Amazon MQ

Developer Guide

Amazon MQ: Developer Guide

Table of Contents

What Is Amazon MQ?

Amazon MQ is a managed message broker service for Apache ActiveMQ that makes it easy to migrate to a message broker in the cloud. A *message broker* allows software applications and components to communicate using various programming languages, operating systems, and formal messaging protocols.

Amazon MQ works with your existing applications and services without the need to manage, operate, or maintain your own messaging system.

Topics

What Are the Main Benefits of Amazon MQ?

- **Security** – You control who can create and modify brokers (p. 85) and who can send messages to and receive messages from (p. 87) an ActiveMQ broker. Amazon MQ encrypts messages at rest and in transit using encryption keys that it manages and stores securely.
- **Durability** – To ensure the safety of your messages, Amazon MQ stores them on redundant shared storage (p. 38).
- **Availability** – You can create a single-instance broker (p. 38) (comprised of one broker in one Availability Zone), or an active/standby broker for high availability (p. 39) (comprised of two brokers in two different Availability Zones). For either broker type, Amazon MQ automatically provisions infrastructure for high durability.
- **Enterprise readiness** – Amazon MQ supports industry-standard APIs and protocols so you can migrate from your existing message broker (p. 63) without rewriting application code (p. 55).
- **Operation offloading** – You can configure many aspects of your ActiveMQ broker (p. 41), such as predefined destinations, destination policies, authorization policies, and plugins. Amazon MQ controls some of these configuration elements, such as network transports and storage, simplifying the maintenance and administration of your messaging system in the cloud.

How Is Amazon MQ Different from Amazon SQS or Amazon SNS?

Amazon MQ is a managed message broker service that provides compatibility with many popular message brokers. We recommend Amazon MQ for migrating applications from existing message brokers that rely on compatibility with APIs such as JMS or protocols such as AMQP, MQTT, OpenWire, and STOMP.

Amazon SQS and Amazon SNS are queue and topic services that are highly scalable, simple to use, and don't require you to set up message brokers. We recommend these services for new applications that can benefit from nearly unlimited scalability and simple APIs.

How Can I Get Started with Amazon MQ?

- To create your first broker with Amazon MQ, see Getting Started with Amazon MQ (p. 6).
- To discover the functionality and architecture of Amazon MQ, see How Amazon MQ Works (p. 32).
- To find out the guidelines and caveats that will help you make the most of Amazon MQ, see Best Practices for Amazon MQ (p. 66).
- 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*.

We Want to Hear from You

We welcome your feedback. To contact us, visit the Amazon MQ Discussion Forum.

Frequently Viewed Amazon MQ Topics

Latest update: October 8, 2018

Amazon MQ Developer Guide	*Amazon MQ REST API Reference*
1. Working Java Example (p. 55)	1. Broker
2. Connecting a Java Application to Your Broker (p. 22)	2. Configuration
3. Getting Started with Amazon MQ (p. 6)	3. User
4. Creating and Configuring a Broker (p. 12)	4. Resources
5. Amazon MQ Broker Configuration Parameters (p. 41)	5. Brokers
6. Connecting to Amazon MQ (p. 67)	6. Broker Reboot
7. Limits in Amazon MQ (p. 73)	7. Configurations
8. Using Amazon MQ Securely (p. 66)	8. Users
9. Amazon MQ Broker Architecture (p. 38)	9. Configuration Revision
10.Amazon MQ Active/Standby Broker for High Availability (p. 39)	10.Configuration Revisions

Setting Up Amazon MQ

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

Topics

Step 1: Create an AWS Account and an IAM Administrator User

To access any AWS service, you must first create an AWS account. This is an Amazon account that can use AWS products. You can use your AWS account to view your activity and usage reports and to manage authentication and access.

1. Navigate to the AWS home page, and then choose **Create an AWS Account**.
2. Follow the instructions.

   Part of the sign-up procedure involves receiving a phone call and entering a PIN using the phone keypad.
3. When you finish creating your AWS account, follow the instructions in the *IAM User Guide* to create your first IAM administrator user and group.

Step 2: Create an IAM User and Get Your AWS Credentials

To avoid using your IAM administrator user for Amazon MQ operations, it is a best practice to create an IAM user for each person who needs administrative access to Amazon MQ.

To work with Amazon MQ, you need the `AmazonMQFullAccess` policy and AWS credentials that are associated with your IAM user. These credentials are comprised of an access key ID and a secret access key. For more information, see What Is IAM? in the *IAM User Guide* and AWS Security Credentials in the *AWS General Reference*.

1. Sign in to the AWS Identity and Access Management console.
2. Choose **Users**, **Add user**.
3. Type a **User name**, such as `AmazonMQAdmin`.
4. Select **Programmatic access** and **AWS Management Console access**.
5. Set a **Console password** and then choose **Next: Permissions**.
6. On the **Set permissions for *`AmazonMQAdmin`*** page, choose **Attach existing policies directly**.
7. Type `AmazonMQ` into the filter, choose **AmazonMQFullAccess**, and then choose **Next: Review**.
8. On the **Review** page, choose **Create user**.

The IAM user is created and the **Access key ID** is displayed, for example:

AKIAIOSFODNN7EXAMPLE

9. To display your **Secret access key**, choose **Show**, for example:

   wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY

   Important
   You can view or download your secret access key *only* when you create your credentials (however, you can create new credentials at any time).
10. To download your credentials, choose **Download .csv**. Keep this file in a secure location.

Step 3: Get Ready to Use the Example Code

The following tutorials show how you can work with Amazon MQ and ActiveMQ using the AWS Management Console and Java. To use the 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 (p. 6) and then connecting a Java application (p. 22) to your broker.

Getting Started with Amazon MQ

This section will help you become more familiar with Amazon MQ by showing you how to create a broker and how to connect your application to it.

The following 3-minute video provides a preview of creating and using an Amazon MQ broker.

Topics

Prerequisites

Before you begin, complete the steps in Setting Up Amazon MQ (p. 4).

Step 1: Create 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`, `t2`) and *size* (`large`, `micro`) is a *broker instance type* (for example, `mq.m5.large`). For more information, see Broker (p. 32).

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. Do one of the following:
   - If this is your first time using Amazon MQ, in the **Create a broker** section, type `MyBroker` for **Broker name** and then choose **Next step**.
   - If you have created a broker before, on the **Create a broker** page, in the **Broker details** section, type `MyBroker` for **Broker name**.
3. In the **Broker details** section, choose a **Broker instance type** (for example, **mq.m5.large**). For more information, see Broker Instance Types (p. 33).
4. Choose a **Deployment mode**. In this example, **Single-instance broker** is selected.
   - A **Single-instance broker** is comprised of one broker in one Availability Zone. The broker communicates with your application and with an AWS storage location. For more information, see Amazon MQ Single-Instance Broker (p. 38).
   - 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 a shared storage location. For more information, see Amazon MQ Active/Standby Broker for High Availability (p. 39).
5. Choose a **Broker engine** version.

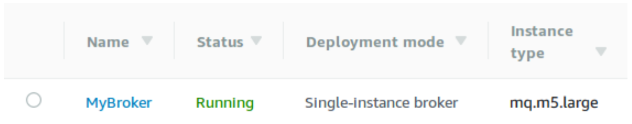
   Note
   Currently, Amazon MQ supports only `ActiveMQ` broker engine versions `5.15.6` and `5.15.0`.

6. In the **ActiveMQ Web Console access** section, type a **Username** and **Password**.
7. 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.



	Name	Status	Deployment mode	Instance type
○	MyBroker	Running	Single-instance broker	mq.m5.large

8. 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
```

Step 2: Connect a Java Application to Your Broker

After you create an Amazon MQ 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 (p. 55).

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

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

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 ActiveMQ 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 an ActiveMQ Web Console).

    a. Choose **Add Rule**.
    b. For **Type**, select **Custom TCP**.
    c. For **Port Range**, type the ActiveMQ 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 ActiveMQ 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>5.15.6</version>
    </dependency>
    <dependency>
        <groupId>org.apache.activemq</groupId>
        <artifactId>activemq-pool</artifactId>
        <version>5.15.6</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.

Create a Message Producer and Send 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 Amazon MQ Broker Architecture (p. 38).
    For wire-level protocol endpoints, you can 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 username and password.
```

```
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();
```

Note

Message producers should always use the `PooledConnectionFactory` class. For more information, see Always Use Connection Pooling (p. 68).

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();
```

Create a Message Consumer and Receive the 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 username and password.
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 (p. 68).

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();
pooledConnectionFactory.stop();
```

Step 3: Delete Your 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.

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.

Next Steps

Now that you have created a broker, connected an application to it, and sent and received a message, you might want to try the following:

- Creating and Configuring a Broker (p. 12) (Advanced Settings)
- Editing Broker Engine Version, CloudWatch Logs, and Maintenance Preferences (p. 16)
- Creating and Applying Broker Configurations (p. 17)
- Editing and Managing Broker Configurations (p. 19)
- Listing Brokers and Viewing Broker Details (p. 25)
- Creating and Managing Amazon MQ Broker Users (p. 26)
- Rebooting a Broker (p. 28)
- Accessing CloudWatch Metrics for Amazon MQ (p. 29)

You can also begin to dive deep into Amazon MQ best practices (p. 66) and Amazon MQ REST APIs, and then plan to migrate to Amazon MQ (p. 63).

Amazon MQ Tutorials

The following tutorials show how you can work with Amazon MQ and ActiveMQ using the AWS Management Console and Java. To use the example code, you must install the Java Standard Edition Development Kit and make some changes to the code.

Topics

Tutorial: Creating and Configuring an Amazon MQ 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`, `t2`) and *size* (`large`, `micro`) is a *broker instance type* (for example, `mq.m5.large`). For more information, see Broker (p. 32).

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 and configure a broker using the AWS Management Console.

Topics

Step 1: Configure Basic Broker Settings

1. Sign in to the Amazon MQ console.
2. Do one of the following:
   - If this is your first time using Amazon MQ, in the **Create a broker** section, type `MyBroker` for **Broker name** and then choose **Next step**.
   - If you have created a broker before, on the **Create a broker** page, in the **Broker details** section, type `MyBroker` for **Broker name**.
3. In the **Broker details** section, choose a **Broker instance type** (for example, **mq.m5.large**). For more information, see Broker Instance Types (p. 33).

4. Choose a **Deployment mode**:

   - A **Single-instance broker** is comprised of one broker in one Availability Zone. The broker communicates with your application and with an AWS storage location. For more information, see Amazon MQ Single-Instance Broker (p. 38).
   - 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 a shared storage location. For more information, see Amazon MQ Active/Standby Broker for High Availability (p. 39).

5. Choose a **Broker engine** version.

   Note
   Currently, Amazon MQ supports only `ActiveMQ` broker engine versions `5.15.6` and `5.15.0`.

6. In the **ActiveMQ Web Console access** section, type a **Username** and **Password**.

Step 2: (Optional) Configure Advanced Broker Settings

Important

- **Subnet(s)** – A single-instance broker requires one subnet (for example, the default subnet). An active/standby broker requires two subnets.
- **Security group(s)** – Both single-instance brokers and active/standby brokers require at least one security group (for example, the default security group).
- **VPC** – A broker's subnet(s) and security group(s) must be in the same VPC. EC2-Classic resources aren't supported.
- **Public accessibility** – Disabling public accessibility makes the broker accessible only within your VPC. For more information, see Prefer Brokers without Public Accessibility (p. 66) and Accessing the ActiveMQ Web Console of a Broker without Public Accessibility (p. 15).

1. Expand the **Advanced settings** section.
2. In the **Configuration** section, choose **Create a new configuration with default values** or **Select an existing configuration**. For more information, see Configuration (p. 36) and Amazon MQ Broker Configuration Parameters (p. 41).
3. In the **Logs** section, choose whether to publish **General** logs and **Audit** logs to Amazon CloudWatch Logs. For more information, see Configuring Amazon MQ to Publish Logs to Amazon CloudWatch Logs (p. 82).

   Important
   If you don't add the `CreateLogGroup` permission to your Amazon MQ user (p. 83) before the user creates or reboots the broker, Amazon MQ doesn't create the log group.
   If you don't configure a resource-based policy for Amazon MQ (p. 83), the broker can't publish the logs to CloudWatch Logs.

4. In the **Network and security section**, configure your broker's connectivity:

   a. Do one of the following:

      - Choose **Use the default VPC, subnet(s), and security group(s).**
      - Choose **Select existing VPC, subnet(s), and security group(s).**
        1. If you choose this option, you can create a new **Virtual Private Cloud (VPC)** on the Amazon VPC console, select an existing VPC, or select the default VPC. For more information, see What is Amazon VPC? in the *Amazon VPC User Guide*.

