# Table of Contents

**What is Amazon MQ?** .................................................................................................................................. 1  
**How is Amazon MQ different from Amazon SQS or Amazon SNS?** .......................................................... 1  
**How can I get started with Amazon MQ?** .............................................................................................. 1  
**We want to hear from you** ...................................................................................................................... 1  
**Setting up** .............................................................................................................................................. 2  
  - Create an AWS account and an IAM administrator user ....................................................................... 2  
  - Create an IAM user and get your AWS credentials ................................................................................ 2  
  - Step 3: get ready to use the example codes .......................................................................................... 3  
  - Next steps ............................................................................................................................................ 3  
**Getting started** ........................................................................................................................................ 4  
  - Prerequisites ....................................................................................................................................... 4  
  - Creating and connecting to an ActiveMQ broker .................................................................................. 4  
    - Step 1: create an ActiveMQ broker ....................................................................................................... 4  
    - Step 2: connect a Java application to your broker .............................................................................. 6  
    - Step 3: (Optional) connect to an AWS Lambda function .................................................................... 9  
    - Step 4: delete your broker .................................................................................................................. 11  
    - Next steps ........................................................................................................................................ 11  
  - Creating and connecting to a RabbitMQ broker .................................................................................... 11  
    - Step 1: create a RabbitMQ broker ....................................................................................................... 11  
    - Step 2: connect a JVM-based application to your broker .................................................................. 13  
    - Step 3: (Optional) connect to an AWS Lambda function .................................................................. 16  
    - Step 4: delete your broker .................................................................................................................. 18  
    - Next steps ........................................................................................................................................ 18  
**Managing a broker** ............................................................................................................................... 19  
  - Listing brokers and viewing broker details ............................................................................................ 19  
  - To list brokers and view broker details .................................................................................................. 19  
  - Maintaining a broker .............................................................................................................................. 21  
    - Adjusting the broker maintenance window .......................................................................................... 22  
  - Upgrading the engine version ................................................................................................................ 24  
    - Manually upgrading the engine version .............................................................................................. 24  
    - Automatically upgrading the minor engine version ........................................................................... 26  
  - Rebooting a broker ............................................................................................................................... 27  
    - To Reboot an Amazon MQ Broker ..................................................................................................... 27  
  - Deleting a broker ................................................................................................................................... 28  
    - Deleting an Amazon MQ broker .......................................................................................................... 28  
**Tutorials** .................................................................................................................................................. 29  
  - Accessing the broker web console without public accessibility ............................................................ 29  
    - Prerequisites ...................................................................................................................................... 29  
    - To Access a broker's web console of a Broker without Public Accessibility ...................................... 29  
  - ActiveMQ tutorials ................................................................................................................................ 30  
    - Creating and configuring a broker ....................................................................................................... 30  
    - Editing broker preferences .................................................................................................................. 34  
    - Creating and configuring a network of brokers .................................................................................... 35  
    - Creating and applying configurations .................................................................................................. 38  
    - Editing configurations and managing configuration revisions ......................................................... 40  
    - Connecting a Java application to your broker ...................................................................................... 43  
    - Integrating ActiveMQ brokers with LDAP ........................................................................................... 47  
    - Creating and managing broker users .................................................................................................... 58  
  - RabbitMQ tutorials ............................................................................................................................... 60  
    - Editing broker preferences .................................................................................................................. 60  
    - Using Python Pika with Amazon MQ for RabbitMQ ............................................................................. 60  
    - Resolving paused queue sync ............................................................................................................. 65  
**Best practices** .......................................................................................................................................... 69  
**Amazon MQ for ActiveMQ best practices** ............................................................................................. 69
What is Amazon MQ?

Amazon MQ is a managed message broker service 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. Currently, Amazon MQ supports Apache ActiveMQ and RabbitMQ engine types.

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

Topics
- How is Amazon MQ different from Amazon SQS or Amazon SNS? (p. 1)
- How can I get started with Amazon MQ? (p. 1)
- We want to hear from you (p. 1)

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 0-9-1, AMQP 1.0, 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. 4).
- To discover the functionality and architecture of Amazon MQ, see How Amazon MQ Works (p. 79).
- To find out the guidelines and caveats that will help you make the most of Amazon MQ, see Best Practices for Amazon MQ (p. 69).
- 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.
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 (p. 2)
• Step 2: create an IAM user and get your AWS credentials (p. 2)
• Step 3: get ready to use the example codes (p. 3)
• Next steps (p. 3)

Step 1: create an AWS account and an IAM administrator user

To access any AWS service, you must first create an Amazon Web Services 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 Amazon Web Services 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:

   **wJalrXUtFbEMI/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 codes

The following tutorials show how you can work with Amazon MQ brokers using the AWS Management Console as well as how to connect to your Amazon MQ for ActiveMQ and Amazon MQ for RabbitMQ brokers programmatically. To use the ActiveMQ Java example code, you must install the **Java Standard Edition Development Kit** and make some changes to the code.

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

## Next steps

Now that you're prepared to work with Amazon MQ, get started by creating a broker (p. 4). Depending on your broker engine type, you can then connect a Java application to your Amazon MQ for ActiveMQ broker (p. 6) or use the RabbitMQ Java client library to connect a JVM-based application to your Amazon MQ for RabbitMQ broker (p. 13).
Getting started with Amazon MQ

This section will help you become more familiar with Amazon MQ by showing you how to create an Amazon MQ for ActiveMQ or RabbitMQ broker and how to connect your application to it.

Creating and connecting to a broker instance is slightly different for each broker engine. Choose one of the following engine types that you want to use for detailed information on creating and connecting to a broker. After you have created and connected to your broker, you can find instructions to help you delete it.

Topics
- Prerequisites (p. 4)
- Creating and connecting to an ActiveMQ broker (p. 4)
- Creating and connecting to a RabbitMQ broker (p. 11)

Prerequisites

Before you begin, complete the steps in Setting Up Amazon MQ (p. 2).

Creating and connecting to an ActiveMQ broker

A broker is a message broker environment running on Amazon MQ. It is the basic building block of Amazon MQ. The combined description of the broker instance class (m5, t3) and size (large, micro) is a broker instance type (for example, mq.m5.large). For more information, see Broker (p. 79).

Topics
- Step 1: create an ActiveMQ broker (p. 4)
- Step 2: connect a Java application to your broker (p. 6)
- Step 3: (Optional) connect to an AWS Lambda function (p. 9)
- Step 4: delete your broker (p. 11)
- Next steps (p. 11)

Step 1: create an ActiveMQ broker

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

1. Sign in to the Amazon MQ console.
2. On the Select broker engine page, choose Apache ActiveMQ.
3. On the Select deployment and storage page, in the Deployment mode and storage type section, do the following:
   a. Choose the Deployment mode (for example, Active/standby broker). For more information, see Broker Architecture (p. 83).
Step 1: create an ActiveMQ broker

• A **Single-instance broker** is comprised of one broker in one Availability Zone. The broker communicates with your application and with an Amazon EBS or Amazon EFS storage volume. For more information, see Amazon MQ single-instance broker (p. 84).

• An **Active/standby broker for high availability** is comprised of two brokers in two different Availability Zones, configured in a *redundant pair*. These brokers communicate synchronously with your application, and with Amazon EFS. For more information, see Amazon MQ active/standby broker for high availability (p. 85).

• For more information on the sample blueprints for a network of brokers, see Sample blueprints (p. 88).

b. Choose the **Storage type** (for example, **EBS**). For more information, see Storage (p. 82).

   **Note**
   Amazon EBS replicates data within a single Availability Zone and doesn't support the ActiveMQ active/standby (p. 85) deployment mode.

c. Choose **Next**.

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

a. Enter the **Broker name**.

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

b. Choose the **Broker instance type** (for example, **mq.m5.large**). For more information, see Broker instance types (p. 136).

5. In the **ActiveMQ Web Console access** section, provide a **Username** and **Password**. The following restrictions apply to broker usernames and passwords:
   
   • Your username can contain only alphanumeric characters, dashes, periods, underscores, and tildas (- . _ ~).
   
   • Your password must be at least 12 characters long, contain at least 4 unique characters and must not contain commas, colons, or equal signs (,:=).

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

6. Choose **Deploy**.

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

   Creating the broker takes about 15 minutes.

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

   ![Broker status table]

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

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

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

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

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

Step 2: connect a Java application to your broker

After you create an Amazon MQ ActiveMQ broker, you can connect your application to it. The following examples show how you can use the Java Message Service (JMS) to create a connection to the broker, create a queue, and send a message. For a complete, working Java example, see Working Java Example (p. 116).

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 web console URL and wire-level protocols.
4. In the Details section, under Security and network, choose the name of your security group or [ ].

