (time to live), identified in the previously supported version, RabbitMQ 3.8.17. We recommend upgrading your existing brokers to version 3.8.22. For more information about the fixes and features in this release, see the following: • RabbitMQ 3.8.22 release notes on the RabbitMQ server GitHub repository • RabbitMQ changelog For more information about supported Amazon MQ for RabbitMQ versions and broker upgrades, see Managing Amazon MQ for RabbitMQ engine versions

Date	Documentation Update
August 25, 2021	Amazon MQ for RabbitMQ has temporarily disabled RabbitMQ engine version 3.8.17 due to an issue identified with queues using per-message time-to-live (TTL). We recommend using version 3.8.11.
July 29, 2021	Amazon MQ for RabbitMQ now supports RabbitMQ version 3.8.17. For more information about the fixes and features contained in this update, see the following: • RabbitMQ 3.8.17 release notes on the RabbitMQ server GitHub repository • RabbitMQ changelog • Managing Amazon MQ for RabbitMQ engine versions
July 16, 2021	You can now adjust the maintenance window of an Amazon MQ broker using the AWS Management Console, AWS CLI, or the Amazon MQ API. To learn more about broker maintenance windows, see the following. • Scheduling the maintenance window for an Amazon MQ broker
July 6, 2021	Amazon MQ for RabbitMQ introduces support for the Consistent Hash exchange type. Consistent Hash exchanges route messages to queues based on a hash value calculated from the routing key of a message. For more information, see the following: • Consistent Hash exchange plugin • RabbitMQ Consistent Hash Exchange Type on the RabbitMQ GitHub repository
June 7, 2021	Amazon MQ now supports ActiveMQ 5.16.2, a new major engine version release. For more information, see the following: • ActiveMQ 5.16.2 Release Page • Managing Amazon MQ for ActiveMQ engine versions • Upgrading an Amazon MQ broker engine version • Using Spring XML configuration files

Date	Documentation Update
May 26, 2021	Amazon MQ for RabbitMQ is now available in the China (Beijing) and China (Ningxia) Regions. For information on available regions, see AWS Regions and Endpoints.
May 18, 2021	Amazon MQ for RabbitMQ implements broker defaults. When you first create a broker, Amazon MQ creates a set of broker policies and vhost limits based on the instance type and deployment mode you choose, in order to optimize the broker's performance. For more informati on, see the following: https://docs.aws.amazon.com//amazon-mq/latest/developer-guide/rabbitmq-defaults.html
May 5, 2021	Amazon MQ now supports ActiveMQ 5.15.15. For more information, see the following: • ActiveMQ 5.15.15 Release Page • Managing Amazon MQ for ActiveMQ engine versions • Using Spring XML configuration files
May 5, 2021	Amazon MQ started tracking changes to AWS managed policies. For more information, see the following: • the section called “AWS managed policies”
April 14, 2021	Amazon MQ is now available in the China (Beijing) and China (Ningxia) Regions. For information on available regions, see AWS Regions and Endpoints.
April 7, 2021	Amazon MQ now supports RabbitMQ 3.8.11. For more information about the fixes and features contained in this update, see the following: • RabbitMQ 3.8.11 release notes on the RabbitMQ server GitHub repository • RabbitMQ changelog • Managing Amazon MQ for RabbitMQ engine versions

Date	Documentation Update
April 1, 2021	Amazon MQ is now available in the Asia Pacific (Osaka) Region. For informati on about available regions, see Amazon MQ regions and endpoints.
December 21, 2020	Amazon MQ now supports ActiveMQ 5.15.14. For more information, see the following: • ActiveMQ 5.15.14 Release Notes • Managing Amazon MQ for ActiveMQ engine versions • Using Spring XML configuration files • **Important** Due to a known Apache ActiveMQ issue in this release, the new **Pause Queue** button in the ActiveMQ web console cannot be used with Amazon MQ for ActiveMQ brokers. For more information about this issue, see AMQ-8104.