2. After you create or select a VPC, you can create new **Subnet(s)** on the Amazon VPC console or select existing ones. For more information, see VPCs and Subnets in the *Amazon VPC User Guide*.
3. After you create or select subnets, you can select the **Security group(s)**.

b. Choose the **Public accessibility** of your broker.

5. In the **Maintenance section**, configure your broker's maintenance schedule:

a. To upgrade the broker to new versions as Apache 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).

Note
For an active/standby broker, if one of the broker instances undergoes maintenance, it takes Amazon MQ a short while to take the inactive instance out of service, allowing the healthy standby instance to become active and to begin accepting incoming communications.

b. Do one of the following:

- To allow Amazon MQ to select the maintenance window automatically, choose **No preference**.
- To set a custom maintenance window, choose **Select maintenance window** and then specify the **Start day** and **Start time** of the upgrades.

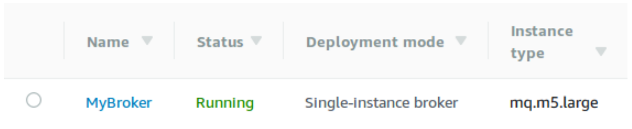
Step 3: Finish Creating the Broker

1. 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.



	Name	Status	Deployment mode	Instance type
○	MyBroker	Running	Single-instance broker	mq.m5.large

2. 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
```

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 Amazon MQ Broker Architecture (p. 38)).

For wire-level protocol endpoints, you can allow your application to connect to either endpoint by using the Failover Transport.

Accessing the ActiveMQ Web Console of a Broker without Public Accessibility

If you disable public accessibility for your broker, you must perform the following steps to be able to access your broker's ActiveMQ Web Console.

Note
The names of the VPCs and security groups are specific to the following example.

Prerequisites

To perform the following steps, you must configure the following:

- **VPCs**
  - The VPC without an internet gateway, to which the Amazon MQ broker is attached, named `private-vpc`.
  - A second VPC, with an internet gateway, named `public-vpc`.
  - Both VPCs must be connected (for example, using VPC peering) so that the Amazon EC2 instances in the public VPC can communicate with the EC2 instances in the private VPC.
  - If you use VPC peering, the route tables for both VPCs must be configured for the peering connection.
- **Security Groups**
  - The security group used to create the Amazon MQ broker, named `private-sg`.
  - A second security group used for the EC2 instance in the `public-vpc` VPC, named `public-sg`.
  - `private-sg` must allow inbound connections from `public-sg`. We recommend restricting this security group to port 8162.
  - `public-sg` must allow inbound connections from your machine on port 22.

To Access the ActiveMQ Web Console of a Broker without Public Accessibility

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 broker 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 ActiveMQ Web Console on your machine.

Tutorial: Editing Broker Engine Version, CloudWatch Logs, and Maintenance Preferences

In addition to editing broker configurations and managing configuration revisions (p. 19), you can configure preferences specific to the broker.

Note
All preferences except for those for automatic minor version upgrades require you to schedule modifications. For more information, see Amazon MQ Broker Configuration Lifecycle (p. 40).

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

To Edit Broker Engine Version, CloudWatch Logs, and Maintenance Preferences

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**.
4. In the **Configuration** section, select the configuration and revision for your broker. For more information, see Editing and Managing Broker Configurations (p. 19).
5. In the **CloudWatch Logs** section, choose whether to publish **General** logs and **Audit** logs to Amazon CloudWatch Logs. For more information, see Configuring Amazon MQ to Publish Logs to Amazon CloudWatch Logs (p. 82).

   Important
   If you don't add the `CreateLogGroup` permission to your Amazon MQ user (p. 83) before the user creates or reboots the broker, Amazon MQ doesn't create the log group.
   If you don't configure a resource-based policy for Amazon MQ (p. 83), the broker can't publish the logs to CloudWatch Logs.
6. 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).

   Note
   For an active/standby broker, if one of the broker instances undergoes maintenance, it takes Amazon MQ a short while to take the inactive instance out of service, allowing the healthy standby instance to become active and to begin accepting incoming communications.
7. 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.

Tutorial: Creating and Applying Amazon MQ 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. You can apply a configuration immediately or during a *maintenance window*.

> **Note**
> For an active/standby broker, if one of the broker instances undergoes maintenance, it takes Amazon MQ a short while to take the inactive instance out of service, allowing the healthy standby instance to become active and to begin accepting incoming communications.

For more information, see the following:

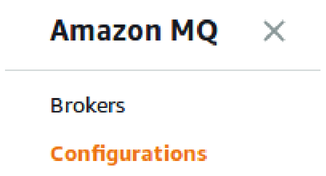
- Configuration (p. 36)
- Amazon MQ Broker Configuration Lifecycle (p. 40)
- Amazon MQ Broker Configuration Parameters (p. 41)
- Editing and Managing Broker Configurations (p. 19)

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

Topics

Step 1: Create a Configuration from Scratch

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**
   > Currently, Amazon MQ supports only `ActiveMQ` broker engine versions `5.15.6` and `5.15.0`.

5. Choose **Create configuration**.

Step 2: Create a New Configuration Revision

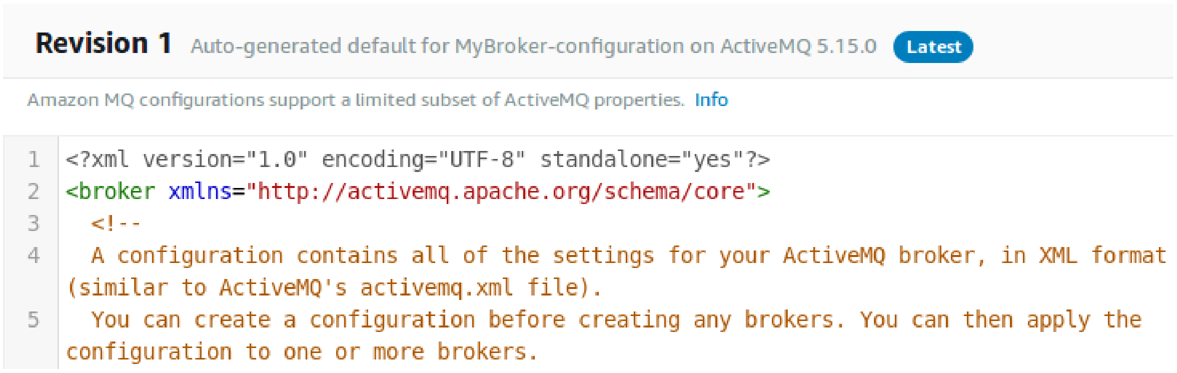
1. From the configuration list, choose ***`MyConfiguration`***.

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 5.15.6**) 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.



Revision 1 Auto-generated default for MyBroker-configuration on ActiveMQ 5.15.0
Latest
Amazon MQ configurations support a limited subset of ActiveMQ properties. Info

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<broker xmlns="http://activemq.apache.org/schema/core">
  <!--
  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.
```

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

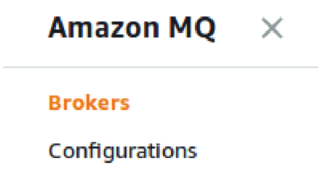
   The **Save revision** dialog box is displayed.
5. (Optional) Type `A description of the changes in this revision.`
6. Choose **Save**.

   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 (p. 41).
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 (p. 21) or reboot the broker (p. 28). For more information, see Amazon MQ Broker Configuration Lifecycle (p. 40).
Currently, it isn't possible to delete a configuration.

Step 3: Apply a 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
   Your broker will be offline while it is being rebooted.
5. Choose **Apply**.

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

Tutorial: Editing Amazon MQ Broker Configurations and Managing Configuration Revisions

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

Note
For an active/standby broker, if one of the broker instances undergoes maintenance, it takes Amazon MQ a short while to take the inactive instance out of service, allowing the healthy standby instance to become active and to begin accepting incoming communications.

To keep track of the changes you make to your configuration, you can create *configuration revisions*.

For more information, see the following:

- Configuration (p. 36)
- Amazon MQ Broker Configuration Lifecycle (p. 40)
- Amazon MQ Broker Configuration Parameters (p. 41)
- Creating and Applying Broker Configurations (p. 17)

The following examples show how you can edit Amazon MQ broker configurations and manage broker configuration revisions using the AWS Management Console.

Topics

To View a Previous 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 **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 5.15.6**) are displayed.

4. Choose **Revision history**.
5. The configuration **Revision** number, **Revision date**, and **Description** are displayed for each revision.
6. Select a revision and choose **View details**.

   The broker configuration in XML format is displayed.

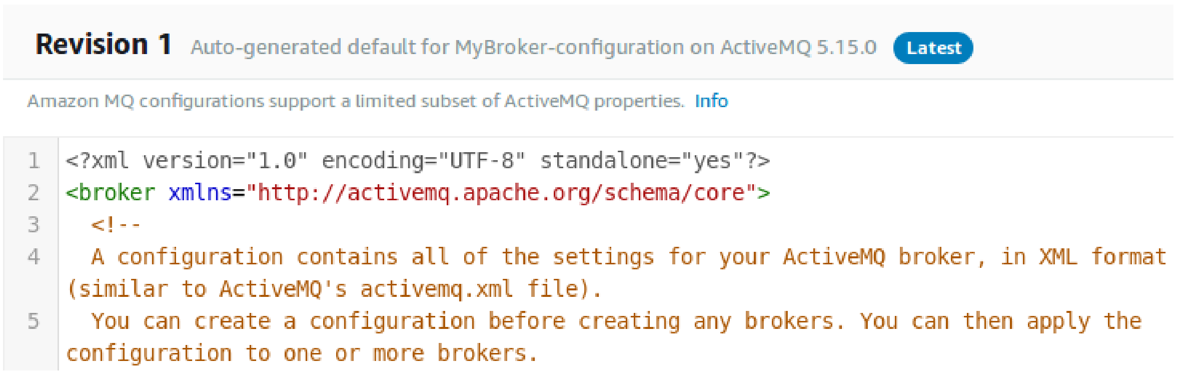
To Edit the Current Configuration Revision

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 5.15.6**) 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.


Revision 1
Auto-generated default for MyBroker-configuration on ActiveMQ 5.15.0
Latest
Amazon MQ configurations support a limited subset of ActiveMQ properties.
Info

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<broker xmlns="http://activemq.apache.org/schema/core">
  <!--
  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.
```

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 (p. 41).
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 (p. 21) or reboot the broker (p. 28). For more information, see Amazon MQ Broker Configuration Lifecycle (p. 40).
Currently, it isn't possible to delete a configuration.

To Apply a Configuration Revision to Your Broker

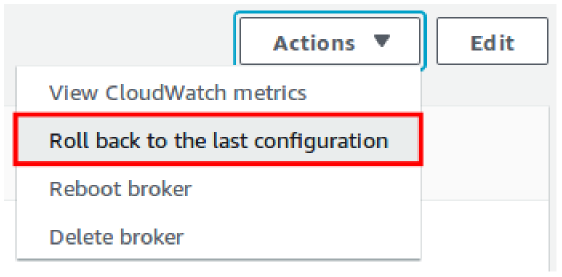
1. Sign in to the Amazon MQ console.
2. From the broker list, select your broker (for example, **MyBroker**) and then choose **Edit**.
3. On the **Edit** ***`MyBroker`*** page, in the **Configuration** section, select a **Configuration** and a **Revision** and then choose **Schedule Modifications**.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

   Important
   Your broker will be offline while it is being rebooted.
5. Choose **Apply**.

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

To Roll Back Your Broker to the Last Configuration Revision

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**, **Roll back to last configuration**.

4. (Optional) To review the **Current configuration** or the **Last configuration**, on the **Roll back to the last configuration** page, in the **Summary** section, choose **Edit** for either configuration.
5. In the **Schedule broker modifications** section, choose whether to apply modifications **During the next scheduled maintenance window** or **Immediately**.

   Important
   Your broker will be offline while it is being rebooted.
6. Choose **Apply**.

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

Tutorial: Connecting a Java Application to Your Amazon MQ Broker

After you create an Amazon MQ 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 (p. 55).

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

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 ActiveMQ 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 an ActiveMQ Web Console).

   a. Choose **Add Rule**.
   b. For **Type**, select **Custom TCP**.
   c. For **Port Range**, type the ActiveMQ 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 ActiveMQ 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>5.15.6</version>
    </dependency>
    <dependency>
        <groupId>org.apache.activemq</groupId>
        <artifactId>activemq-pool</artifactId>
        <version>5.15.6</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

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 Amazon MQ Broker Architecture (p. 38).
   For wire-level protocol endpoints, you can 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 username and password.
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();
```

   Note
   Message producers should always use the `PooledConnectionFactory` class. For more information, see Always Use Connection Pooling (p. 68).

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

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 username and password.
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 (p. 68).