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

   a. Choose Add Rule.
   b. For Type, select Custom TCP.
   c. For Port Range, type the web console port (8162).
   d. For Source, leave Custom selected and then type the IP address of the system that you want to be able to access the web console (for example, 192.0.2.1).
   e. Choose Save.

Your broker can now accept inbound connections.
Add Java dependencies

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

```xml
<dependencies>
  <dependency>
    <groupId>org.apache.activemq</groupId>
    <artifactId>activemq-client</artifactId>
    <version>5.15.8</version>
  </dependency>
  <dependency>
    <groupId>org.apache.activemq</groupId>
    <artifactId>activemq-pool</artifactId>
    <version>5.15.8</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 Broker Architecture (p. 83).
   
   For wire-level protocol endpoints, you can allow your application to connect to either endpoint by using the Failover Transport.

   ```java
   // 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. 70).

2. Create a session, a queue named MyQueue, and a message producer.
Step 2: connect a Java application to your broker

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

```java
// 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.

```java
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.

```java
// 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. 70).

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

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

```java
// 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.

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

### Step 3: (Optional) connect to an AWS Lambda function

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

**To connect your broker to a Lambda function**

1. Add the following IAM role permissions to your Lambda function execution role.
   - `mq:DescribeBroker`
   - `ec2:CreateNetworkInterface`
   - `ec2:DeleteNetworkInterface`
   - `ec2:DescribeNetworkInterfaces`
   - `ec2:DescribeSecurityGroups`
   - `ec2:DescribeSubnets`
   - `ec2:DescribeVpcs`
   - `logs:CreateLogGroup`
   - `logs:CreateLogStream`
   - `logs:PutLogEvents`
   - `secretsmanager:GetSecretValue`

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

2. (Optional) If you have created a broker without public accessibility, you must do one of the following to allow Lambda to connect to your broker:
Step 3: (Optional) connect to an AWS Lambda function

- Configure one NAT gateway per public subnet. For more information, see Internet and service access for VPC-connected functions in the AWS Lambda Developer Guide.

- Create a connection between your Amazon Virtual Private Cloud (Amazon VPC) and Lambda using a VPC endpoint. Your Amazon VPC must also connect to AWS Security Token Service (AWS STS) and Secrets Manager endpoints. For more information, see Configuring interface VPC endpoints for Lambda in the AWS Lambda Developer Guide.

3. Configure your broker as an event source for a Lambda function using the AWS Management Console. You can also use the create-event-source-mapping AWS Command Line Interface command.

4. Write some code for your Lambda function to process the messages consumed from your broker. The Lambda payload that retrieved by your event source mapping depends on the engine type of the broker. The following is an example of a Lambda payload for an Amazon MQ for ActiveMQ queue.

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

   ```json
   {  
     "eventSource": "aws:amq",
     "messages": [  
       {  
         "messageType": "jms/text-message",
         "data": "QUJDOkFBQUE==",
         "connectionId": "myJMSCoID",
         "redelivered": false,
         "destination": {  
           "physicalname": "testQueue"
         },
         "timestamp": 1598827811958,
         "brokerInTime": 1598827811958,
         "brokerOutTime": 1598827811959
       },
       {  
         "messageType": "jms/bytes-message",
         "data": "3DTOOW7crj51prgVLQaGQ82S48k=",
         "connectionId": "myJMSCoID1",
         "persistent": false,
         "destination": {  
           "physicalname": "testQueue"
         },
         "timestamp": 1598827811958,
         "brokerInTime": 1598827811958,
         "brokerOutTime": 1598827811959
       }
     ]
   }
   ```

For more information about connecting Amazon MQ to Lambda, the options Lambda supports for an Amazon MQ event source, and event source mapping errors, see Using Lambda with Amazon MQ in the AWS Lambda Developer Guide.
Step 4: 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. 30) (Additional Settings)
- Editing broker engine version, Amazon CloudWatch Logs, and maintenance preferences (p. 34)
- Creating and applying broker configurations (p. 38)
- Editing and Managing Broker Configurations (p. 40)
- Listing brokers and viewing broker details (p. 19)
- Creating and managing ActiveMQ broker users (p. 58)
- Rebooting a Broker (p. 27)
- Accessing CloudWatch metrics for Amazon MQ (p. 168)

You can also begin to dive deep into Amazon MQ best practices (p. 69) and Amazon MQ REST APIs, and then plan to migrate to Amazon MQ.

Creating and connecting to a RabbitMQ broker

A broker is a message broker environment running on Amazon MQ. It is the basic building block of Amazon MQ. The combined description of the broker instance class (m5, t3) and size (large, micro) is a broker instance type (for example, mq.m5.large).

Topics

- Step 1: create a RabbitMQ broker (p. 11)
- Step 2: connect a JVM-based application to your broker (p. 13)
- Step 3: (Optional) connect to an AWS Lambda function (p. 16)
- Step 4: delete your broker (p. 18)
- Next steps (p. 18)

Step 1: create a RabbitMQ broker

The first and most common Amazon MQ task is creating a broker. The following example shows how you can use the AWS Management Console to create a basic broker.
1. Sign in to the Amazon MQ console.
2. On the Select broker engine page, choose RabbitMQ, and then choose Next.
3. On the Select deployment mode page, choose the Deployment mode, for example, Cluster deployment, and then choose Next.
   - A single-instance broker is comprised of one broker in one Availability Zone behind a Network Load Balancer (NLB). The broker communicates with your application and with an Amazon EBS storage volume. For more information, see Single-instance broker (p. 133).
   - A RabbitMQ cluster deployment for high availability is a logical grouping of three RabbitMQ broker nodes behind a Network Load Balancer, each sharing users, queues, and a distributed state across multiple Availability Zones (AZ). For more information, see Cluster deployment for high availability (p. 133).
4. On the Configure settings page, in the Details section, the following:
   a. Enter the Broker name.
      **Important**
      Do not add personally identifiable information (PII) or other confidential or sensitive information in broker names. Broker names are accessible to other AWS services, including CloudWatch Logs. Broker names are not intended to be used for private or sensitive data.
   b. Choose the Broker instance type (for example, mq.m5.large). For more information, see Broker instance types (p. 136).
      **Note**
      The Additional settings section provides options to enable CloudWatch logs and configure network access for your broker. If you create a private RabbitMQ broker without public accessibility, you must select a Virtual Private Cloud (VPC) and configure a security group to access your broker.
5. On the Configure settings page, in the RabbitMQ access section, provide a Username and Password. The following restrictions apply to broker usernames and passwords:
   - Your username can contain only alphanumeric characters, dashes, periods, and underscores (- . _). This value must not contain any tilde (~) characters. Amazon MQ prohibits using guest as a username.
   - Your password must be at least 12 characters long, contain at least 4 unique characters and must not contain commas, colons, or equal signs (,:=).
      **Important**
      Do not add personally identifiable information (PII) or other confidential or sensitive information in broker usernames. Broker usernames are accessible to other AWS services, including CloudWatch Logs. Broker usernames are not intended to be used for private or sensitive data.
6. Choose Next.
7. On the Review and create page, you can review your selections and edit them as needed.
8. Choose Create broker.
   While Amazon MQ creates your broker, it displays the Creation in progress status.
   Creating the broker takes about 15 minutes.
   When your broker is created successfully, Amazon MQ displays the Running status.
Step 2: connect a JVM-based application to your broker

After you create a RabbitMQ broker, you can connect your application to it. The following examples show how you can use the RabbitMQ Java client library to create a connection to your broker, create a queue, and send a message. You can connect to RabbitMQ brokers using supported RabbitMQ client libraries for a variety of languages. For more information about supported RabbitMQ client libraries, see RabbitMQ client libraries and developer tools.

Prerequisites

Note
The following prerequisite steps are only applicable to RabbitMQ brokers created without public accessibility. If you are creating a broker with public accessibility you can skip them.

Enable VPC attributes

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

Enable inbound connections

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

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

   a. Choose **Add Rule**.
   b. For **Type**, select **Custom TCP**.
   c. For **Source**, leave **Custom** selected and then type the IP address of the system that you want to be able to access the web console (for example, 192.0.2.1).
   d. Choose **Save**.

   Your broker can now accept inbound connections.