Date	Documentation Update
November 4, 2020	Amazon MQ now supports RabbitMQ, a popular open source message broker. This enables you to migrate your existing RabbitMQ message brokers to AWS without having to rewrite code. Amazon MQ for RabbitMQ manages both individual and clustered message brokers and handles tasks like provisioning the infrastructure, setting up the broker, and updating the software. • Amazon MQ supports RabbitMQ 3.8.6. For more information about supported engine versions, see the section called "Version management". • The AWS Free Tier includes up to 750 hours of a single-instance `mq.t3.micro` broker and up to 20GB of storage per month for one year. For more information about supported instance types, see Broker instance types. • With Amazon MQ for RabbitMQ, you can access your brokers using AMQP 0-9-1, and with any language supported by the RabbitMQ client libraries . For more information about supported protocols and cipher suites, see the section called "Amazon MQ for RabbitMQ protocols". • Amazon MQ for RabbitMQ is available in all regions that Amazon MQ is currently available. To learn more about all of the available regions, see the AWS Region Table. To get started with using Amazon MQ, create a broker, and connect a JVM-based application to your RabbitMQ broker, see *Getting started: Creating and connecting to a RabbitMQ broker*.
October 22, 2020	Amazon MQ supports ActiveMQ 5.15.13. For more information, see the following: • ActiveMQ 5.15.13 Release Notes • Managing Amazon MQ for ActiveMQ engine versions • Using Spring XML configuration files

Date	Documentation Update
September 30, 2020	Amazon MQ is now available in the Europe (Milan) Region. For information about available regions, see Amazon MQ regions and endpoints.
July 27, 2020	You can authenticate Amazon MQ users using the credentials stored in your Active Directory or other LDAP server. You can also add, delete, and modify Amazon MQ users and assign permissions to topics and queues. For more information, see Integrate LDAP with ActiveMQ.
July 17, 2020	Amazon MQ now supports the `mq.t3.micro` instance type. For more information, see Broker instance types.
June 30, 2020	Amazon MQ supports ActiveMQ 5.15.12. For more information, see the following: • ActiveMQ 5.15.12 Release Notes • Managing Amazon MQ for ActiveMQ engine versions • Using Spring XML configuration files
April 30, 2020	Amazon MQ supports a new child collection element, `systemUsage`, on the `broker` element. For more information, see systemUsage. Amazon MQ also supports three new attributes on the `kahaDB` child element. • `journalDiskSyncInterval` - Interval (ms) for when to perform a disk sync if `journalDiskSyncStrategy=periodic`. • `journalDiskSyncStrategy` - configures the disk sync policy. • `preallocationStrategy` - configures how the broker will try to preallocate the journal files when a new journal file is needed. For more information, see Attributes.

Date	Documentation Update
March 3, 2020	Amazon MQ supports two new CloudWatch metrics • `TempPercentUsage` - The percentage of available temporary storage used by non-persistent messages. • `JobSchedulerStorePercentUsage` - The percentage of disk space used by the job scheduler store. For more information, see Monitoring and logging Amazon MQ brokers.
February 4, 2020	Amazon MQ is available in the Asia Pacific (Hong Kong) and Middle East (Bahrain) regions. For information on available regions, see AWS Regions and Endpoints.
January 22, 2020	Amazon MQ supports ActiveMQ 5.15.10. For more information, see the following: • ActiveMQ 5.15.10 Release Notes • Managing Amazon MQ for ActiveMQ engine versions • Using Spring XML configuration files
December 19, 2019	Amazon MQ is available in the Europe (Stockholm) and South America (São Paulo) regions. For information on available regions, see AWS Regions and Endpoints.