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();
pooledConnectionFactory.stop();
```

Tutorial: Listing Amazon MQ Brokers and Viewing Broker Details

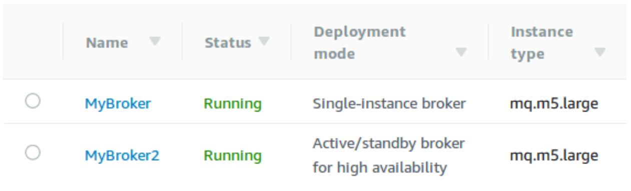
When you request that Amazon MQ create a broker, the creation process can take about 15 minutes..

The following example shows how you can confirm your broker's existence by listing your brokers in the current region using the AWS Management Console.

To List Brokers and View Broker Details

1. Sign in to the Amazon MQ console.

   Your brokers in the current region are listed.



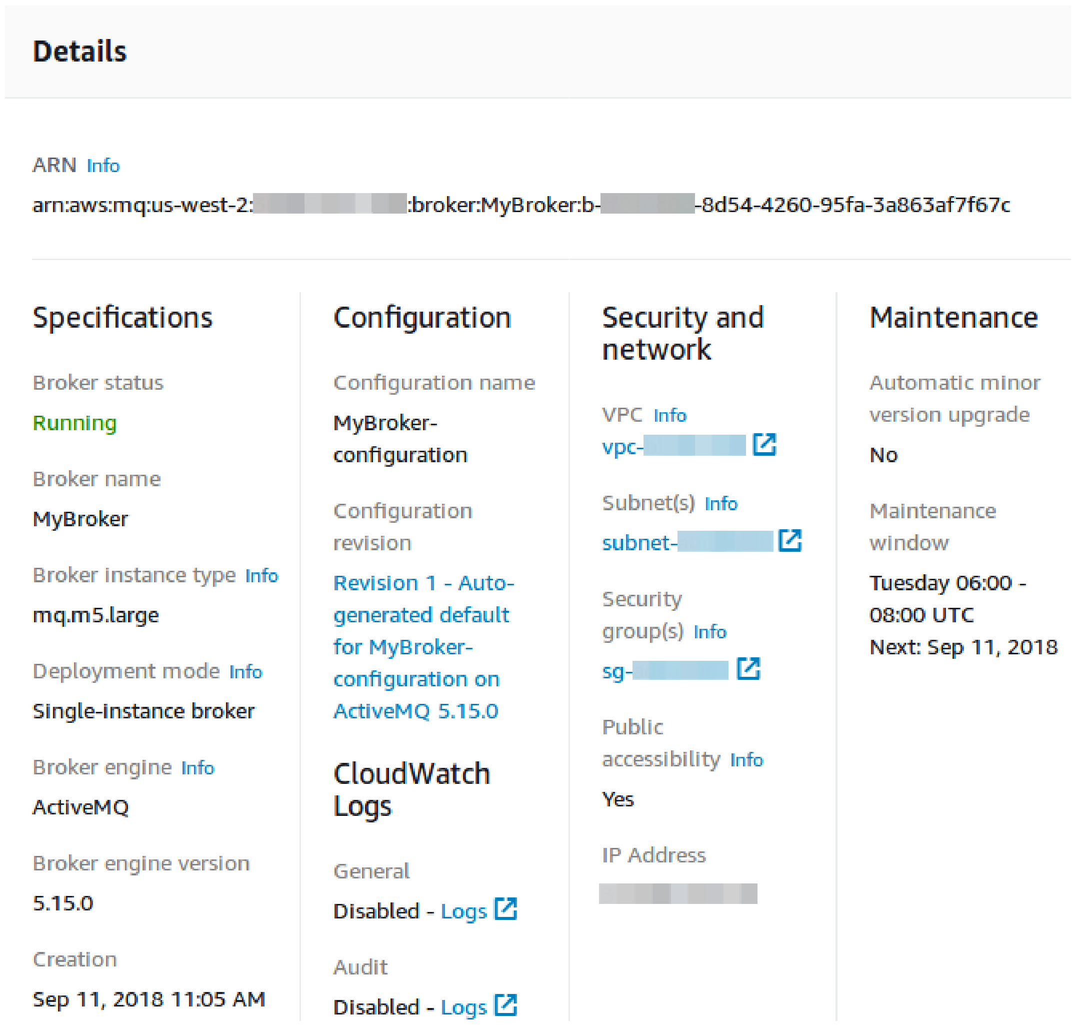
	Name	Status	Deployment mode	Instance type
○	MyBroker	Running	Single-instance broker	mq.m5.large
○	MyBroker2	Running	Active/standby broker for high availability	mq.m5.large

   The following information is displayed for each broker:

   - **Name**
   - **Creation** date
   - **Status** (p. 36)
   - **Deployment mode** (p. 38)
   - **Instance type** (p. 33)

2. Choose your broker's name (for example, **MyBroker**).

   On the ***MyBroker*** page, the configured (p. 36) **Details** are displayed for your broker:



Details

ARN Info

arn:aws:mq:us-west-2:[redacted]:broker:MyBroker:b-[redacted]-8d54-4260-95fa-3a863af7f67c

Specifications

Broker status
Running

Broker name
MyBroker

Broker instance type Info
mq.m5.large

Deployment mode Info
Single-instance broker

Broker engine Info
ActiveMQ

Broker engine version
5.15.0

Creation
Sep 11, 2018 11:05 AM

Configuration

Configuration name
MyBroker-configuration

Configuration revision
Revision 1 - Auto-generated default for MyBroker-configuration on ActiveMQ 5.15.0

CloudWatch Logs

General
Disabled - Logs

Audit
Disabled - Logs

Security and network

VPC Info
vpc-[redacted]

Subnet(s) Info
subnet-[redacted]

Security group(s) Info
sg-[redacted]

Public accessibility Info
Yes

IP Address
[redacted]

Maintenance

Automatic minor version upgrade
No

Maintenance window
Tuesday 06:00 - 08:00 UTC
Next: Sep 11, 2018

Below the **Details** section, the following information is displayed:

- In the **Connections** section, the ActiveMQ Web Console URL and the wire-level protocol endpoints
- In the **Users** section, the users (p. 37) associated with the broker

Tutorial: Creating and Managing Amazon MQ 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.

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.

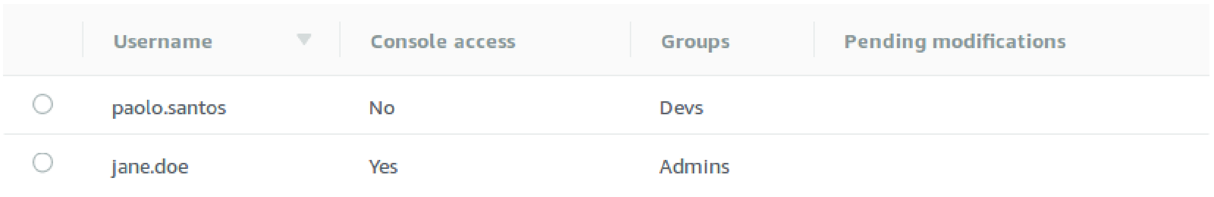
The following examples show how you can create, edit, and delete Amazon MQ broker users using the AWS Management Console.

Topics

To Create a New 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 **Edit**.

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



	Username	Console access	Groups	Pending modifications
○	paolo.santos	No	Devs	
○	jane.doe	Yes	Admins	

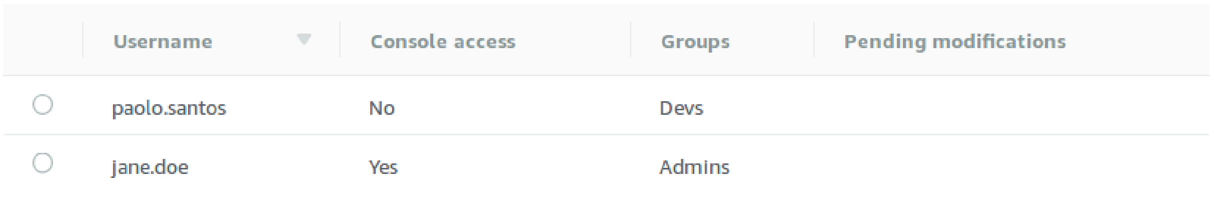
3. Choose **Create user**.
4. In the **Create user** dialog box, type a **Username** and **Password**.
5. (Optional) Type the names of groups to which the user belongs, separated by commas (for example: `Devs, Admins`).
6. (Optional) To enable the user to access the ActiveMQ Web Console, choose **ActiveMQ 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 (p. 21) or reboot the broker (p. 28). For more information, see Amazon MQ Broker Configuration Lifecycle (p. 40).

To edit an existing 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 **Edit**.

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



	Username	Console access	Groups	Pending modifications
○	paolo.santos	No	Devs	
○	jane.doe	Yes	Admins	

3. Select a username 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`).

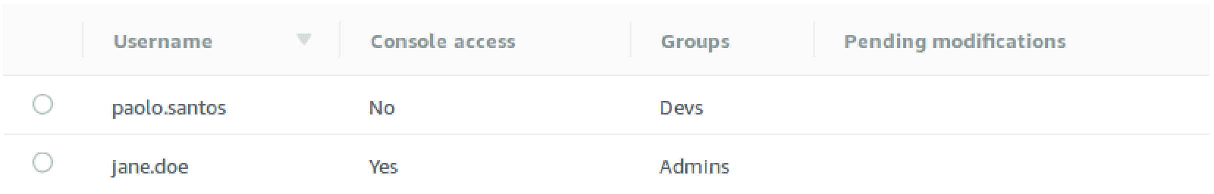
6. (Optional) To enable the user to access the ActiveMQ Web Console, choose **ActiveMQ Web Console**.
7. To save the changes to the user, choose **Done**.

   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 (p. 21) or reboot the broker (p. 28). For more information, see Amazon MQ Broker Configuration Lifecycle (p. 40).

To Delete an Existing 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 **Edit**.

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



	Username	Console access	Groups	Pending modifications
○	paolo.santos	No	Devs	
○	jane.doe	Yes	Admins	

3. Select a username (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 (p. 21) or reboot the broker (p. 28). For more information, see Amazon MQ Broker Configuration Lifecycle (p. 40).

Tutorial: Rebooting an Amazon MQ Broker

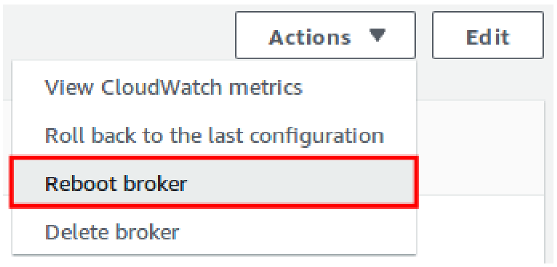
To apply a new configuration to a broker, you can reboot the broker. In addition, if your broker becomes unresponsive, you can reboot it to recover from a faulty state.

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
   Your broker will be offline while it is being rebooted.



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

   Rebooting the broker takes about 5 minutes.

Tutorial: 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.

To Delete 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.

Tutorial: Accessing CloudWatch Metrics for Amazon MQ

Amazon MQ and Amazon CloudWatch are integrated so you can use CloudWatch to view and analyze metrics for your ActiveMQ broker and the broker's destinations (queues and topics). You can view and analyze your Amazon MQ metrics from the CloudWatch console, the AWS CLI, or the CloudWatch CLI. CloudWatch metrics for Amazon MQ are automatically polled from the broker and then pushed to CloudWatch every minute.

For a full list of Amazon MQ metrics, see Monitoring Amazon MQ Using CloudWatch (p. 76).

For information about creating a CloudWatch alarm for a metrics, see Create or Edit a CloudWatch Alarm in the *Amazon CloudWatch User Guide*.

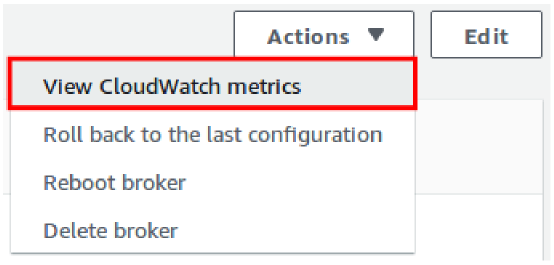
Note
There is no charge for the Amazon MQ metrics reported in CloudWatch. These metrics are provided as part of the Amazon MQ service.
CloudWatch monitors only the first 200 destinations.

Topics

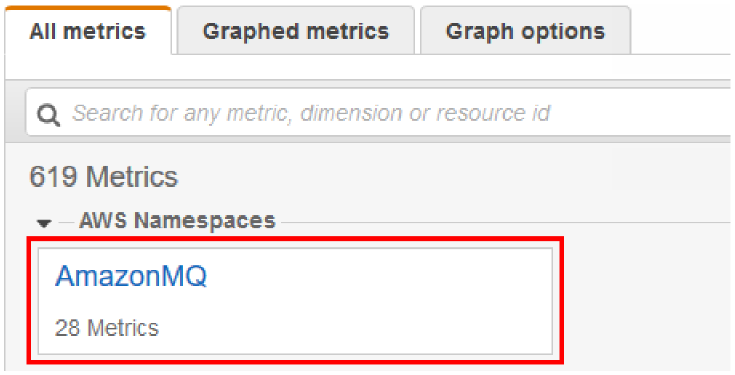
AWS Management Console

The following example shows you how to access CloudWatch metrics for Amazon MQ using the AWS Management Console.

Note
If you're already signed into the Amazon MQ console, on the broker **Details** page, choose **Actions**, **View CloudWatch metrics**.

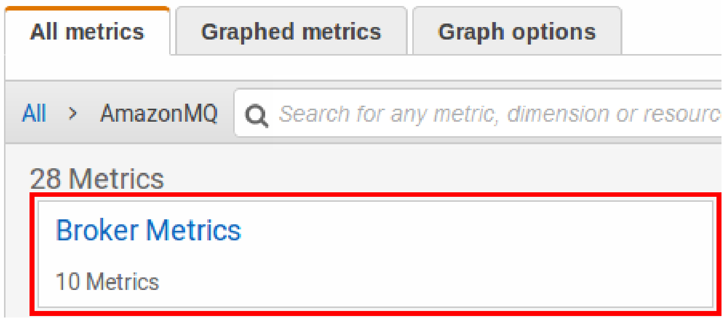


1. Sign in to the CloudWatch console.
2. On the navigation panel, choose **Metrics**.
3. Select the **AmazonMQ** metric namespace.



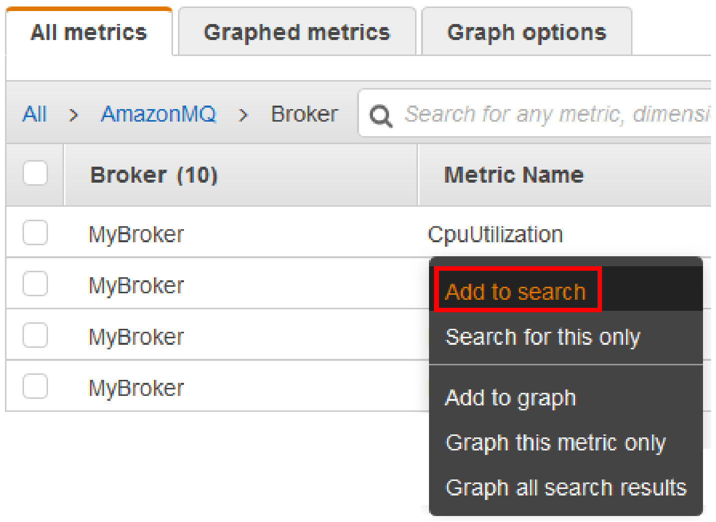
4. Select one of the following metric dimensions:
   - **Broker Metrics**
   - **Queue Metrics by Broker**
   - **Topic Metrics by Broker**

   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**.



AWS Command Line Interface

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*.

Amazon CloudWatch API

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*.

How Amazon MQ Works

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, lists available Amazon MQ broker instance types and their statuses, provides an overview of broker architecture, explains broker configuration parameters and offers a working example of using Java Message Service (JMS) with an ActiveMQ broker.

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

Topics

Amazon MQ Basic Elements

This section introduces key concepts essential to understanding Amazon MQ.

Topics

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`, `t2`) and *size* (`large`, `micro`) is a *broker instance type* (for example, `mq.m5.large`). For more information, see Broker Instance Types (p. 33).

- A *single-instance broker* is comprised of one broker in one Availability Zone. The broker communicates with your application and with an AWS storage location.
- 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 a shared storage location.

For more information, see Amazon MQ Broker Architecture (p. 38).

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:

- Creating and Configuring a Broker (p. 12)
- Limits Related to Brokers (p. 73)

- Broker Statuses (p. 36)

Attributes

A 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

Instance Types