**Add Java dependencies**

If you are using Apache Maven for automating builds, add the following dependency to your `pom.xml` file. For more information about Project Object Model files in Apache Maven, see [Introduction to the POM](#).

```xml
<dependency>
  <groupId>com.rabbitmq</groupId>
  <artifactId>amqp-client</artifactId>
  <version>5.9.0</version>
</dependency>
```

If you are using **Gradle** for automating builds, declare the following dependency.

```groovy
dependencies {
  compile 'com.rabbitmq:amqp-client:5.9.0'
}
```

**Import Connection and Channel classes**

RabbitMQ Java client uses `com.rabbitmq.client` as its top-level package, with `Connection` and `Channel` API classes representing an AMQP 0-9-1 connection and channel, respectively. Import the `Connection` and `Channel` classes before using them, as shown in the following example.

```java
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.Channel;
```

**Create a ConnectionFactory and connect to your broker**

Use the following example to create an instance of the `ConnectionFactory` class with the given parameters. Use the `setHost` method to configure the broker endpoint you noted earlier. For AMQPS wire-level connections, use port 5671.

```java
ConnectionFactory factory = new ConnectionFactory();
factory.setUsername(username);
factory.setPassword(password);
// Replace the URL with your information
factory.setHost("b-c8352341-ec91-4a78-ad9c-a43f23d325bb.mq.us-west-2.amazonaws.com");
factory.setPort(5671);
// Allows client to establish a connection over TLS
factory.useSslProtocol();
```
// Create a connection
Connection conn = factory.newConnection();

// Create a channel
Channel channel = conn.createChannel();

**Publish a message to an exchange**

You can use `Channel.basicPublish` to publish messages to an exchange. The following example uses the AMQP `Builder` class to build a message properties object with content-type `plain/text`.

```java
byte[] messageBodyBytes = "Hello, world!".getBytes();
channel.basicPublish(exchangeName, routingKey,
    new AMQP.BasicProperties.Builder()
    .contentType("text/plain")
    .userId("userId")
    .build(),
    messageBodyBytes);
```

**Note**

Note that `BasicProperties` is an inner class of the autogenerated holder class, `AMQP`.

**Subscribe to a queue and receive a message**

You can receive a message by subscribing to a queue using the `Consumer` interface. Once subscribed, messages will then be delivered automatically as they arrive.

The easiest way to implement a `Consumer` is to use the subclass `DefaultConsumer`. A `DefaultConsumer` object can be passed as part of a `basicConsume` call to set up the subscription as shown in the following example.

```java
boolean autoAck = false;
channel.basicConsume(queueName, autoAck, "myConsumerTag",
    new DefaultConsumer(channel) {
        @Override
        public void handleDelivery(String consumerTag,
            Envelope envelope,
            AMQP.BasicProperties properties,
            byte[] body)
            throws IOException {
            String routingKey = envelope.getRoutingKey();
            String contentType = properties.getContentType();
            long deliveryTag = envelope.getDeliveryTag();
            // (process the message components here ...)
            channel.basicAck(deliveryTag, false);
        }
    });
```

**Note**

Because we specified `autoAck = false`, it is necessary to acknowledge messages delivered to the `Consumer`, most conveniently done in the `handleDelivery` method, as shown in the example.

**Close your connection and disconnect from the broker**

In order to disconnect from your RabbitMQ broker, close both the channel and connection as shown in the following.
Step 3: (Optional) connect to an AWS Lambda function

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

To connect your broker to a Lambda function

1. Add the following IAM role permissions to your Lambda function execution role.
   - mq:DescribeBroker
   - ec2:CreateNetworkInterface
   - ec2:DeleteNetworkInterface
   - ec2:DescribeNetworkInterfaces
   - ec2:DescribeSecurityGroups
   - ec2:DescribeSubnets
   - ec2:DescribeVpcs
   - logs:CreateLogGroup
   - logs:CreateLogStream
   - logs:PutLogEvents
   - secretsmanager:GetSecretValue

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

2. (Optional) If you have created a broker without public accessibility, you must do one of the following to allow Lambda to connect to your broker:
   - Configure one NAT gateway per public subnet. For more information, see Internet and service access for VPC-connected functions in the AWS Lambda Developer Guide.
   - Create a connection between your Amazon Virtual Private Cloud (Amazon VPC) and Lambda using a VPC endpoint. Your Amazon VPC must also connect to AWS Security Token Service (AWS STS) and Secrets Manager endpoints. For more information, see Configuring interface VPC endpoints for Lambda in the AWS Lambda Developer Guide.

3. Configure your broker as an event source for a Lambda function using the AWS Management Console. You can also use the create-event-source-mapping AWS Command Line Interface command.

4. Write some code for your Lambda function to process the messages from your consumed from your broker. The Lambda payload that retrieved by your event source mapping depends on the engine type of the broker. The following is an example of a Lambda payload for an Amazon MQ for RabbitMQ queue.
Note
In the example, test is the name of the queue, and / is the name of the default virtual host. When receiving messages, the event source lists messages under test::/.

```
{
  "eventSource": "aws:rmq",
  "rmqMessagesByQueue": {
    "test::/": [
      {
        "basicProperties": {
          "contentType": "text/plain",
          "contentEncoding": null,
          "headers": {
            "header1": {
              "bytes": [
                118,
                97,
                108,
                117,
                101,
                49
              ]
            },
            "header2": {
              "bytes": [
                118,
                97,
                108,
                117,
                101,
                50
              ]
            }
          },
          "numberInHeader": 10
        }
      }
    ],
    "deliveryMode": 1,
    "priority": 34,
    "correlationId": null,
    "replyTo": null,
    "expiration": "60000",
    "messageId": null,
    "timestamp": "Jan 1, 1970, 12:33:41 AM",
    "type": null,
    "userId": "AIDACKCEVSQ6C2EXAMPLE",
    "appId": null,
    "clusterId": null,
    "bodySize": 80
  },
  "redelivered": false,
  "data": "eyJ0aW1lb3V0IjowLCJkYXRhIjoiQ1pybWYwR3c4T3Y0YnFMUXhENEUifQ=="
}
```

For more information about connecting Amazon MQ to Lambda, the options Lambda supports for an Amazon MQ event source, and event source mapping errors, see Using Lambda with Amazon MQ in the AWS Lambda Developer Guide.
Step 4: 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:

- Editing broker engine version, Amazon CloudWatch Logs, and maintenance preferences (p. 34)
- Listing brokers and viewing broker details (p. 19)
- Creating and managing ActiveMQ broker users (p. 58)
- Rebooting a Broker (p. 27)
- Accessing CloudWatch metrics for Amazon MQ (p. 168)

You can also begin to dive deep into Amazon MQ best practices (p. 69) and Amazon MQ REST APIs before planning to migrate to Amazon MQ.
Managing an Amazon MQ broker

In the following sections, you can find instructions for managing and maintaining your Amazon MQ brokers.

Topics
- Listing Amazon MQ brokers and viewing broker details (p. 19)
- Maintaining an Amazon MQ broker (p. 21)
- Upgrading an Amazon MQ broker engine version (p. 24)
- Rebooting an Amazon MQ broker (p. 27)
- Deleting an Amazon MQ broker (p. 28)

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.

   The following information is displayed for each broker:
   - Name
   - Creation date
   - Status (p. 140)
   - Deployment mode
   - Instance type (p. 136)

2. Choose your broker's name.

   For ActiveMQ brokers, on the MyBroker page, the configured (p. 81) Details are displayed for your broker:

---

19
For Amazon MQ for RabbitMQ brokers, you can view your selected settings on the MyBroker2 page, under the Details section as shown in the following.

Under the Details section, the following information is displayed:

- In the Connections section, for Amazon MQ for ActiveMQ brokers, the web console URL and the wire-level protocol endpoints.
Maintaining a broker

In the **Connections** section, for Amazon MQ for RabbitMQ brokers, the web console URL and the secure AMQP endpoint.

**Connections**

Access your queues and exchanges and connect your application to the broker. If you disable public accessibility for your broker, your endpoints are reachable only within a VPC.

**ActiveMQ Web Console**

For more information, see the documentation on the Amazon MQ for RabbitMQ console.

**Endpoints**

<table>
<thead>
<tr>
<th>Name</th>
<th>URL</th>
</tr>
</thead>
</table>

- For Amazon MQ for ActiveMQ brokers, in the **Users** section, the users (p. 82) associated with the broker

  **Important**

  Managing users via the AWS Management Console and the Amazon MQ API is not supported for Amazon MQ for RabbitMQ brokers.

Maintaining an Amazon MQ broker

Periodically, Amazon MQ performs maintenance to the hardware, operating system, or the engine software a message broker. The duration of the maintenance varies, but can last up to two hours, depending on the operations that are scheduled for your message broker. For example, if you've activated automatic minor engine version upgrades (p. 26), or changed the broker instance type, Amazon MQ will apply your changes during the next scheduled maintenance window.

To minimize downtime during a maintenance window, we recommend selecting a broker deployment mode with high availability across multiple Availability Zones (AZ). Depending on your broker engine type, Amazon MQ provides the following Multi-AZ deployment modes.