Date	Documentation Update
December 16, 2019	Amazon MQ supports creating throughput-optimized brokers by using Amazon Elastic Block Store (EBS)—instead of the default Amazon Elastic File System (Amazon EFS)—for broker storage. To take advantage of high durability and replication across multiple Availability Zones, use Amazon EFS. To take advantage of low latency and high throughput, use Amazon EBS. **Important** • You can use Amazon EBS only with the `mq.m5` broker instance type family. • Although you can change the *broker instance type*, you can't change the *broker storage type* after you create the broker. • Amazon EBS replicates data within a single Availability Zone and doesn't support the ActiveMQ active/standby deployment mode. For more information, see the following: • Storage • Choose the correct broker storage type for the best throughput • The `storageType` property of the `broker-instance-options` resource in the *Amazon MQ REST API Reference* • The `BurstBalance`, `VolumeReadOps`, and `VolumeWriteOps` metrics in the Monitoring and logging Amazon MQ brokers section.
October 18, 2019	Two Amazon CloudWatch metrics are available: `TotalEnqueueCount` and `TotalDequeueCount`. For more information, seeMonitoring and logging Amazon MQ brokers

Date	Documentation Update
October 11, 2019	Amazon MQ now supports Federal Information Processing Standard 140-2 (FIPS) compliant endpoints in U.S. commercial regions. For more information see the following: • Federal Information Processing Standard (FIPS) 140-2 • Amazon MQ Regions and Endpoints
September 30, 2019	Amazon MQ now includes the ability to scale your brokers by changing the host instance type. For more information, see the `hostInstanceType` property of `UpdateBrokerInput`, and the `pendingHostInstanceType` property of `DescribeBrokerOutput`.
August 30, 2019	You can now update the security groups associated with a broker, both in the console and with `UpdateBrokerInput`.
July 22, 2019	Amazon MQ integrates with AWS Key Management Service (KMS) to offer server-side encryption. You can now select your own customer managed CMK, or use an AWS managed KMS key in your AWS KMS account. For more information, see Encryption at rest. Amazon MQ supports using AWS KMS keys in the following ways. • **AWS owned KMS key** — The key is owned Amazon MQ and is not in your account. • **AWS managed KMS key** — The AWS managed KMS key (aws/mq) is a KMS key in your account that is created, managed, and used on your behalf by Amazon MQ. • **Select existing customer managed CMK** — Customer managed CMKs are created and managed by you in AWS Key Management Service (KMS).
June 19, 2019	Amazon MQ is available in the Europe (Paris) and Asia Pacific (Mumbai) regions. For information on available regions, see AWS Regions and Endpoints.

Date	Documentation Update
June 12, 2019	Amazon MQ is available in the Canada (Central) region. For information on available regions, see AWS Regions and Endpoints.
June 3, 2019	Two new Amazon CloudWatch metrics are available: `EstablishedConnectionsCount` and `InactiveDurableSubscribers`. For more information, see the following: • Monitoring and logging Amazon MQ brokers • Monitoring and logging Amazon MQ brokers
May 10, 2019	Data storage for new `mq.t2.micro` instance types is limited to 20 GB. For more information, see the following: • the section called “Data Storage” • Broker instance types
April 29, 2019	You can now use tag-based policies and resource-level permissions. For more information, see the following: • How Amazon MQ works with IAM • Resource-level permissions for Amazon MQ API actions
April 16, 2019	You can now retrieve information about broker engine and broker instance options using the REST API. For more information, see the following: • Broker instance options • Broker engine types
April 8, 2019	Amazon MQ supports ActiveMQ 5.15.9. For more information, see the following: • ActiveMQ 5.15.9 Release Notes • Managing Amazon MQ for ActiveMQ engine versions • Using Spring XML configuration files

Date	Documentation Update
March 4, 2019	Improved the documentation for configuring dynamic failover and the rebalancing of clients for a network of brokers. Enable dynamic failover by configuring `transportConnectors` along with `networkConnectors` configuration options. For more information, see the following: • Dynamic Failover With Transport Connectors • Amazon MQ network of brokers • Amazon MQ Broker Configuration Parameters
February 27, 2019	Amazon MQ is available in the Europe (London) Region in addition to the following regions: • Asia Pacific (Singapore) • US East (Ohio) • US East (N. Virginia) • US West (N. California) • US West (Oregon) • Asia Pacific (Tokyo) • Asia Pacific (Seoul) • Asia Pacific (Sydney) • Europe (Frankfurt) • Europe (Ireland)
January 24, 2019	The default configuration now includes a policy to purge inactive destinations.
January 17, 2019	Amazon MQ `mq.t2.micro` instance types now support only 100 connections per wire-level protocol. For more information, see, Quotas in Amazon MQ.