The combined description of the broker instance *class* (`m5`, `t2`) and *size* (`large`, `micro`) is a *broker instance type* (for example, `mq.m5.large`). The following table lists the available Amazon MQ broker instance types.

Instance Type	vCPU	Memory (GiB)	Network Performance	Notes
`mq.t2.micro`	1	1	Low	Use the `mq.t2.micro` instance type for basic evaluation of Amazon MQ. This instance type (single-instance brokers only) qualifies for the AWS Free Tier. **Note** 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` (p. 76) metric). If your application requires *fixed performance*, consider using an `mq.m5.large` instance type.
`mq.m5.large`	2	8	High	Use the `mq.m5.large` instance for regular development, testing, and production workloads.

Instance Type	vCPU	Memory (GiB)	Network Performance	Notes
`mq.m5.xlarge`	4	16	High	Use the `mq.m5.xlarge`, `mq.m5.2xlarge`, and `mq.m5.4xlarge` instance types for regular development, testing and production workloads that require high throughput. **Note** When your system uses persistent messages, its throughput depends on how quickly messages are consumed. If messages aren't consumed immediately, using larger instance types with persistent messages might not improve system throughput. In this case, we recommend setting the `concurrentStoreAndDisp` attribute to `false`. For more
`mq.m5.2xlarge`	8	32	High	
`mq.m5.4xlarge`	16	64	High	

Instance Type	vCPU	Memory (GiB)	Network Performance	Notes
				information, see Disable Concurrent Store and Dispatch for Queues with Slow Consumers (p. 69).
`mq.m4.large`	2	8	Moderate	Use the `mq.m4.large` instance type for compatibility with existing broker deployments. We recommend using an `mq.m5.*` instance for new brokers.

For more information about throughput considerations, see Choose the Correct Broker Instance Type for the Best Throughput (p. 70).

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.

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.

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 (p. 21) or reboot the broker (p. 28). For more information, see Amazon MQ Broker Configuration Lifecycle (p. 40). Currently, it isn't possible to delete a configuration.

For information about creating, editing, and managing configurations, see the following:

- Creating and Applying Broker Configurations (p. 17)
- Editing and Managing Broker Configurations (p. 19)
- Limits Related to Configurations (p. 73)
- Amazon MQ Broker Configuration Parameters (p. 41)

To keep track of the changes you make to your configuration, you can create *configuration revisions*. For more information, see Creating and Applying Broker Configurations (p. 17) and Editing and Managing Broker Configurations (p. 19).

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:MyConfiguration: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

Engine

A *broker engine* is a type of message broker that runs on Amazon MQ.

Amazon MQ supports the following versions of ActiveMQ:

- ActiveMQ 5.15.0
- ActiveMQ 5.15.6

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.

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 (p. 21) or reboot the broker (p. 28). For more information, see Amazon MQ Broker Configuration Lifecycle (p. 40).

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 and Managing Amazon MQ Broker Users (p. 26)
- Limits Related to Users (p. 74)

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

Amazon MQ Broker Architecture

Amazon MQ brokers can be created as *single-instance brokers* or *active/standby brokers*. For both deployment modes, Amazon MQ provides high durability by storing its data redundantly, across multiple Availability Zones (multi-AZs) within an AWS Region. Amazon MQ ensures high availability by providing failover to a standby instance in a second Availability Zone.

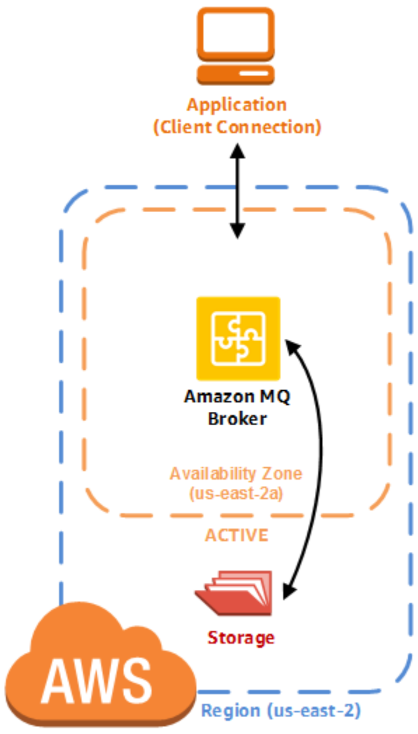
Note
Amazon MQ uses Apache KahaDB as its data store. Other data stores, such as JDBC and LevelDB, aren't supported.

Topics

Amazon MQ Single-Instance Broker

A *single-instance broker* is comprised of one broker in one Availability Zone. The broker communicates with your application and with an AWS storage location.

The following diagram illustrates a single-instance broker.



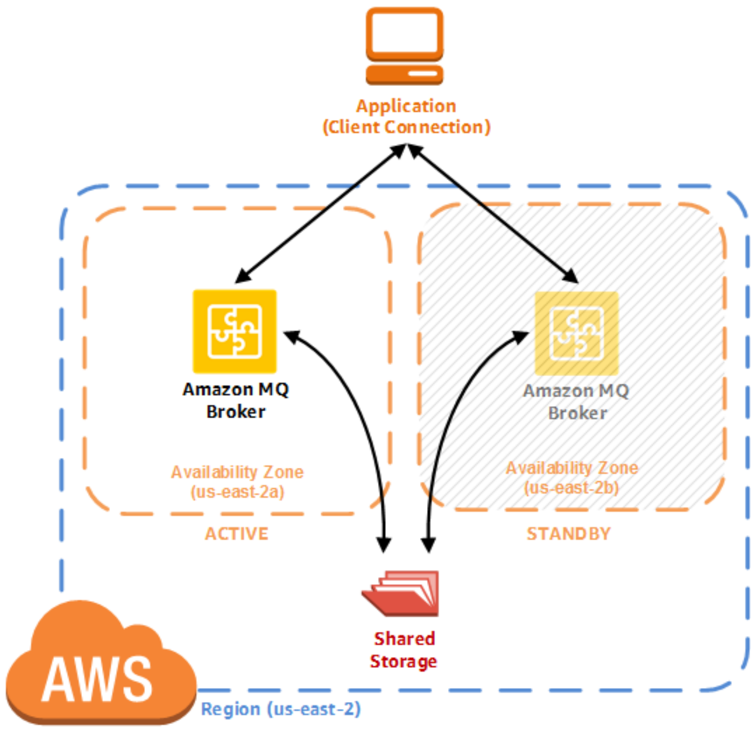
Amazon MQ Active/Standby Broker for High Availability

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 a shared storage location.

Normally, 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, allowing the healthy standby instance to become active and to begin accepting incoming communications. 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 can allow your application to connect to either endpoint by using the Failover Transport.

The following diagram illustrates an active/standby broker.

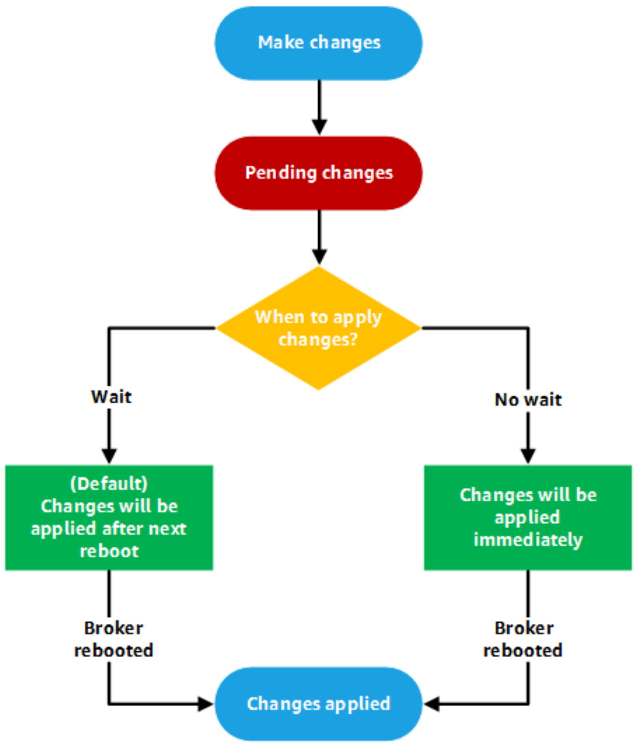


Amazon MQ Broker Configuration Lifecycle

Making changes to a configuration revision or an ActiveMQ user does *not* apply the changes immediately. To apply your changes, you must wait for the next maintenance window (p. 21) or reboot the broker (p. 28). For more information, see Amazon MQ Broker Configuration Lifecycle (p. 40).

The following diagram illustrates the configuration lifecycle.

Important
The next scheduled maintenance window triggers a reboot. If the broker is rebooted before the next scheduled maintenance window, the changes are applied after the reboot.



For information about creating, editing, and managing configurations, see the following:

- Creating and Applying Broker Configurations (p. 17)
- Editing and Managing Broker Configurations (p. 19)
- Amazon MQ Broker Configuration Parameters (p. 41)

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

- Creating and Managing Amazon MQ Broker Users (p. 26)
- Limits Related to Users (p. 74)

Amazon MQ Broker Configuration Parameters

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. For more information, see the following:

- Configuration (p. 36)
- Creating and Applying Broker Configurations (p. 17)
- Editing and Managing Broker Configurations (p. 19)
- Limits Related to Configurations (p. 73)

Working with 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 schema. You can use this schema 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.

Topics

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) (p. 44)
`abortSlowConsumerStrategy` (attributes) (p. 44)
`authorizationEntry` (attributes) (p. 44)
`authorizationMap` (child collection elements) (p. 51)
`authorizationPlugin` (child collection elements) (p. 51)
`broker` (attributes (p. 45) | child collection elements) (p. 51)
`cachedMessageGroupMapFactory` (attributes) (p. 46)
`compositeQueue` (attributes (p. 46) | child collection elements) (p. 51)
`compositeTopic` (attributes (p. 46) | child collection elements) (p. 52)
`constantPendingMessageLimitStrategy` (attributes) (p. 46)
`discarding` (attributes) (p. 46)
`discardingDLQBrokerPlugin` (attributes) (p. 46)
`fileCursor`
`fileDurableSubscriberCursor`
`fileQueueCursor`
`filteredDestination` (attributes) (p. 46)
`fixedCountSubscriptionRecoveryPolicy` (attributes) (p. 46)

Element
`fixedSizedSubscriptionRecoveryPolicy` (attributes) (p. 46)
`forcePersistencyModeBrokerPlugin` (attributes) (p. 46)
`individualDeadLetterStrategy` (attributes) (p. 47)
`lastImageSubscriptionRecoveryPolicy`
`messageGroupHashBucketFactory` (attributes) (p. 47)
`mirroredQueue` (attributes) (p. 47)
`noSubscriptionRecoveryPolicy`
`oldestMessageEvictionStrategy` (attributes) (p. 47)
`oldestMessageWithLowestPriorityEvictionStrategy` (attributes) (p. 47)
`policyEntry` (attributes (p. 47) | child collection elements) (p. 52)
`policyMap` (child collection elements) (p. 53)
`prefetchRatePendingMessageLimitStrategy` (attributes) (p. 49)
`priorityDispatchPolicy`
`priorityNetworkDispatchPolicy`
`queryBasedSubscriptionRecoveryPolicy` (attributes) (p. 49)
`queue` (attributes) (p. 49)
`redeliveryPlugin` (attributes (p. 49) | child collection elements) (p. 53)
`redeliveryPolicy` (attributes) (p. 49)
`redeliveryPolicyMap` (child collection elements) (p. 53)
`retainedMessageSubscriptionRecoveryPolicy` (child collection elements) (p. 53)
`roundRobinDispatchPolicy`
`sharedDeadLetterStrategy` (attributes (p. 49) | child collection elements) (p. 53)
`simpleDispatchPolicy`
`simpleMessageGroupMapFactory`