- **Amazon MQ for ActiveMQ** – Amazon MQ for ActiveMQ provides active/standby (p. 85) deployments for high availability. In active/standby mode, Amazon MQ performs maintenance operations one instance at a time, ensuring that at least one instance remains available. In addition, you can configure a network of brokers (p. 86) with maintenance windows scattered across the week.
Adjusting the broker maintenance window

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

**Important**
You can only adjust the maintenance window of a broker up to four times before the next scheduled maintenance window. Amazon MQ applies a limit of four maintenance window adjustments to ensure that critical software and security patches, as well as important hardware upgrades, are not indefinitely deferred and postponed. Once a broker maintenance window is completed, Amazon MQ resets the limit, allowing you to adjust the schedule before the next maintenance window occurs.

**AWS Management Console**

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

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

**Note**
The Start time options are configured in UTC+0 time zone.
5. Scroll to the bottom of the page, and choose Save. The maintenance window is adjusted immediately.
6. On the broker details page, under Maintenance window, verify that your new preferred schedule is displayed.
AWS CLI

To adjust the broker maintenance window using the AWS CLI

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

   - `--broker-id` – The unique ID that Amazon MQ generates for the broker. You can parse the ID from your broker ARN. For example, given the following ARN, arn:aws:mq:us-east-2:123456789012:broker:MyBroker:b-1234a5b6-78cd-901e-2fgh-3i45j6k17819, the broker ID would be `b-1234a5b6-78cd-901e-2fgh-3i45j6k17819`.
   - `--maintenance-window-start-time` – The parameters that determine the weekly maintenance window start time provided in the following structure.
     - `DayOfWeek` – The day of the week, in the following syntax: MONDAY | TUESDAY | WEDNESDAY | THURSDAY | FRIDAY | SATURDAY | SUNDAY
     - `TimeOfDay` – The time, in 24-hour format.
     - `TimeZone` – (Optional) The time zone, in either the Country/City, or the UTC offset format. Set to UTC by default.

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

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

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

Amazon MQ API

To adjust the broker maintenance window using the Amazon MQ API

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

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

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

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

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

Amazon MQ provides new broker engine versions for all supported broker engine type. New engine versions might include security patches, bug fixes and other broker engine improvements. When Amazon MQ supports a new engine version, you can control how and when to upgrade your broker.

Broker engine versions are organized as $X.Y.Z$. In the Amazon MQ implementation of each engine type, $X.Y$ is considered a major version and $Z$ is considered a minor version. There are two types of upgrades:

- **Major version upgrade** – Occurs when the major engine version numbers change. For example, upgrading from version 1.0 to version 1.1 is considered a major version upgrade.
- **Minor version upgrade** – Occurs when only the minor engine version number changes. For example, upgrading from version 1.1.0 to version 1.1.1 is considered a minor version upgrade.

For more information about major and minor version management for each specific broker engine type, see the following topics.

- the section called “Version management” (p. 115)
- the section called “Version management” (p. 134)

When you activate the automatic minor version upgrade option, Amazon MQ upgrades your broker to new minor versions as they become available. Automatic minor version upgrades occur only if the broker is running a minor engine version that is lower than the new recommended minor version. For major upgrades, you must manually upgrade the engine version.

Both manual and automatic version upgrades occur during the scheduled maintenance window or after you reboot your broker (p. 27).

The following topics describe how you can manually upgrade the broker engine version, and activate automatic minor version upgrades.

**Topics**
- Manually upgrading the engine version (p. 24)
- Automatically upgrading the minor engine version (p. 26)

**Manually upgrading the engine version**

To manually upgrade the engine version of a broker to a new major or minor version, you can use the AWS Management Console, the AWS CLI, or the Amazon MQ API.

**AWS Management Console**

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

1. Sign in to the Amazon MQ console.
2. In the left navigation pane, choose **Brokers**, and then choose the broker that you want to upgrade from the list.

3. On the broker details page, choose **Edit**.

4. Under **Specifications**, for **Broker engine version** choose the new version number from the dropdown list.

5. Scroll to the bottom of the page, and choose **Schedule modifications**.

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

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

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

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

**AWS CLI**

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

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

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

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

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

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

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

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

**Amazon MQ API**

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

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

```
PUT /v1/brokers/broker-id HTTP/1.1
```
Automatically upgrading the minor engine version

You can control whether automatic minor version upgrade is activated for a broker when you first create the broker, or by modifying broker preferences. To activate auto minor version upgrades for an existing broker, you can use the AWS Management Console, the AWS CLI, or the Amazon MQ API.

**AWS Management Console**

**To activate automatic minor version upgrades by using the AWS Management Console**

1. Sign in to the Amazon MQ console.
2. In the left navigation pane, choose **Brokers**, and then choose the broker that you want to upgrade from the list.
3. On the broker details page, choose **Edit**.
4. Under **Maintenance**, choose **Enable automatic minor version upgrades**.
   
   **Note**
   If the option is already selected, you do not need to make any changes.
5. Choose **Save** at the bottom of the page.

**AWS CLI**

To activate automatic minor version upgrades via the AWS CLI, use the `update-broker` CLI command and specify the following parameters.

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

---

**Automatically upgrading the minor engine version**

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

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

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

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

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

**Important**
Your broker will be offline while it is being rebooted.
• `--auto-minor-version-upgrade` – Activates the auto minor version upgrade option.

```bash
aws mq update-broker --broker-id broker-id --auto-minor-version-upgrade
```

If you want to deactivate auto minor version upgrades for your broker, use the `--no-auto-minor-version-upgrade` parameter.

**Amazon MQ API**

To activate automatic minor version upgrades via the Amazon MQ API, use the `UpdateBroker` API operation. Specify `broker-id` as a path parameter. The following example assumes a broker in the `us-west-2` region. For more information about available Amazon MQ endpoints, see Amazon MQ endpoints and quotas. in the AWS General Reference

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

Use the `autoMinorVersionUpgrade` property in the request payload to activate auto minor version upgrade.

```json
{
    "autoMinorVersionUpgrade": "true"
}
```

If you want to deactivate auto minor version upgrades for your broker, set `"autoMinorVersionUpgrade": "false"` in the request payload.

### 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`).

**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.

Deleting an Amazon MQ broker

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

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

Deleting an Amazon MQ broker

1. Sign in to the Amazon MQ console.
2. From the broker list, select your broker (for example, MyBroker) and then choose Delete.
3. In the Delete MyBroker? dialog box, type delete and then choose Delete.
   
   Deleting a broker takes about 5 minutes.
Amazon MQ Tutorials

The following tutorials show how you can work with Amazon MQ brokers using the AWS Management Console to perform common tasks and operations.

**Topics**
- Accessing the broker web console without public accessibility (p. 29)
- ActiveMQ tutorials (p. 30)
- RabbitMQ tutorials (p. 60)

Accessing the broker web console 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 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 a broker's 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 public EC2 instance. For example:

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

A forward proxy server is started on your machine.

4. Install a proxy client such as FoxyProxy on your machine.

5. Configure your proxy client using the following settings:
   - For proxy type, specify SOCKS5.
   - For IP address, DNS name, and server name, specify localhost.
   - For port, specify 8080.
   - Remove any existing URL patterns.
   - For the URL pattern, specify *.mq.*.amazonaws.com*
   - For the connection type, specify HTTP(S).

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

**ActiveMQ tutorials**

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

**Topics**
- Creating and configuring an ActiveMQ broker (p. 30)
- Editing broker engine version, instance type, CloudWatch logs, and maintenance preferences (p. 34)
- Creating and configuring an Amazon MQ network of brokers (p. 35)
- Creating and applying ActiveMQ broker configurations (p. 38)
- Editing ActiveMQ broker configurations and managing configuration revisions (p. 40)
- Connecting a Java application to your Amazon MQ broker (p. 43)
- Integrating ActiveMQ brokers with LDAP (p. 47)
- Creating and managing ActiveMQ broker users (p. 58)

**Creating and configuring 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, t3) and *size* (large, micro) is a *broker instance type* (for example, mq.m5.large). For more information, see *Broker* (p. 79).

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

**Topics**
- Step 1: Configure Basic Broker Settings (p. 31)
- Step 2: (Optional) Configure Additional Broker Settings (p. 32)
- Step 3: Finish Creating the Broker (p. 33)
Step 1: Configure Basic Broker Settings

1. Sign in to the Amazon MQ console.

2. On the Select broker engine page, choose Apache ActiveMQ.

3. On the Select deployment and storage page, in the Deployment mode and storage type section, do the following:
   a. Choose the Deployment mode (for example, Active/standby broker). For more information, see Broker Architecture (p. 83).