Date	Documentation Update
December 19, 2018	You can configure a series of Amazon MQ brokers in a network of brokers. For more information, see the following sections: • Amazon MQ network of brokers • Creating and Configuring a Network of Brokers • Configure Your Network of Brokers Correctly • `networkConnector` • `networkConnectionStartAsync`

December 11, 2018	Amazon MQ supports ActiveMQ 5.15.8, 5.15.6, and 5.15.0.
	• Resolved bugs and improvements in ActiveMQ:
	  • ActiveMQ 5.15.8 Release Notes
	  • ActiveMQ 5.15.7 Release Notes

December 5, 2018	AWS supports resource tagging to help track your cost allocation. You can tag resources when creating them, or by viewing the details of that resource. For more information, see Tagging resources.
November 19, 2018	AWS has expanded its SOC compliance program to include Amazon MQ as an SOC compliant service.
October 15, 2018	• The maximum number of groups per user is 20. For more information, see Users. • The maximum number of connections per broker, per wire-level protocol is 1,000. For more information, see Brokers.
October 2, 2018	AWS has expanded its HIPAA compliance program to include Amazon MQ as a HIPAA Eligible Service.

Date	Documentation Update

September 27, 2018	Amazon MQ supports ActiveMQ 5.15.6, in addition to 5.15.0. For more information, see the following:
	• Getting started: Creating and connecting to an ActiveMQ broker
	• Resolved bugs and improvements in the ActiveMQ documentation:
	  • ActiveMQ 5.15.6 Release Notes
	  • ActiveMQ 5.15.5 Release Notes
	  • ActiveMQ 5.15.4 Release Notes
	  • ActiveMQ 5.15.3 Release Notes
	  • ActiveMQ 5.15.2 Release Notes
	  • ActiveMQ 5.15.1 Release Notes
	• ActiveMQ Client 5.15.6

August 31, 2018	• The following metrics are available:
	  • `CurrentConnectionsCount`
	  • `TotalConsumerCount`
	  • `TotalProducerCount`
	  For more information, see the Monitoring and logging Amazon MQ brokers section.
	• The IP address of the broker is displayed on the **Details** page.
	  Note
	  For brokers with public accessibility disabled, the internal IP address is displayed.

Date	Documentation Update
August 30, 2018	Amazon MQ is available in the Asia Pacific (Singapore) Region in addition to the following regions: • US East (Ohio) • US East (N. Virginia) • US West (N. California) • US West (Oregon) • Asia Pacific (Tokyo) • Asia Pacific (Seoul) • Asia Pacific (Sydney) • Europe (Frankfurt) • Europe (Ireland)
July 30, 2018	You can configure Amazon MQ to publish general and audit logs to Amazon CloudWatch Logs. For more information, see Monitoring and logging Amazon MQ brokers.
July 25, 2018	Amazon MQ is available in the Asia Pacific (Tokyo) and Asia Pacific (Seoul) Regions in addition to the following regions: • US East (Ohio) • US East (N. Virginia) • US West (N. California) • US West (Oregon) • Asia Pacific (Sydney) • Europe (Frankfurt) • Europe (Ireland)
July 19, 2018	You can use AWS CloudTrail to log Amazon MQ API calls. For more informati on, see Logging Amazon MQ API calls using CloudTrail.