`statisticsBrokerPlugin`
`storeCursor`
`storeDurableSubscriberCursor` (attributes) (p. 50)
`strictOrderDispatchPolicy`
`tempDestinationAuthorizationEntry` (attributes) (p. 50)
`tempQueue` (attributes) (p. 50)
`tempTopic` (attributes) (p. 50)

Element
`timedSubscriptionRecoveryPolicy` (attributes) (p. 50)
`timeStampingBrokerPlugin` (attributes) (p. 50)
`topic` (attributes) (p. 50)
`uniquePropertyMessageEvictionStrategy` (attributes) (p. 50)
`virtualDestinationInterceptor` (child collection elements) (p. 53)
`virtualTopic` (attributes) (p. 50)
`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`

Element	Attribute
	`read`
	`tempQueue`
	`tempTopic`
	`topic`
	`write`
`broker`	`advisorySupport`
	`allowTempAutoCreationOnSend`
	`cacheTempDestinations`
	`consumerSystemUsagePortion`
	`dedicatedTaskRunner`
	`deleteAllMessagesOnStartup`
	`keepDurableSubsActive`
	`maxPurgedDestinationsPerSweep`
	`monitorConnectionSplits`
	`offlineDurableSubscriberTaskSchedule`
	`offlineDurableSubscriberTimeout`
	`persistenceThreadPriority`
	`persistent`
	`populateJMSXUserID`
	`producerSystemUsagePortion`
	`rejectDurableConsumers`
	`rollbackOnlyOnAsyncException`
	`schedulePeriodForDestinationPurge`
	`schedulerSupport`
	`splitSystemUsageForProducersConsumers`
	`taskRunnerPriority`
	`timeBeforePurgeTempDestinations`
	`useAuthenticatedPrincipalForJMSXUserID`
	`useMirroredQueues`
	`useTempMirroredQueues`
	`useVirtualDestSubs`

Element	Attribute
	useVirtualDestSubsOnCreation
	useVirtualTopics
cachedMessageGroupMapFactory	cacheSize
compositeQueue	concurrentSend
	copyMessage
	forwardOnly
	name
compositeTopic	concurrentSend
	copyMessage
	forwardOnly
	name
constantPendingMessageLimitStrategy	limit
discarding	deadLetterQueue
	enableAudit
	expiration
	maxAuditDepth
	maxProducersToAudit
	processExpired
	processNonPersistent
discardingDLQBrokerPlugin	dropAll
	dropOnly
	dropTemporaryQueues
	dropTemporaryTopics
	reportInterval
filteredDestination	queue
	selector
	topic
fixedCountSubscriptionRecoveryPolicy	maximumSize
fixedSizedSubscriptionRecoveryPolicy	maximumSize
	useSharedBuffer
forcePersistencyModeBrokerPlugin	persistenceFlag

Element	Attribute
individualDeadLetterStrategy	destinationPerDurableSubscriber
	enableAudit
	expiration
	maxAuditDepth
	maxProducersToAudit
	processExpired
	processNonPersistent
	queuePrefix
	queueSuffix
	topicPrefix
	topicSuffix
	useQueueForQueueMessages
	useQueueForTopicMessages
messageGroupHashBucketFactory	bucketCount
	cacheSize
mirroredQueue	copyMessage
	postfix
	prefix
oldestMessageEvictionStrategy	evictExpiredMessagesHighWatermark
oldestMessageWithLowestPriorityEvictionStrategy	evictExpiredMessagesHighWatermark
policyEntry	advisoryForConsumed
	advisoryForDelivery
	advisoryForDiscardingMessages
	advisoryForFastProducers
	advisoryForSlowConsumers
	advisoryWhenFull
	allConsumersExclusiveByDefault
	alwaysRetroactive
	blockedProducerWarningInterval
	consumersBeforeDispatchStarts
	cursorMemoryHighWaterMark

Element	Attribute
	`doOptimzeMessageStorage`
	`durableTopicPrefetch`
	`enableAudit`
	`expireMessagesPeriod`
	`gcInactiveDestinations`
	`gcWithNetworkConsumers`
	`inactiveTimeoutBeforeGC`
	`inactiveTimoutBeforeGC`
	`includeBodyForAdvisory`
	`lazyDispatch`
	`maxAuditDepth`
	`maxBrowsePageSize`
	`maxDestinations`
	`maxExpirePageSize`
	`maxPageSize`
	`maxProducersToAudit`
	`maxQueueAuditDepth`
	`memoryLimit`
	`messageGroupMapFactoryType`
	`minimumMessageSize`
	`optimizedDispatch`
	`optimizeMessageStoreInFlightLimit`
	`persistJMSRedelivered`
	`prioritizedMessages`
	`producerFlowControl`
	`queue`
	`queueBrowserPrefetch`
	`queuePrefetch`
	`reduceMemoryFootprint`
	`sendAdvisoryIfNoConsumers`
	`storeUsageHighWaterMark`

Element	Attribute
	strictOrderDispatch
	tempQueue
	tempTopic
	timeBeforeDispatchStarts
	topic
	topicPrefetch
	useCache
	useConsumerPriority
	usePrefetchExtension
prefetchRatePendingMessageLimitStrategy	multiplier
queryBasedSubscriptionRecoveryPolicy	query
queue	DLQ
	physicalName
redeliveryPlugin	fallbackToDeadLetter
	sendToDlqIfMaxRetriesExceeded
redeliveryPolicy	backOffMultiplier
	collisionAvoidancePercent
	initialRedeliveryDelay
	maximumRedeliveries
	maximumRedeliveryDelay
	preDispatchCheck
	queue
	redeliveryDelay
	tempQueue
	tempTopic
	topic
	useCollisionAvoidance
	useExponentialBackOff
sharedDeadLetterStrategy	enableAudit
	expiration
	maxAuditDepth

Element	Attribute
	maxProducersToAudit
	processExpired
	processNonPersistent
storeDurableSubscriberCursor	immediatePriorityDispatch
	useCache
tempDestinationAuthorizationEntry	admin
	queue
	read
	tempQueue
	tempTopic
	topic
	write
tempQueue	DLQ
	physicalName
tempTopic	DLQ
	physicalName
timedSubscriptionRecoveryPolicy	zeroExpirationOverride
timeStampingBrokerPlugin	recoverDuration
	futureOnly
	processNetworkMessages
	ttlCeiling
topic	DLQ
	physicalName
uniquePropertyMessageEvictionStrategy	evictExpiredMessagesHighWatermark
	propertyName
virtualTopic	concurrentSend
	local
	name
	postfix
	prefix
	selectorAware

Element	Attribute
	`transactedSend`

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` (`attributes`) (p. 54)
		`tempDestinationAuthorizationEntry`
	`defaultEntry`	`authorizationEntry`
		`tempDestinationAuthorizationEntry`
	`tempDestinationAuthorizationEntry`	`tempDestinationAuthorizationEntry`
`authorizationPlugin`	`map`	`authorizationMap`
`broker`	`destinationInterceptors`	`mirroredQueue`
		`virtualDestinationInterceptor`
	`destinationPolicy`	`policyMap`
	`destinations`	`queue`
		`tempQueue`
		`tempTopic`
		`topic`
	`persistenceAdapter`	`kahaDB` (`attributes`) (p. 54)
	`plugins`	`authorizationPlugin`
		`discardingDLQBrokerPlugin`
		`forcePersistencyModeBrokerPlugin`
		`redeliveryPlugin`
		`statisticsBrokerPlugin`
		`timeStampingBrokerPlugin`
`compositeQueue`	`forwardTo`	`queue`
		`tempQueue`

Element	Child Collection Element	Child Element
		tempTopic
		topic
		filteredDestination
compositeTopic	forwardTo	queue
		tempQueue
		tempTopic
		topic
		filteredDestination
policyEntry	deadLetterStrategy	discarding
		individualDeadLetterStrategy
		sharedDeadLetterStrategy
	destination	queue
		tempQueue
		tempTopic
		topic
	dispatchPolicy	priorityDispatchPolicy
		priorityNetworkDispatchPolicy
		roundRobinDispatchPolicy
		simpleDispatchPolicy
		strictOrderDispatchPolicy
	messageEvictionStrategy	oldestMessageEvictionStrategy
		oldestMessageWithLowestPriorityEvict
		uniquePropertyMessageEvictionStrateg
	messageGroupMapFactory	cachedMessageGroupMapFactory
		messageGroupHashBucketFactory
		simpleMessageGroupMapFactory
	pendingDurableSubscriberPolicy	fileDurableSubscriberCursor
		storeDurableSubscriberCursor
		vmDurableCursor
	pendingMessageLimitStrategy	constantPendingMessageLimitStrategy
		prefetchRatePendingMessageLimitStrat

Element	Child Collection Element	Child Element
	`pendingQueuePolicy`	`fileQueueCursor`
		`storeCursor`
		`vmQueueCursor`
	`pendingSubscriberPolicy`	`fileCursor`
		`vmCursor`
	`slowConsumerStrategy`	`abortSlowAckConsumerStrategy`
		`abortSlowConsumerStrategy`
	`subscriptionRecoveryPolicy`	`fixedCountSubscriptionRecoveryPolicy`
		`fixedSizedSubscriptionRecoveryPolicy`
		`lastImageSubscriptionRecoveryPolicy`
		`noSubscriptionRecoveryPolicy`
		`queryBasedSubscriptionRecoveryPolicy`
		`retainedMessageSubscriptionRecoveryP`
		`timedSubscriptionRecoveryPolicy`
`policyMap`	`defaultEntry`	`policyEntry`
	`policyEntries`	`policyEntry`
`redeliveryPlugin`	`redeliveryPolicyMap`	`redeliveryPolicyMap`
`redeliveryPolicyMap`	`defaultEntry`	`redeliveryPolicy`
	`redeliveryPolicyEntries`	`redeliveryPolicy`
`retainedMessageSubscriptionRecoveryPolicy`	`wrapped`	`fixedCountSubscriptionRecoveryPolicy`
		`fixedSizedSubscriptionRecoveryPolicy`
		`lastImageSubscriptionRecoveryPolicy`
		`noSubscriptionRecoveryPolicy`
		`queryBasedSubscriptionRecoveryPolicy`
		`retainedMessageSubscriptionRecoveryP`
		`timedSubscriptionRecoveryPolicy`
`sharedDeadLetterStrategy`	`deadLetterQueue`	`queue`
		`tempQueue`
		`tempTopic`
		`topic`
`virtualDestinationInterceptor`	`virtualDestinations`	`compositeQueue`

Element	Child Collection Element	Child Element
		`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.

authorizationEntry

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

admin|read|write

Amazon MQ Default: Not configured.

Example 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>
```