      - A Single-instance broker is comprised of one broker in one Availability Zone. The broker communicates with your application and with an Amazon EBS or Amazon EFS storage volume. For more information, see Amazon MQ single-instance broker (p. 84).

      - An Active/standby broker for high availability is comprised of two brokers in two different Availability Zones, configured in a redundant pair. These brokers communicate synchronously with your application, and with Amazon EFS. For more information, see Amazon MQ active/standby broker for high availability (p. 85).

      - For more information on the sample blueprints for a network of brokers, see Sample blueprints (p. 88).

   b. Choose the Storage type (for example, EBS). For more information, see Storage (p. 82).

      Note
      Amazon EBS replicates data within a single Availability Zone and doesn't support the ActiveMQ active/standby (p. 85) deployment mode.

   c. Choose Next.

4. On the Configure settings page, in the Details section, do the following:
   a. Enter the Broker name.

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

   b. Choose the Broker instance type (for example, mq.m5.large). For more information, see Broker instance types (p. 136).

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

   - Your username can contain only alphanumeric characters, dashes, periods, underscores, and tildas (- . _ ~).

   - Your password must be at least 12 characters long, contain at least 4 unique characters and must not contain commas, colons, or equal signs (,:=).

      Important
      Do not add personally identifiable information (PII) or other confidential or sensitive information in broker usernames. Broker usernames are accessible to other AWS services, including CloudWatch Logs. Broker usernames are not intended to be used for private or sensitive data.
Step 2: (Optional) Configure Additional 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. Amazon MQ only supports default VPC tenancy, and does not support dedicated VPC tenancy.
- **Encryption** – Choose the customer master key to encrypt your data. See Encryption at rest (p. 144).
- **Public accessibility** – Disabling public accessibility makes the broker accessible only within your VPC. For more information, see Prefer brokers without public accessibility (p. 166) and Accessing the broker web console without public accessibility (p. 29).

1. Expand the **Additional 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. 81) and Amazon MQ Broker Configuration Parameters (p. 97).
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. 183).

   Important
   If you don't add the CreateLogGroup permission to your Amazon MQ user (p. 184) 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. 185), 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 customer master key (CMK) that will be used to encrypt your data. See Encryption at rest (p. 144).

      c. 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. This allows 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 Deploy.

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

Creating the broker takes about 15 minutes.

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Status</th>
<th>Deployment mode</th>
<th>instance type</th>
</tr>
</thead>
<tbody>
<tr>
<td>MyBroker</td>
<td>Running</td>
<td>Single-instance broker</td>
<td>mq.m5.large</td>
</tr>
</tbody>
</table>

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 Broker Architecture (p. 83).

For wire-level protocol endpoints, you can allow your application to connect to either endpoint by using the Failover Transport.
Editing broker engine version, instance type, CloudWatch logs, and maintenance preferences

In addition to editing broker configurations and managing configuration revisions (p. 40), 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. 96).

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

**Edit ActiveMQ broker options**

1. Sign in to the Amazon MQ console.
2. From the broker list, select your broker (for example, **MyBroker**) and then choose **Edit**.
3. On the **Edit MyBroker** page, in the **Specifications** section, select a **Broker engine version** or a **Broker Instance type**.
4. In the **Configuration** section, select the configuration and revision for your broker. For more information, see Editing and Managing Broker Configurations (p. 40).
5. In the **Security and network** section, select a group from the **Security group(s)** drop-down, or choose **Create a new security group** to open the Amazon VPC console.
6. In the **CloudWatch Logs** section, choose whether to publish **General** logs and **Audit** logs to Amazon CloudWatch Logs.

   For more information about configuring CloudWatch logs for ActiveMQ brokers, see Configuring Amazon MQ to publish logs to Amazon CloudWatch Logs (p. 183).

   **Important**
   If you don't add the **CreateLogGroup** permission to your Amazon MQ user (p. 184) 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. 185), the broker can't publish the logs to CloudWatch Logs.
7. 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. This allows the healthy standby instance to become active and to begin accepting incoming communications.
8. 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.
Creating and configuring an Amazon MQ network of brokers

A network of brokers is comprised of multiple simultaneously active single-instance brokers (p. 84) or active/standby brokers (p. 85). You can configure networks of brokers in a variety of topologies (p. 88) (for example, concentrator, hub-and-spokes, tree, or mesh), depending on your application's needs, such as high availability and scalability. For instance, a hub and spoke (p. 91) network of brokers can increase resiliency, preserving messages if one broker is not reachable. A network of brokers with a concentrator (p. 92) topology can collect messages from a larger number of brokers accepting incoming messages, and concentrate them to more central brokers, to better handle the load of many incoming messages. In this tutorial, you learn how to create a two-broker network of brokers with a source and sink topology.

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

- Amazon MQ Network of brokers (p. 86)
- Configure Your Network of Brokers Correctly (p. 73)
- networkConnector (p. 112)
- networkConnectionStartAsync (p. 108)
- Networks of Brokers in the ActiveMQ documentation

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

Topics

- Prerequisites (p. 35)
- Step 1: Allow Traffic between Brokers (p. 36)
- Step 2: Configure Network Connectors for Your Broker (p. 36)
- Next Steps (p. 37)

Prerequisites

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

- Two or more simultaneously active brokers (named MyBroker1 and MyBroker2 in this tutorial). For more information about creating brokers, see Creating and configuring a broker (p. 30).
- The two brokers must be in the same VPC or in peered VPCs. For more information about VPCs, see What is Amazon VPC? in the Amazon VPC User Guide and What is VPC Peering? in the Amazon VPC Peering Guide.
  
  **Important**
  
  If you don't have a default VPC, subnet(s), or security group, you must create them first. For more information, see the following in the Amazon VPC User Guide:
  - Creating a Default VPC
  - Creating a Default Subnet
  - Creating a Security Group

- Two users with identical usernames and passwords for both brokers. For more information about creating users, see Creating and managing ActiveMQ broker users (p. 58).
  
  **Note**
  
  When integrating LDAP authentication with a network of brokers, make sure that the user exists both as an ActiveMQ brokers, as well as an LDAP user.
Creating and configuring a network of brokers

The following example uses two single-instance brokers (p. 84). However, you can create networks of brokers using active/standby brokers (p. 85) or a combination of broker deployment modes.

Step 1: Allow Traffic between Brokers

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

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

   The Security Groups page of the EC2 Dashboard is displayed.

2. From the security group list, choose your security group.

3. At the bottom of the page, choose Inbound, and then choose Edit.

4. In the Edit inbound rules dialog box, add a rule for the OpenWire endpoint.
   a. Choose Add Rule.
   b. For Type, select Custom TCP.
   c. For Port Range, type the OpenWire port (61617).
   d. Do one of the following:
      • If you want to restrict access to a particular IP address, for Source, leave Custom selected, and then enter the IP address of MyBroker1, followed by /32. (This converts the IP address to a valid CIDR record). For more information see Elastic Network Interfaces.
        Tip
        To retrieve the IP address of MyBroker1, on the Amazon MQ console, choose the name of the broker and navigate to the Details section.
      • If all the brokers are private and belong to the same VPC, for Source, leave Custom selected and then type the ID of the security group you are editing.
        Note
        For public brokers, you must restrict access using IP addresses.
   e. Choose Save.

Your broker can now accept inbound connections.

Step 2: Configure Network Connectors for Your Broker

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

1. Edit the configuration revision for broker MyBroker1.
   a. On the MyBroker1 page, choose Edit.
   b. On the Edit MyBroker1 page, in the Configuration section, choose View.

      The broker engine type and version that the configuration uses (for example, Apache ActiveMQ 5.15.0) are displayed.
   c. On the Configuration details tab, the configuration revision number, description, and broker configuration in XML format are displayed.
   d. Choose Edit configuration.
   e. At the bottom of the configuration file, uncomment the <networkConnectors> section and include the following information:
      • The name for the network connector.
      • The ActiveMQ Web Console username (p. 35) that is common to both brokers.
Creating and configuring a network of brokers

- Enable duplex connections.
- Do one of the following:
  - If you are connecting the broker to a single-instance broker, use the **static:** prefix and the OpenWire endpoint **uri** for MyBroker2. For example:

    ```xml
    <networkConnectors>
      <networkConnector name="connector_1_to_2" userName="myCommonUser"
        duplex="true"
        uri="static:(ssl://b-1234a5b6-78cd-901e-2fgh-3145j6k178l9-1.mq.us-east-2.amazonaws.com:61617)"/>
    </networkConnectors>
    
    Note
    Don’t include the password for the ActiveMQ user.
    
    f. Choose Save.
    g. In the Save revision dialog box, type Add network of brokers connector for MyBroker2.
    h. Choose Save to save the new revision of the configuration.