Date	Documentation Update
June 29, 2018	In addition to `mq.t2.micro` and `mq.m4.large`, the following broker instance types are available for regular development, testing, and production workloads that require high throughput: • `mq.m5.large` • `mq.m5.xlarge` • `mq.m5.2xlarge` • `mq.m5.4xlarge` For more information, see Broker instance types.
June 27, 2018	Amazon MQ is available in the US West (N. California) Region in addition to the following regions: • US East (Ohio) • US East (N. Virginia) • US West (Oregon) • Asia Pacific (Sydney) • Europe (Frankfurt) • Europe (Ireland)

Date	Documentation Update

June 14, 2018	• You can use the `AWS::Amazon MQ::Broker` AWS CloudFormation resource to perform the following actions:
	  • Create a broker.
	  • Add configuration changes or modify users for the specified broker.
	  • Return information about the specified broker.
	  • Delete the specified broker.
	
	  Note
	  When you change any property of the Amazon MQ Broker ConfigurationId or Amazon MQ Broker User property type, the broker is rebooted immediately.
	• You can use the `AWS::Amazon MQ::Configuration` AWS CloudFormation resource to perform the following actions:
	  • Create a configuration.
	  • Update the specified configuration.
	  • Return information about the specified configuration.
	
	  Note
	  You can use CloudFormation to modify—but not delete—an Amazon MQ configuration.

June 7, 2018	The Amazon MQ console supports German, Brazilian Portuguese, Spanish, Italian, and Traditional Chinese.
May 17, 2018	The limit of number of users per broker is 250. For more information, see Users.
March 13, 2018	Creating a broker takes about 15 minutes. For more information, see Finish creating the broker.

Date	Documentation Update
March 1, 2018	• You can configure the concurrent store and dispatch for Apache KahaDB using the `concurrentStoreAndDispatchQueues` attribute. • The `>CpuCreditBalance` CloudWatch metric is available for `mq.t2.micro` broker instance type.
January 10, 2018	The following changes affect the Amazon MQ console: • In the broker list, the **Creation** column is hidden by default. To customize the page size and columns, choose  . • On the ***MyBroker*** page, in the **Connections** section, choosing the name of your security group or  opens the EC2 console (instead of the VPC console). The EC2 console allows more intuitive configuration of inbound and outbound rules. For more information, see the updated Connecting a Java application to your broker section.
January 9, 2018	• The permission for REST operation ID `UpdateBroker` is listed correctly as `mq:UpdateBroker` on the IAM console. • The erroneous `mq:DescribeEngine` permission is removed from the IAM console.

Date	Documentation Update

November 28, 2017	This is the initial release of Amazon MQ and the *Amazon MQ Developer Guide*.
	• Amazon MQ is avaialble in the following regions:
	  • US East (Ohio)
	  • US East (N. Virginia)
	  • US West (Oregon)
	  • Asia Pacific (Sydney)
	  • Europe (Frankfurt)
	  • Europe (Ireland)
	  Using the `mq.t2.micro` instance type is subject to *CPU credits and baseline performance*—with the ability to *burst* above the baseline level (for more information, see the `CpuCreditBalance` metric). If your application requires *fixed performance*, consider using an `mq.m5.large` instance type.
	• You can create `mq.m4.large` and `mq.t2.micro` brokers.
	  Using the `mq.t2.micro` instance type is subject to *CPU credits and baseline performance*—with the ability to *burst* above the baseline level (for more information, see the `CpuCreditBalance` metric). If your application requires *fixed performance*, consider using an `mq.m5.large` instance type.
	• You can use the ActiveMQ 5.15.0 broker engine.
	• You can also create and manage brokers programmatically using Amazon MQ REST API and AWS SDKs.
	• You can access your brokers by using any programming language that ActiveMQ supports and by enabling TLS explicitly for the following protocols:
	  • AMQP
	  • MQTT
	  • MQTT over WebSocket
	  • OpenWire

Date	Documentation Update
	• STOMP • STOMP over WebSocket • You can connect to ActiveMQ brokers using various ActiveMQ clients. We recommend using the ActiveMQ Client. For more information, see Connecting a Java application to your broker. • Your broker can send and receive messages of any size.