For more information, see Always Configure an Authorization Map (p. 67).

kahaDB

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

concurrentStoreAndDispatchQueues

Amazon MQ Default: `true`

Example Example Configuration

```
<persistenceAdapter>
   <kahaDB concurrentStoreAndDispatchQueues="false"/>
</persistenceAdapter>
```

For more information, see Disable Concurrent Store and Dispatch for Queues with Slow Consumers (p. 69).

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 (p. 22).
- 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 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 ActiveMQ 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 an ActiveMQ Web Console).

   a. Choose **Add Rule**.
   b. For **Type**, select **Custom TCP**.
   c. For **Port Range**, type the ActiveMQ 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 ActiveMQ 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>5.15.6</version>
    </dependency>
    <dependency>
        <groupId>org.apache.activemq</groupId>
        <artifactId>activemq-pool</artifactId>
        <version>5.15.6</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 2010-2018 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 username and password.
        connectionFactory.setUserName(ACTIVE_MQ_USERNAME);
        connectionFactory.setPassword(ACTIVE_MQ_PASSWORD);
        return connectionFactory;
    }
}
```

MQTT

```
/*
 * Copyright 2010-2018 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 username and password.
        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 2010-2018 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);
        }
    }
}
```

Migrating to Amazon MQ

Use the following topics to get started with migrating your on-premises message broker to Amazon MQ.

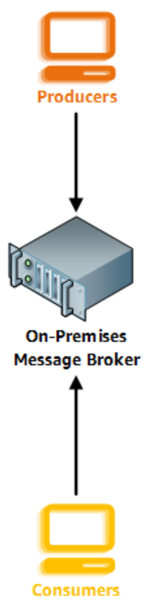
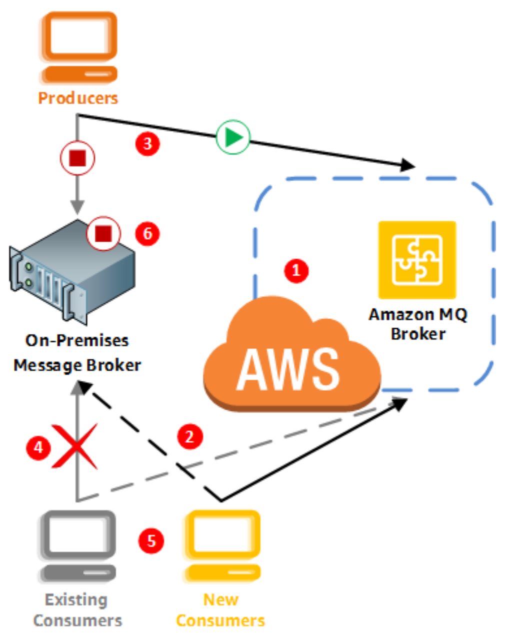
Topics

For detailed information and examples, see Migrating from RabbitMQ to Amazon MQ in the *AWS Compute Blog*.

Migrating to Amazon MQ without Service Interruption

The following diagrams illustrate the scenario of migrating from an on-premises message broker to an Amazon MQ broker in the AWS Cloud without service interruption.

Important
This scenario might cause messages to be delivered out of order. If you're concerned about message ordering, follow the steps in Migrating to Amazon MQ with Service Interruption (p. 64).

On-Premises Message Broker	Migration to Amazon MQ with Standard (Unordered) Queues
	 Producers 3 6 1 Amazon MQ Broker AWS On-Premises Message Broker 2 4 5 Existing Consumers New Consumers

To migrate to Amazon MQ without service interruption

❶Create and configure an Amazon MQ broker (p. 12) and note your broker's endpoint, for example:

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

❷ For either of the following cases, use the Failover Transport to allow your consumers to randomly connect to your on-premises broker's endpoint or your Amazon MQ broker's endpoint. For example:

```
failover:(ssl://on-premises-broker.example.com:61617,ssl://
b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-east-2.amazonaws.com:61617)?randomize=true
```

Do one of the following:

- One by one, point each existing consumer to your Amazon MQ broker's endpoint.
- Create new consumers and point them to your Amazon MQ broker's endpoint.

  Note
  If you scale up your consumer fleet during the migration process, it is a best practice to scale it down afterward.

❸ One by one, stop each existing producer, point the producer to your Amazon MQ broker's endpoint, and then restart the producer.

❹ Wait for your consumers to drain the destinations on your on-premises broker.

❺ Change your consumers' Failover transport to include only your Amazon MQ broker's endpoint. For example:

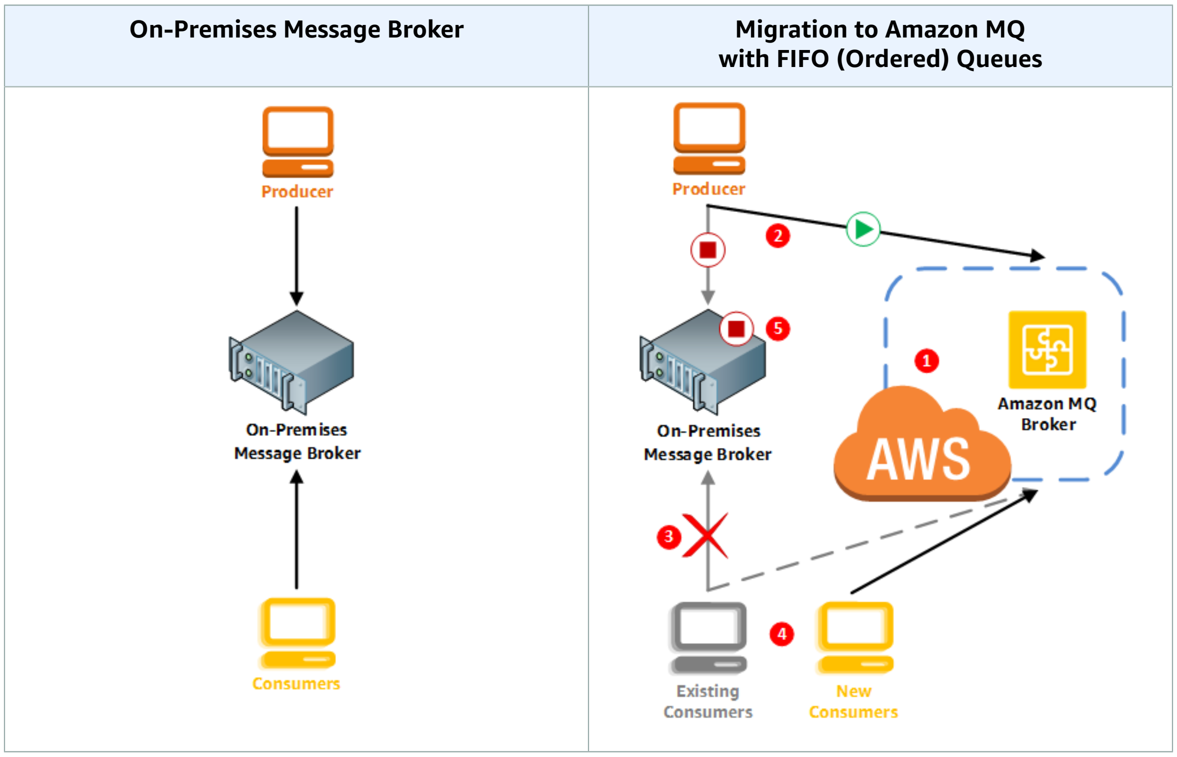
```
failover:(ssl://b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-east-2.amazonaws.com:61617)
```

❻ Stop your on-premises broker.

Migrating to Amazon MQ with Service Interruption

The following diagrams illustrate the scenario of migrating from an on-premises message broker to an Amazon MQ broker in the AWS Cloud with service interruption.

> **Important**
> This scenario requires you to point your producer to your Amazon MQ broker's endpoint *before* you do the same for your consumers. This sequence ensures that any messages in a FIFO (first-in-first-out) queue maintain their order during the migration process. If you're not concerned about message ordering, follow the steps in Migrating to Amazon MQ without Service Interruption (p. 63).


On-Premises Message Broker
Producer
On-Premises Message Broker
Consumers
Migration to Amazon MQ with FIFO (Ordered) Queues
Producer
2
5
1
3
4
On-Premises Message Broker
AWS
Amazon MQ Broker
Existing Consumers
New Consumers

To migrate to Amazon MQ with service interruption

❶ Create and configure an Amazon MQ broker (p. 12) and note your broker's endpoint, for example:

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

❷ Stop your existing producer, point the producer to your Amazon MQ broker's endpoint, and then restart the producer.

> **Important**
> This step requires an interruption of your application's functionality because no consumers are yet consuming messages from the Amazon MQ broker.

❸ Wait for your consumers to drain the destinations on your on-premises broker.

❹ Do one of the following:

- One by one, point each existing consumer to your Amazon MQ broker's endpoint.
- Create new consumers and point them to your Amazon MQ broker's endpoint.

> **Note**
> If you scale up your consumer fleet during the migration process, it is a best practice to scale it down afterward.

❺ Stop your on-premises broker.

Best Practices for Amazon MQ

Use these best practices to make the most of Amazon MQ.

Topics

Using Amazon MQ Securely

The following design patterns can improve the security of your Amazon MQ broker.

Topics

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 Accessing the ActiveMQ Web Console of a Broker without Public Accessibility (p. 15) in this guide and How to Help Prepare for DDoS Attacks by Reducing Your Attack Surface on the AWS Security Blog.

Always Use Client-Side Encryption as a Complement to TLS

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

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

Amazon MQ encrypts messages at rest and in transit using encryption keys that it manages and stores securely. For additional security, we highly recommend designing your application to use client-side encryption. For more information, see the *AWS Encryption SDK Developer Guide*.

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` (p. 54).

Always Configure a System Group

Amazon MQ uses a *system group* (called `activemq-webconsole`) to allow the ActiveMQ Web Console to communicate with the ActiveMQ broker.

The settings for the `activemq-webconsole` group in the authorization map restrict which operations can be performed on queues or topics from the web console. For more information and an example configuration, see `authorizationEntry` (p. 54).

> **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.

Connecting to Amazon MQ

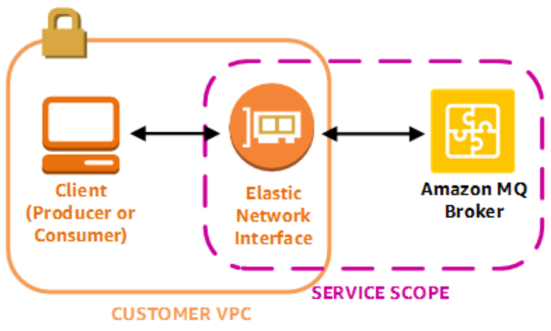
The following design patterns can improve the effectiveness of your application's connection to your Amazon MQ broker.

Topics

Never Modify or Delete the Amazon MQ Elastic Network Interface

When you first create an Amazon MQ broker (p. 12), Amazon MQ provisions an elastic network interface in the Virtual Private Cloud (VPC) under your account and, thus, requires a number of EC2 permissions (p. 85). 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.
> **Currently, you can't recover your broker if you delete its network interface. You can only recreate your broker.**



Always Use Connection Pooling

In a scenario with a single producer and single consumer (such as the Getting Started with Amazon MQ (p. 6) 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 username and password.
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 username and password.
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 broker (p. 12) 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-987615k4-32ji-109h-8gfe-7d65c4b132a1-2.mq.us-
east-2.amazonaws.com:61617)?randomize=true
```

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.

Ensuring Effective Amazon MQ Performance

The following design patterns can improve the effectiveness and performance of your Amazon MQ broker.

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` (p. 54).

Choose the Correct Broker Instance Type for the Best Throughput

The message throughput of a broker instance type (p. 33) 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. Because Amazon MQ has highly-durable storage (all persistent messages are replicated across three Availability Zones), the throughput to persistent storage is smaller than the throughput to local, single-AZ storage.

To determine the best broker instance type for your application, we recommend testing different broker instance types. For more information, see Broker Instance Types (p. 33) and also Measuring the Throughput for Amazon MQ using the JMS Benchmark.