2. Edit MyBroker1 to set the latest configuration revision to apply immediately.
   
   a. On the MyBroker1 page, choose Edit.
   b. On the Edit MyBroker1 page, in the Configuration section, choose Schedule Modifications.
   c. In the Schedule broker modifications section, choose to apply modifications **Immediately**.
   d. Choose Apply.

   MyBroker1 is rebooted and your configuration revision is applied.

   The network of brokers is created.

**Next Steps**

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

**Important**

Make sure that you **enable inbound connections** (p. 117) **from your local machine** for broker MyBroker1 on port 8162 (for the ActiveMQ Web Console) and port 61617 (for the OpenWire endpoint).

You might also need to adjust your security group(s) settings to allow the producer and consumer to connect to the network of brokers.

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

   In the **Network Bridges** section, the name and the address of MyBroker2 are listed in the **Remote Broker** and **Remote Address** columns.

4. From any machine that has access to broker MyBroker2, create a consumer. For example:

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

   The consumer connects to the OpenWire endpoint of MyBroker2 and begins to consume messages from queue MyQueue.

5. From any machine that has access to broker MyBroker1, create a producer and send some messages. For example:

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

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

**Creating and applying ActiveMQ broker configurations**

A **configuration** contains all of the settings for your ActiveMQ broker, in XML format (similar to ActiveMQ's activemq.xml file). You can create a configuration before creating any brokers. You can then apply the configuration to one or more brokers. 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. This allows the healthy standby instance to become active and to begin accepting incoming communications.

For more information, see the following:

- Configuration (p. 81)
- Amazon MQ Broker Configuration Lifecycle (p. 96)
- Amazon MQ Broker Configuration Parameters (p. 97)
- Editing and Managing Broker Configurations (p. 40)

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 (p. 39)
- Step 2: Create a New Configuration Revision (p. 39)
- Step 3: Apply a Configuration Revision to Your Broker (p. 40)

### 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**.

    ![Amazon MQ console](image)

3. On the **Configurations** page, choose **Create configuration**.

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

    **Note**
    To learn more about ActiveMQ engine versions supported by Amazon MQ for ActiveMQ, see the section called “Version management” (p. 115).

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 5.15.8*) 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.

    ```xml
    <?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). -->
    5  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. 97).

   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. 42) or reboot the broker (p. 27). For more information, see Amazon MQ Broker Configuration Lifecycle (p. 96).

   Currently, you can't delete a configuration.

---

**Step 3: Apply a Configuration Revision to Your Broker**

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

   ![Amazon MQ](image)

   **Brokers**

   **Configurations**

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.

---

**Editing ActiveMQ 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**.

---

40
**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. This allows 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. 81)
- Amazon MQ Broker Configuration Lifecycle (p. 96)
- Amazon MQ Broker Configuration Parameters (p. 97)
- Creating and applying broker configurations (p. 38)

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 (p. 41)
- To Edit the Current Configuration Revision (p. 34)
- To Apply a Configuration Revision to Your Broker (p. 42)
- To Roll Back Your Broker to the Last Configuration Revision (p. 43)

**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 5.15.8*) 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 5.15.8) are displayed.

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

Note
Editing the current configuration creates a new configuration revision.

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

The Save revision dialog box is displayed.

8. (Optional) Type A description of the changes in this revision.
9. Choose Save.

The new revision of the configuration is saved.

Important
The Amazon MQ console automatically sanitizes invalid and prohibited configuration parameters according to a schema. For more information and a full list of permitted XML parameters, see Amazon MQ Broker Configuration Parameters (p. 97).
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. 42) or reboot the broker (p. 27). For more information, see Amazon MQ Broker Configuration Lifecycle (p. 96). Currently, you can't 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.
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.

Connecting a Java application to your Amazon MQ broker

After you create an Amazon MQ ActiveMQ broker, you can connect your application to it. The following examples show how you can use the Java Message Service (JMS) to create a connection to the broker, create a queue, and send a message. For a complete, working Java example, see Working Java Example (p. 116).

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

Topics
- Prerequisites (p. 44)
- To Create a Message Producer and Send a Message (p. 45)
- To Create a Message Consumer and Receive the Message (p. 46)
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 web console URL and wire-level protocols.
4. In the Details section, under Security and network, choose the name of your security group or

   The Security Groups page of the EC2 Dashboard is displayed.
5. From the security group list, choose your security group.
6. At the bottom of the page, choose Inbound, and then choose Edit.
7. In the Edit inbound rules dialog box, add a rule for every URL or endpoint that you want to be publicly accessible (the following example shows how to do this for a broker web console).
   a. Choose Add Rule.
   b. For Type, select Custom TCP.
   c. For Port Range, type the web console port (8162).
   d. For Source, leave Custom selected and then type the IP address of the system that you want to be able to access the web console (for example, 192.0.2.1).
   e. Choose Save.

Your broker can now accept inbound connections.

Add Java Dependencies

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

```xml
<dependencies>
  <dependency>
    <groupId>org.apache.activemq</groupId>
    <artifactId>activemq-client</artifactId>
    <version>5.15.8</version>
  </dependency>
  <dependency>
    <groupId>org.apache.activemq</groupId>
    <artifactId>activemq-pool</artifactId>
    <version>5.15.8</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 Broker Architecture (p. 83). For wire-level protocol endpoints, you can allow your application to connect to either endpoint by using the Failover Transport.

   ```java
   // 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. 70).

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

   ```java
   // 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 string "Hello from Amazon MQ!" and then send the message.
   final String text = "Hello from Amazon MQ!";
   TextMessage producerMessage = producerSession.createTextMessage(text);

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

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

```java
// 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. 70).

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

```java
// 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.

```java
// 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.

```java
consumer.close();
consumerSession.close();
consumerConnection.close();
```
Integrating ActiveMQ brokers with LDAP

Important
LDAP integration is not supported for RabbitMQ brokers.

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

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

Amazon MQ offers a choice between native ActiveMQ authentication and LDAP authentication and authorization to manage user permissions. For information about restrictions related to ActiveMQ usernames and passwords, see Users (p. 189).

To authorize ActiveMQ users and groups to works with queues and topics, you must edit your broker's configuration (p. 40). 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. 166) and authorizationEntry (p. 111).

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

Topics

- Integrate LDAP with ActiveMQ (p. 47)
- Prerequisites (p. 48)
- Getting Started with LDAP (p. 48)
- How LDAP integration works (p. 51)

Integrate LDAP with ActiveMQ

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

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

Amazon MQ provides LDAP support using the Apache ActiveMQ JAAS plugin. Any LDAP server, such as Microsoft Active Directory or OpenLDAP that is supported by the plugin is also supported by Amazon MQ. For more information about the plugin, see the Security section of the Active MQ documentation.

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

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

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

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

Getting Started with LDAP

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

Provide the following information about the service account:

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

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

- **Service account password** The password of the user performing the initial bind.

The following image highlights where to supply these details.
In the LDAP login configuration section, provide the following required information:

- **User Base** The distinguished name of the node in the directory information tree (DIT) that will be searched for users.

- **User Search Matching** The LDAP search filter that will be used to find users within the `userBase`. The client's username will be substituted into the `{0}` placeholder in the search filter. For more information, see Authentication (p. 51) and Authorization (p. 52).

- **Role Base** The distinguished name of the node in the DIT that will be searched for roles. Roles can be configured as explicit LDAP group entries in your directory. A typical role entry may consist of one attribute for the name of the role, such as `common name (CN)`, and another attribute, such as `member`, with values representing the distinguished names or usernames of the users belonging to the role group. For example, given the organizational unit, `group`, you can provide the following distinguished name: `ou=group,dc=example,dc=com`.

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

The following image highlights where to specify these details.
In the **Optional settings** section, you can provide the following optional information:

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

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

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

- **Role Name** The group name attribute in a role entry whose value is the name of that role. For example, you can specify `cn` for a group entry's **common name**. If authentication succeeds, the user is assigned the the value of the `cn` attribute for each role entry that they are a member of.

- **User Search Subtree** Defines the scope for the LDAP user search query. If true, the scope is set to search the entire subtree under the node defined by `userBase`.

- **Role Search Subtree** Defines the scope for the LDAP role search query. If true, the scope is set to search the entire subtree under the node defined by `roleBase`.
The following image highlights where to specify these optional settings.

![Image showing optional settings for LDAP integration]

### How LDAP integration works

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

**Authentication**

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

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

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

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

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

**Authorization**

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

**Note**

To be able to use the `cachedLdapAuthorizationMap` element in your broker's `activemq.xml` configuration file, you must choose the LDAP Authentication and Authorization option when creating a configuration via the AWS Management Console.
You must provide the following three attributes as part of the cachedLDAPAuthorizationMap element:

- queueSearchBase
- topicSearchBase
- tempSearchBase

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

- connectionURL
- connectionUsername
- connectionPassword

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

The following demonstrates a working example of the cachedLdapAuthorizationMap.