> **Note**
> You can't change an existing broker to a different broker instance type. Using a different broker instance type requires creating a new broker (p. 12), modifying your application's configuration (p. 22) to use the new broker's wire-level protocol endpoint, and deleting the old broker. You must also drain all the messages from the old broker before using the new broker.

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 (p. 39) (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` (p. 54) 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.

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.

One cause of these unresolved transactions is an issue with Apache ActiveMQ. This may cause unresolved prepared transactions when Amazon MQ restarts. For more information, see the related Apache ActiveMQ defect.

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 Limits in Amazon MQ (p. 73).

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()`.

Limits 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 Limits in the *Amazon Web Services General Reference*.

Topics

Brokers

The following table lists limits 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 characters, dashes, periods, underscores, and tildes (– . _ ~).
Brokers per broker instance type (p. 33), per AWS account, per region	20
Broker configuration history depth	10
Connections per wire-level protocol	1,000
Security groups per broker	5
Destinations (queues and topics) monitored in CloudWatch	CloudWatch monitors only the first 200 destinations.

Configurations

The following table lists limits related to Amazon MQ configurations.

Limit	Description
Configuration name	• Must be 1-150 characters long. • Must contain only characters specified in the ASCII Printable Character Set.

Limit	Description
	• Can contain only alphanumeric characters, dashes, periods, underscores, and tildes (- . _ ~).
Configurations per AWS account	1,000
Revisions per configuration	300

Users

The following table lists limits related to Amazon MQ users (p. 37).

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. • Must not contain commas (,).
Users per broker	250
Groups per user	20

Data Storage

The following table lists limits related to Amazon MQ data storage.

Limit	Description
Storage capacity per broker	200 GB
Data store	Amazon MQ uses Apache KahaDB as its data store. Other data stores, such as JDBC and LevelDB, aren't supported.

API Throttling

The following throttling limits 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 limits don't apply to ActiveMQ broker messaging APIs. For example, Amazon MQ doesn't throttle the sending or receiving of messages.

Bucket Size	Refill Rate per Second
100	15

Monitoring and Logging Amazon MQ Brokers

This section provides information about monitoring and logging Amazon MQ broker activity.

Topics

Monitoring Amazon MQ Brokers Using Amazon CloudWatch

Amazon MQ and Amazon CloudWatch are integrated so you can use CloudWatch to view and analyze metrics for your ActiveMQ broker and the broker's destinations (queues and topics). You can view and analyze your Amazon MQ metrics from the CloudWatch console, the AWS CLI, or the CloudWatch CLI. CloudWatch metrics for Amazon MQ are automatically polled from the broker and then pushed to CloudWatch every minute.

For information about accessing Amazon MQ CloudWatch metrics, see Accessing CloudWatch Metrics for Amazon MQ (p. 29).

Note
The following statistics are valid for all of the metrics:

- `Average`
- `Minimum`
- `Maximum`
- `Sum`

The `AWS/AmazonMQ` namespace includes the following metrics.

Broker Metrics

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

Metric	Unit	Description
		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 EC2 compute units that the broker currently uses.
`CurrentConnectionsCount`	Count	The current number of active connections on the current broker.
`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.
`HeapUsage`	Percent	The percentage of the ActiveMQ JVM memory limit that the broker currently uses.
`NetworkIn`	Bytes	The volume of incoming traffic for the broker.
`NetworkOut`	Bytes	The volume of outgoing traffic for the broker.
`OpenTransactionsCount`	Count	The total number of transactions in progress.
`StorePercentUsage`	Percent	The percent used by the storage limit. If this reaches 100 the broker will refuse messages.
`TotalConsumerCount`	Count	The number of message consumers subscribed to destinations on the current broker.
`TotalMessageCount`	Count	The number of messages stored on the broker.

Metric	Unit	Description
`TotalProducerCount`	Count	The number of message producers active on destinations on the current broker.

Dimension for Broker 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.

Destination (Queue and Topic) Metrics

Important
The following metrics record only values since CloudWatch polled the metrics last:

- `EnqueueCount`
- `ExpiredCount`
- `DequeueCount`
- `DispatchCount`

Metric	Unit	Description
`ConsumerCount`	Count	The number of consumers subscribed to the destination.
`EnqueueCount`	Count	The number of messages sent to the destination.
`EnqueueTime`	Time (milliseconds)	The amount of time it takes the broker to accept a message from the producer and send it to the destination.
`ExpiredCount`	Count	The number of messages that couldn't be delivered because they expired.
`DispatchCount`	Count	The number of messages sent to consumers.
`DequeueCount`	Count	The number of messages acknowledged by consumers.

Metric	Unit	Description
`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.

Dimensions for 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.

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 (p. 82).

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`

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 IAM 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": "5.15.6",
        "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 to Publish General and Audit Logs to Amazon CloudWatch Logs

Amazon MQ is integrated with Amazon CloudWatch Logs, a service that monitors, stores, and accesses your log files from a variety of sources. For example, you can configure CloudWatch alarms to receive notifications of broker reboots or troubleshoot broker configuration (p. 41) errors. For more information about CloudWatch Logs, see the *Amazon CloudWatch Logs User Guide*.

To allow Amazon MQ to publish logs to CloudWatch Logs, you must add a permission to your Amazon MQ user (p. 83) and also configure a resource-based policy for Amazon MQ (p. 83) before you create or restart the broker.

For more information about configuring Amazon MQ to publish general and audit logs to CloudWatch Logs, see Configure Advanced Broker Settings (p. 13).

Topics

Understanding the Structure of Logging in CloudWatch Logs

You can enable *general* and *audit* logging when you configure advanced broker settings (p. 13) 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 (p. 38) or an active/standby broker (p. 39), 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` to user 111122223333.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Principal": {
            "AWS": "111122223333"
         },
         "Action": "logs:CreateLogGroup",
         "Resource": "arn:aws:logs:*:*:log-group:/aws/amazonmq/*"
      }
   ]
}
```

For more information, see `CreateLogGroup` in the *Amazon CloudWatch Logs API Reference*.

Configure a Resource-Based Policy for Amazon MQ

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.

Important
If you don't configure a resource-based policy for Amazon MQ, the broker can't publish the logs to CloudWatch Logs.

The following example resource-based policy grants permission for `logs:CreateLogStream` and `logs:PutLogEvents` to AWS.

```
{
  "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.

You can achieve the same effect using the following AWS CLI command:

```
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/*" } ] }'
```

Troubleshooting CloudWatch Logs Configuration

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 (p. 83) 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 (p. 83). This allows your broker to publish its logs.

Amazon MQ Security

This section provides information about Amazon MQ and ActiveMQ authentication and authorization. For information about security best practices, see Using Amazon MQ Securely (p. 66).

Topics

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.

```
{
    "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:DescribeNetworkInterfacePermission"
        ],
        "Effect": "Allow",
        "Resource": "*",
        "Condition": {
            "StringEquals": {
                "ec2:AuthorizedService": "mq.amazonaws.com"
            }
        }
    }]
}
```

For more information, see Create an IAM User and Get Your AWS Credentials (p. 4) and Never Modify or Delete the Amazon MQ Elastic Network Interface (p. 67).

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`
`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`

Amazon MQ REST APIs	Required Permissions
`ListConfigurations`	`mq:ListConfigurations`
`ListUsers`	`mq:ListUsers`
`RebootBroker`	`mq:RebootBroker`
`UpdateBroker`	`mq:UpdateBroker`
`UpdateConfiguration`	`mq:UpdateConfiguration`
`UpdateUser`	`mq:UpdateUser`

Messaging Authentication and Authorization for ActiveMQ

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

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

Amazon MQ uses native ActiveMQ authentication to manage user permissions. For information about restrictions related to ActiveMQ usernames and passwords, see Limits Related to Users (p. 74).

To authorize ActiveMQ users and groups to works with queues and topics, you must edit your broker's configuration (p. 19). 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 (p. 67) and `authorizationEntry` (p. 54).

Note
Currently, Amazon MQ doesn't support Client Certificate Authentication or plugins for Java Authentication and Authorization Service (JAAS).

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::Amazon MQ::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::Amazon MQ::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.
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

Apache 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.

Resource	Description
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 Release Notes

The following table lists Amazon MQ feature releases and improvements. For changes to the *Amazon MQ Developer Guide*, see Amazon MQ Document History (p. 93).

Date	Feature Release
October 15, 2018	• The maximum number of groups per user is 20. For more information, see Limits Related to Users (p. 74). • The maximum number of connections per broker, per wire-level protocol is 1,000. For more information, see Limits Related to Brokers (p. 73).
October 2, 2018	AWS has expanded its HIPAA compliance program to include Amazon MQ as a HIPAA Eligible Service.
September 27, 2018	Amazon MQ supports ActiveMQ 5.15.6, in addition to 5.15.0. For more information, see the following: • Editing Broker Engine Version, CloudWatch Logs, and Maintenance Preferences (p. 16) • 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 Broker Metrics (p. 76) 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.
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)

Date	Feature Release
	• EU (Frankfurt) • EU (Ireland)
July 30, 2018	You can configure Amazon MQ to publish general and audit logs to Amazon CloudWatch Logs. For more information, see Configuring Amazon MQ to Publish Logs to Amazon CloudWatch Logs (p. 82).
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) • EU (Frankfurt) • EU (Ireland)
July 19, 2018	You can use AWS CloudTrail to log Amazon MQ API calls. For more information, see Logging Amazon MQ API Calls Using CloudTrail (p. 79).
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 (p. 33).
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) • EU (Frankfurt) • EU (Ireland)

Date	Feature Release
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 AWS 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 Limits Related to Users (p. 74).
March 13, 2018	Creating a broker takes about 15 minutes. For more information, see Finish creating the broker (p. 14).
March 1, 2018	• You can configure the concurrent store and dispatch (p. 69) for Apache KahaDB using the `concurrentStoreAndDispatchQueues` (p. 54) attribute. • The `CpuCreditBalance` CloudWatch metric (p. 76) 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 Enable Inbound Connections (p. 7) 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.

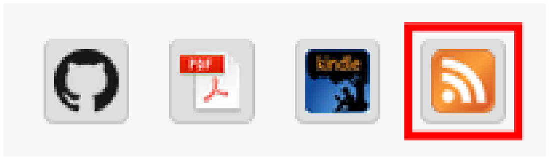
Date	Feature Release
November 28, 2017	This is the initial release of Amazon MQ and the *Amazon MQ Developer Guide*. • Amazon MQ is avaialble in the following regions: • US East (Ohio) • US East (N. Virginia) • US West (Oregon) • Asia Pacific (Sydney) • EU (Frankfurt) • EU (Ireland) Using the `mq.t2.micro` 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` (p. 76) 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` (p. 76) 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 • 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 (p. 22). • Your broker can send and receive messages of any size.

Amazon MQ Document History

The following table lists changes to the *Amazon MQ Developer Guide*. For Amazon MQ feature releases and improvements, see Amazon MQ Release Notes (p. 90).

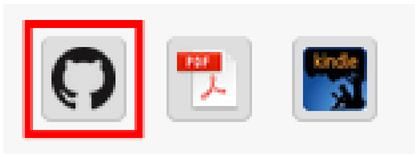
Date	Documentation Update
October 26, 2018	Added a new Best Practices topic. See: Avoid Slow Restarts by Recovering Prepared XA Transactions (p. 71).
October 15, 2018	Updated the Limits in Amazon MQ (p. 73) section.
October 8, 2018	Updated the Frequently Viewed Amazon MQ Topics (p. 3) section.

Date	Documentation Update
October 3, 2018	Corrected outdated links in the Setting Up Amazon MQ (p. 4) and Amazon MQ Tutorials (p. 12) sections.
October 1, 2018	Corrected the information in the Next Steps (p. 10) section.
September 27, 2018	• Added the Editing Broker Engine Version, CloudWatch Logs, and Maintenance Preferences (p. 16) section. • Updated the following sections: • Broker Engine (p. 37) • Create an ActiveMQ Broker (p. 6) • Configure Basic Broker Settings (p. 12)
September 18, 2018	Added the following note to the Creating and Managing Amazon MQ Broker Users (p. 26) section: 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.
September 10, 2018	Updated the Frequently Viewed Amazon MQ Topics (p. 3) section.
August 31, 2018	• Clarified the terminology for active/standby brokers. For more information, see Amazon MQ Active/Standby Broker for High Availability (p. 39). • Simplified the terminology for the maintenance window. For more information, see Amazon MQ Broker Configuration Lifecycle (p. 40). • Rewrote the Configure Advanced Broker Settings (p. 13) section. • Updated the Broker Metrics (p. 76) and Listing Brokers and Viewing Broker Details (p. 25) sections.
August 15, 2018	Corrected the information in the Create an ActiveMQ Broker (p. 6) section.
August 13, 2018	Added the Accessing the ActiveMQ Web Console of a Broker without Public Accessibility (p. 15) section.
August 3, 2018	Updated the Frequently Viewed Amazon MQ Topics (p. 3) section.
August 2, 2018	• Added the Troubleshooting CloudWatch Logs Configuration (p. 84) section. • Added the following admonition throughout this guide: **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.
August 1, 2018	Corrected the information in the following sections: • Understanding the Structure of Logging in CloudWatch Logs (p. 82) • Connect a Java Application to Your Broker (p. 7)
July 31, 2018	• Moved the 3-minute demo video to the Getting Started with Amazon MQ (p. 6) section. • Added the 3-minute getting started video to the What is Amazon MQ? (p. 1) section.