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

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

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

```
OU=Queue,OU=Destination,OU=corp,DC=corp,DC=example,DC=com
```
Similarly, permissions rules for topics and temp destinations would be located at the same level in the DIT:

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

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

```
OU=DEMO.EVENTS.$,OU=Queue,OU=Destination,OU=corp,DC=corp,DC=example,DC=com
```
Note
The DEMO.EVENTS.* OU is within the Queue OU.

For more info on wildcards in ActiveMQ, see Wildcards

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

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

Within each OU that represents a destination or a wildcard, you must create three security groups. As with all permissions in ActiveMQ, these are read/write/admin permissions. For more information on what each of these permissions allows a user to do, see Security in the ActiveMQ documentation.
You must name these security groups read, write, and admin. Within each of these security groups, you can add users or groups, who will then have permission to perform the associated actions. You’ll need these security groups for each wildcard destination set or individual destination.

**Note**
When you create the admin group, a conflict will arise with the group name. This conflict happens because the legacy pre-Windows 2000 rules do not allow groups to share the same name, even if the groups are in different locations of the DIT. The value in the **pre-Windows 2000** text box has no impact on the setup, but it must be globally unique. To avoid this conflict, you can append a *uuid* suffix to each admin group.
Adding a user to the admin security group for a particular destination will enable the user to create and delete that topic. Adding them to the read security group will enable them to read from the destination, and adding them to the write group will enable them to write to the destination.

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

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

Creating and managing ActiveMQ 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 (p. 58)
- To edit an existing user (p. 59)
- To delete an existing user (p. 59)

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 View details.

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

<table>
<thead>
<tr>
<th>Username</th>
<th>Console access</th>
<th>Groups</th>
<th>Pending modifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>paolo.santos</td>
<td>No</td>
<td>Devs</td>
<td></td>
</tr>
<tr>
<td>jane.doe</td>
<td>Yes</td>
<td>Admins</td>
<td></td>
</tr>
</tbody>
</table>

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. 42) or
reboot the broker (p. 27). For more information, see Amazon MQ Broker Configuration Lifecycle (p. 96).

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 View details.

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

<table>
<thead>
<tr>
<th>Username</th>
<th>Console access</th>
<th>Groups</th>
<th>Pending modifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>paolo.santos</td>
<td>No</td>
<td>Devs</td>
<td></td>
</tr>
<tr>
<td>jane.doe</td>
<td>Yes</td>
<td>Admins</td>
<td></td>
</tr>
</tbody>
</table>

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. 42) or reboot the broker (p. 27). For more information, see Amazon MQ Broker Configuration Lifecycle (p. 96).

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 View details.

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

<table>
<thead>
<tr>
<th>Username</th>
<th>Console access</th>
<th>Groups</th>
<th>Pending modifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>paolo.santos</td>
<td>No</td>
<td>Devs</td>
<td></td>
</tr>
<tr>
<td>jane.doe</td>
<td>Yes</td>
<td>Admins</td>
<td></td>
</tr>
</tbody>
</table>

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. 42) or reboot the broker (p. 27). For more information, see Amazon MQ Broker Configuration Lifecycle (p. 96).
RabbitMQ tutorials

The following tutorials show how you can configure and use RabbitMQ on Amazon MQ. To learn more about working with supported client libraries in a variety of programming languages such as Node.js, Python, .NET, and more, see RabbitMQ Tutorials in the RabbitMQ Getting Started Guide.

Topics

- Editing broker preferences (p. 60)
- Using Python Pika with Amazon MQ for RabbitMQ (p. 60)
- Resolving RabbitMQ paused queue synchronization (p. 65)

Editing broker preferences

You can edit your broker preferences, such as enabling or disabling CloudWatch logs using the AWS Management Console.

Edit RabbitMQ broker options

1. Sign in to the Amazon MQ console.
2. From the broker list, select your broker (for example, MyBroker) and then choose Edit.
3. On the Edit MyBroker page, in the Specifications section, select a Broker engine version or a Broker Instance type.

4. In the CloudWatch Logs section, click the toggle button to enable or disable general logs. No other steps are required.

   Note

   - For RabbitMQ brokers, Amazon MQ automatically uses a Service-Linked Role (SLR) to publish general logs to CloudWatch. For more information, see the section called “Using service-linked roles” (p. 159)
   - Amazon MQ does not support audit logging for RabbitMQ brokers.

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

   To upgrade the broker to new versions as AWS releases them, choose Enable automatic minor version upgrades. Automatic upgrades occur during the maintenance window defined by the day of the week, the time of day (in 24-hour format), and the time zone (UTC by default).

6. Choose Schedule modifications.

   Note

   If you choose only Enable automatic minor version upgrades, the button changes to Save because no broker reboot is necessary.

   Your preferences are applied to your broker at the specified time.

Using Python Pika with Amazon MQ for RabbitMQ

The following tutorial shows how you can set up a Python Pika client with TLS configured to connect to an Amazon MQ for RabbitMQ broker. Pika is a Python implementation of the AMQP 0-9-1 protocol for RabbitMQ. This tutorial guides you through installing Pika, declaring a queue, setting up a publisher to send messages to the broker's default exchange, and setting up a consumer to receive messages from the queue.
Topics

- Prerequisites (p. 61)
- Permissions (p. 61)
- Step one: Create a basic Python Pika client (p. 61)
- Step two: Create a publisher and send a message (p. 62)
- Step three: Create a consumer and receive a message (p. 63)
- Step four: (Optional) Set up an event loop and consume messages (p. 64)
- What's next? (p. 65)

Prerequisites

To complete the steps in this tutorial, you need the following prerequisites:

- An Amazon MQ for RabbitMQ broker. For more information, see Creating an Amazon MQ for RabbitMQ broker (p. 11).
- Python 3 installed for your operating system.
- Pika installed using Python `pip`. To install Pika, open a new terminal window and run the following.

```
$ python3 -m pip install pika
```

Permissions

For this tutorial, you need at least one Amazon MQ for RabbitMQ broker user with permission to write to, and read from, a vhost. The following table describes the necessary minimum permissions as regular expression (regexp) patterns.

<table>
<thead>
<tr>
<th>Tags</th>
<th>Configure regexp</th>
<th>Write regexp</th>
<th>Read regexp</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>.*</td>
<td>.*</td>
<td>.*</td>
</tr>
</tbody>
</table>

The user permissions listed provide only read and write permissions to the user, without granting access to the management plugin to perform administrative operations on the broker. You can further restrict permissions by providing regexp patterns that limit the user’s access to specified queues. For example, if you change the read regexp pattern to `^[hello world].*`, the user will only have permission to read from queues that start with `hello world`.

For more information about creating RabbitMQ users and managing user tags and permissions, see User (p. 130).

Step one: Create a basic Python Pika client

To create a Python Pika client base class that defines a constructor and provides the SSL context necessary for TLS configuration when interacting with an Amazon MQ for RabbitMQ broker, do the following.

1. Open a new terminal window, create a new directory for your project, and navigate to the directory.

```
$ mkdir pika-tutorial
$ cd pika-tutorial
```
2. Create a new file, `basicClient.py`, that contains the following Python code.

```python
textarea
class BasicPikaClient:
    def __init__(self, rabbitmq_broker_id, rabbitmq_user, rabbitmq_password, region):
        # SSL Context for TLS configuration of Amazon MQ for RabbitMQ
        ssl_context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2)
        ssl_context.set_ciphers('ECDHE+AESGCM:!ECDSA')

        url = f'amqps://{rabbitmq_user}:{rabbitmq_password}@{rabbitmq_broker_id}.mq.{region}.amazonaws.com:5671'
        parameters = pika.URLParameters(url)
        parameters.ssl_options = pika.SSLOptions(context=ssl_context)

        self.connection = pika.BlockingConnection(parameters)
        self.channel = self.connection.channel()
```

You can now define additional classes for your publisher and consumer that inherit from `BasicPikaClient`.

**Step two: Create a publisher and send a message**

To create a publisher that declares a queue, and sends a single message, do the following.