Date	Documentation Update
July 30, 2018	• Added the Configuring Amazon MQ to Publish Logs to Amazon CloudWatch Logs (p. 82) section. • Updated the Configure Advanced Broker Settings (p. 13) section.
July 19, 2018	• Added the Logging Amazon MQ API Calls Using CloudTrail (p. 79) section. • Corrected the information in the What Are the Main Benefits of Amazon MQ? (p. 1) section.
July 5, 2018	• Added an `authorizationEntry` child element cross-reference to the Always Configure an Authorization Map (p. 67) section. • Clarified the information in the Messaging Authentication and Authorization for ActiveMQ (p. 87) section. • Clarified the information in the Limits Related to API Throttling (p. 74) section. • Updated the Frequently Viewed Amazon MQ Topics (p. 3) section.
June 29, 2018	• Updated the information in the Broker Instance Types (p. 33) section. • Added the Choose the Correct Broker Instance Type for the Best Throughput (p. 70) section.
June 26, 2018	Added a link to a related resource to the Migrating to Amazon MQ (p. 63) section.
June 8, 2018	Updated the Frequently Viewed Amazon MQ Topics (p. 3) section.
June 4, 2018	In addition to GitHub, HTML, PDF, and Kindle, the *Amazon MQ Developer Guide* release notes are available as an RSS feed. 
May 29, 2018	Made the following changes in the Working Java Example (p. 55) section: • Added a STOMP+WSS Java example. The STOMP+WSS example Java code connects to a broker, creates a queue, and publishes and receives a message. • Improved the MQTT Java example. • Improved the OpenWire Java example.
May 24, 2018	Corrected the wire-level protocol endpoint port in the MQTT Java example in the Working Java Example (p. 55) section.
May 22, 2018	Corrected the information in all Java dependency sections.
May 17, 2018	Corrected the information in the Limits Related to Users (p. 74) section.
May 15, 2018	Corrected the information in the Ensuring Effective Amazon MQ Performance (p. 69) section.
May 8, 2018	• Placed the Amazon MQ REST API Permissions Reference (p. 86) in its own section. • Created the IAM Permissions Required to Create an Amazon MQ Broker (p. 85) section with an example custom IAM policy.

Date	Documentation Update
May 7, 2018	• Clarified throughout this guide that the broker maintenance window is 2 hours long. For more information, see Amazon MQ Broker Configuration Lifecycle (p. 40). • Added explanations for why the `ec2:CreateNetworkInterface` and `ec2:CreateNetworkInterfacePermission` permissions are necessary for creating a broker. For more information, see API Authentication and Authorization for Amazon MQ (p. 85).
May 4, 2018	Updated the Frequently Viewed Amazon MQ Topics (p. 3) section.
May 1, 2018	Clarified the information about the maintenance window for active/standby brokers in the following sections: • Amazon MQ Active/Standby Broker for High Availability (p. 39) • Creating and Configuring a Broker (p. 12) • Creating and Applying Broker Configurations (p. 17) • Editing and Managing Broker Configurations (p. 19)
April 27, 2018	Rewrote the following sections and optimized example Java code to match the recommendation to use connection pooling only for producers, *not* consumers: • Always Use Connection Pooling (p. 68) • Create a Message Producer and Send a Message (p. 8) • Create a Message Consumer and Receive the Message (p. 9) • AmazonMQExample.java (p. 57)
April 26, 2018	Added an MQTT Java example to the Working Java Example (p. 55) section. The MQTT example Java code connects to a broker, creates a topic, and publishes and receives a message.
April 6, 2018	Updated the Frequently Viewed Amazon MQ Topics (p. 3) section.
April 4, 2018	Renamed the Communicating with Amazon MQ section to Connecting to Amazon MQ (p. 67).
April 3, 2018	Clarified and corrected the information in the the Disable Concurrent Store and Dispatch for Queues with Slow Consumers (p. 69) section.
April 2, 2018	Moved the Concurrent Store and Dispatch for Queues in Amazon MQ section to the Disable Concurrent Store and Dispatch for Queues with Slow Consumers (p. 69) section.
March 27, 2018	• Replaced the re:Invent launch video with a 3-minute demo video in the What is Amazon MQ? (p. 1) section. • Rewrote the What Are the Main Benefits of Amazon MQ? (p. 1) section to include information about recently released features. • Restructured the following sections: • Amazon MQ Broker Architecture (p. 38) • How Amazon MQ Works (p. 32) • Migrating to Amazon MQ (p. 63) • Moved Amazon MQ Broker Configuration Lifecycle (p. 40) under the Amazon MQ Broker Architecture (p. 38) section.

Date	Documentation Update
March 22, 2018	Clarified the following statement throughout this guide: Amazon MQ encrypts messages at rest and in transit using encryption keys that it manages and stores securely. For additional security, we highly recommend designing your application to use client-side encryption. For more information, see the *AWS Encryption SDK Developer Guide*.
March 19, 2018	Clarified the following statement throughout this guide: 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 a shared storage location.
March 15, 2018	• Restructured the Amazon MQ Basic Elements (p. 32) section. • Improved the explanation of the diagrams in the following sections: • Migrating to Amazon MQ without Service Interruption (p. 63) • Migrating to Amazon MQ with Service Interruption (p. 64)
March 12, 2018	• Clarified and corrected the information in the Using Amazon MQ Securely (p. 66) and Connecting to Amazon MQ (p. 67) sections. • Added the Disable Concurrent Store and Dispatch for Queues with Slow Consumers (p. 69) section. • Grouped admonitions into a preface for the Configure advanced broker settings (p. 13) section.
March 9, 2018	• Clarified and corrected the information in the Always Configure an Authorization Map (p. 67) and Always Configure a System Group (p. 67) sections. • Added the authorizationEntry (p. 54) section and updated the kahaDB (p. 54) section.
March 8, 2018	• Added the Always Configure an Authorization Map (p. 67) and Always Configure a System Group (p. 67) sections. • Added notes about broker suffixes to the Monitoring Amazon MQ Using CloudWatch (p. 76) section.
March 7, 2018	Updated the Frequently Viewed Amazon MQ Topics (p. 3) section.
March 6, 2018	Added the following note throughout this guide: **Note** 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` (p. 76) metric). If your application requires *fixed performance*, consider using an `mq.m5.large` instance type.
March 1, 2018	• Added the `CpuCreditBalance` metric to the Broker Metrics (p. 76) section. • Added the Amazon MQ Child Element Attributes (p. 54) section. • Added links from elements in the the section called "Permitted Elements" (p. 42) section to their attributes and to child collection elements. • Made corrections to the AWS Glossary in GitHub.
February 28, 2018	Corrected image display in GitHub.

Date	Documentation Update
February 27, 2018	In addition to HTML, PDF, and Kindle, the *Amazon MQ Developer Guide* is available on GitHub. To leave feedback, choose the GitHub icon in the upper right-hand corner. 
February 26, 2018	• Made regions consistent in all examples and diagrams. • Optimized links to the AWS console and product webpages.
February 22, 2018	Clarified and corrected the information in the following sections: • Prefer Brokers without Public Accessibility (p. 66) • Always Use Client-Side Encryption as a Complement to TLS (p. 66) • Always Use the Failover Transport to Connect to Multiple Broker Endpoints (p. 69) • API Authentication and Authorization for Amazon MQ (p. 85) • Messaging Authentication and Authorization for ActiveMQ (p. 87)
February 21, 2018	Corrected the Java code in the following sections: • Working Java Example (p. 55) • Connect a Java Application to Your Broker (p. 7) • Always Use Connection Pooling (p. 68)
February 20, 2018	Clarified and corrected the information in the Amazon MQ Security (p. 85) and Best Practices for Amazon MQ (p. 66) sections.
February 19, 2018	• Corrected the Java code in the Always Use Connection Pooling (p. 68) section. • Clarified and corrected the information in the Always Use Client-Side Encryption as a Complement to TLS (p. 66) section. • Restructured and expanded the Best Practices for Amazon MQ (p. 66) and Amazon MQ Security (p. 85) sections.
February 16, 2018	• Added the Using Amazon MQ Securely (p. 66) section. • Updated the Connecting to Amazon MQ (p. 67) section. • Corrected the Java code in the following sections: • Getting Started with Amazon MQ (p. 6) • AmazonMQExample.java (p. 57)
February 15, 2018	• Restructured and expanded the Best Practices for Amazon MQ (p. 66) section. • Updated the following sections: • How Can I Get Started with Amazon MQ? (p. 2) • Next Steps (p. 10) (Getting Started) • Related Resources (p. 88)

Date	Documentation Update
February 14, 2018	Updated the following sections: • Limits in Amazon MQ (p. 73) • Limits Related to API Throttling (p. 74) • Best Practices for Amazon MQ (p. 66) • Amazon MQ Security (p. 85)
February 13, 2018	• Updated the Related Resources (p. 88) section. • Updated the Limits in Amazon MQ (p. 73) section. • Added the We Want to Hear from You (p. 2) section.
February 2, 2018	Created the Frequently Viewed Amazon MQ Topics (p. 3) section.
January 25, 2018	• Fixed an error in the Add Java Dependencies (p. 55) subsection of the Working Java Example (p. 55) section. • The permission for REST operation ID `RebootBroker` is listed correctly as `mq:RebootBroker` on the IAM console.
January 24, 2018	• Added the Never Modify or Delete the Amazon MQ Elastic Network Interface (p. 67) section. • Updated all diagrams throughout this guide. • Added links to the *Amazon MQ REST API Reference* throughout this guide and links to specific REST APIs to the API Authentication and Authorization for Amazon MQ (p. 85) section.
January 19, 2018	Updated the information in the Apache ActiveMQ Resources (p. 88) section.
January 18, 2018	Clarified and corrected the information in the Limits in Amazon MQ (p. 73) section.
January 17, 2018	Reinstated the recommendation to prefer virtual destinations over durable subscriptions (p. 69), with an improved explanation.
January 11, 2018	• The *Amazon MQ Developer Guide* is available in Kindle format, in addition to HTML and PDF. • Clarified and corrected information in the API Authentication and Authorization for Amazon MQ (p. 85) and Create an IAM User and Get Your AWS Credentials (p. 4) sections.
January 3, 2018	Added `DescribeConfigurationRevision` to the API Authentication and Authorization for Amazon MQ (p. 85) section.
December 15, 2017	Removed the recommendation against durable subscriptions from the Best Practices for Amazon MQ (p. 66) section.
December 8, 2017	• Added the Enable Inbound Connections (p. 7) prerequisite to the Connecting a Java Application to Your Broker (p. 22) and Working Java Example (p. 55) sections. • Added the following note throughout this guide: Currently, it isn't possible to delete a configuration.
December 7, 2017	• Improved the code in the AmazonMQExample.java (p. 57). • Added the API Authentication and Authorization for Amazon MQ (p. 85) section.

Date	Documentation Update
December 5, 2017	• Clarified and corrected information in the Monitoring Amazon MQ Using CloudWatch (p. 76) section: • Improved the metric descriptions. • Added the Dimension for Broker Metrics (p. 78) and Dimensions for Destination (Queue and Topic) Metrics (p. 79) sub-sections. • Added the "Introducing Amazon MQ" video to the What is Amazon MQ? (p. 1) section.
December 4, 2017	• Clarified the following information in the Limits Related to Data Storage (p. 74) section: Storage capacity *per broker* is 200 GB. • Added the Prerequisites (p. 55) to the Working Java Example (p. 55) section. (The `activemq-client.jar` and `activemq-pool.jar` packages are required for the example to work. For more information, see Connecting a Java Application to Your Broker (p. 22)).
December 1, 2017	• Updated and improved the screenshots in all the tutorials. • Clarified the following explanation throughout this guide: Making changes to a configuration revision or an ActiveMQ user does *not* apply the changes immediately. To apply your changes, you must wait for the next maintenance window (p. 21) or reboot the broker (p. 28). For more information, see Amazon MQ Broker Configuration Lifecycle (p. 40).

AWS Glossary

For the latest AWS terminology, see the AWS Glossary in the *AWS General Reference*.