1. Copy the contents of the following code sample, and save locally as `publisher.py` in the same directory you created in the previous step.

```python
from basicClient import BasicPikaClient
class BasicMessageSender(BasicPikaClient):
    def declare_queue(self, queue_name):
        print(f"Trying to declare queue({queue_name})...")
        self.channel.queue_declare(queue=queue_name)

    def send_message(self, exchange, routing_key, body):
        channel = self.connection.channel()
        channel.basic_publish(exchange=exchange,
                              routing_key=routing_key,
                              body=body)
        print(f"Sent message. Exchange: {exchange}, Routing Key: {routing_key}, Body: {body}")

    def close(self):
        self.channel.close()
        self.connection.close()

if __name__ == "__main__":
    # Initialize Basic Message Sender which creates a connection
    # and channel for sending messages.
    basic_message_sender = BasicMessageSender("<broker-id>",
                                              "<username>",
                                              "<password>",
                                              "<region>"
    )
```
```python
# Declare a queue
basic_message_sender.declare_queue("hello world queue")

# Send a message to the queue.
basic_message_sender.send_message(exchange="", routing_key="hello world queue", body=b'Hello World!')

# Close connections.
basic_message_sender.close()
```

The `BasicMessageSender` class inherits from `BasicPikaClient` and implements additional methods for declaring a queue, sending a message to the queue, and closing connections. The code sample routes a message to the default exchange, with a routing key equal to the name of the queue.

2. Under `if __name__ == '__main__':`, replace the parameters passed to the `BasicMessageSender` constructor statement with the following information.

   - `<broker-id>` – The unique ID that Amazon MQ generates for the broker. You can parse the ID from your broker ARN. For example, given the following ARN, `arn:aws:mq:us-east-2:123456789012:broker:MyBroker:b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9`, the broker ID would be `b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9`.
   - `<username>` – The username for a broker user with sufficient permissions to write messages to the broker.
   - `<password>` – The password for a broker user with sufficient permissions to write messages to the broker.
   - `<region>` – The AWS region in which you created your Amazon MQ for RabbitMQ broker. For example, `us-west-2`.

3. Run the following command in the same directory you created `publisher.py`.

   ```sh
   python3 publisher.py
   ```

   If the code runs successfully, you will see the following output in your terminal window.

   ```
   Trying to declare queue(hello world queue)...  
   Sent message. Exchange: , Routing Key: hello world queue, Body: b'Hello World!'  
   ```

**Step three: Create a consumer and receive a message**

To create a consumer that receives a single message from the queue, do the following.

1. Copy the contents of the following code sample, and save locally as `consumer.py` in the same directory.

   ```python
   from basicClient import BasicPikaClient

   class BasicMessageReceiver(BasicPikaClient):
       def get_message(self, queue):
           method_frame, header_frame, body = self.channel.basic_get(queue)
           if method_frame:
               print(method_frame, header_frame, body)
               self.channel.basic_ack(method_frame.delivery_tag)
               return method_frame, header_frame, body
           else:
               print('No message returned')
   ```
def close(self):
    self.channel.close()
    self.connection.close()

if __name__ == '__main__':
    # Create Basic Message Receiver which creates a connection
    # and channel for consuming messages.
    basic_message_receiver = BasicMessageReceiver(
        "<broker-id>",
        "<username>",
        "<password>",
        "<region>
    )

    # Consume the message that was sent.
    basic_message_receiver.get_message("hello world queue")

    # Close connections.
    basic_message_receiver.close()

Similar to the the publisher you created in the previous step, BasicMessageReciever inherits from BasicPikaClient and implements additional methods for receiving a single message, and closing connections.

2. Under the `if __name__ == '__main__':` statement, replace the parameters passed to the BasicMessageReceiver constructor with your information.

3. Run the following command in your project directory.

```
$ python3 consumer.py
```

If the code runs successfully, you will see the message body, and headers including the routing key, displayed in your terminal window.

```
<Basic.GetOk(['delivery_tag=1', 'exchange=', 'message_count=0', 'redelivered=False', 'routing_key=hello world queue'])> <BasicProperties> b'Hello World!'
```

**Step four: (Optional) Set up an event loop and consume messages**

To consume multiple messages from a queue, use Pika's `basic_consume` method and a callback function as shown in the following

1. In `consumer.py`, add the following method definition to the BasicMessageReceiver class.

```python
def consume_messages(self, queue):
    def callback(ch, method, properties, body):
        print("[x] Received %r" % body)
        self.channel.basic_consume(queue=queue, on_message_callback=callback,
        auto_ack=True)

        print('[*] Waiting for messages. To exit press CTRL+C')
        self.channel.start_consuming()
```

2. In `consumer.py`, under `if __name__ == '__main__':`, invoke the `consume_messages` method you defined in the previous step.
if __name__ == "__main__":

    # Create Basic Message Receiver which creates a connection and channel for consuming messages.
    basic_message_receiver = BasicMessageReceiver(
        "<broker-id>",
        "<username>",
        "<password>",
        "<region>
    )

    # Consume the message that was sent.
    # basic_message_receiver.get_message("hello world queue")

    # Consume multiple messages in an event loop.
    basic_message_receiver.consume_messages("hello world queue")

    # Close connections.
    basic_message_receiver.close()

3. Run consumer.py again, and if successful, the queued messages will be displayed in your terminal window.

    [*] Waiting for messages. To exit press CTRL+C
    [x] Received b'Hello World!'
    [x] Received b'Hello World!'
    ...

**What's next?**

- For more information about other supported RabbitMQ client libraries, see RabbitMQ Client Documentation on the RabbitMQ website.

## Resolving RabbitMQ paused queue synchronization

In an Amazon MQ for RabbitMQ cluster deployment (p. 133), messages published to each queue are replicated across three broker nodes. This replication, referred to as *mirroring*, provides high availability (HA) for RabbitMQ brokers. Queues in a cluster deployment consist of a main replica on one node and one or more mirrors. Every operation applied to a mirrored queue, including enqueuing messages, is first applied to the main queue and then replicated across its mirrors.

For example, consider a mirrored queue replicated across three nodes: the main node (main) and two mirrors (mirror-1 and mirror-2). If all messages in this mirrored queue are successfully propagated to all mirrors, then the queue is synchronized. If a node (mirror-1) becomes unavailable for an interval of time, the queue is still operational and can continue to enqueue messages. However, for the queue to synchronize, messages published to main while mirror-1 is unavailable must be replicated to mirror-1.

For more information about mirroring, see *Classic Mirrored Queues* on the RabbitMQ website.

### Maintenance and queue synchronization

During maintenance windows (p. 60), Amazon MQ performs all maintenance work one node at a time to ensure that the broker remains operational. As a result, queues might need to synchronize as each node resumes operation. During synchronization, messages that need to be replicated to mirrors are loaded into memory from the corresponding Amazon Elastic Block Store (Amazon EBS) volume to be processed in batches. Processing messages in batches lets queues synchronize faster.
If queues are kept short and messages are small, the queues successfully synchronize and resume operation as expected. However, if the amount of data in a batch approaches the node's memory limit, the node raises a high memory alarm, pausing the queue sync. You can confirm memory usage by comparing the RabbitMemUsed and RabbitMqMemLimit broker node metrics in CloudWatch (p. 168). Synchronization can't complete until messages are consumed or deleted, or the number of messages in the batch is reduced.

**Note**
Reducing the queue synchronization batch size can result in a higher number of replication transactions.

To resolve a paused queue synchronization, follow the steps in this tutorial, which demonstrates applying an `ha-sync-batch-size` policy and restarting the queue sync.

**Topics**
- Prerequisites (p. 66)
- Step 1: Apply an `ha-sync-batch-size` policy (p. 66)
- Step 2: Restart the queue sync (p. 68)
- Next steps (p. 68)
- Related resources (p. 68)

**Prerequisites**
For this tutorial, you must have an Amazon MQ for RabbitMQ broker user with administrator permissions. You can use the administrator user created when you first created the broker, or another user that you might have created afterwards. The following table provides the necessary administrator user tag and permissions as regular expression (regexp) patterns.

<table>
<thead>
<tr>
<th>Tags</th>
<th>Read regexp</th>
<th>Configure regexp</th>
<th>Write regexp</th>
</tr>
</thead>
<tbody>
<tr>
<td>administrator</td>
<td>.*</td>
<td>.*</td>
<td>.*</td>
</tr>
</tbody>
</table>

For more information about creating RabbitMQ users and managing user tags and permissions, see User (p. 130).

**Step 1: Apply an `ha-sync-batch-size` policy**

The following procedures demonstrate adding a policy that applies to all queues created on the broker. You can use the RabbitMQ web console or the RabbitMQ management API. For more information, see Management Plugin on the RabbitMQ website.

**To apply an `ha-sync-batch-size` policy using the RabbitMQ web console**

1. Sign in to the Amazon MQ console.
2. In the left navigation pane, choose Brokers.
3. From the list of brokers, choose the name of the broker to which you want to apply the new policy.
4. On the broker's page, in the Connections section, choose the RabbitMQ web console URL. The RabbitMQ web console opens in a new browser tab or window.
5. Log in to the RabbitMQ web console with your broker administrator user name and password.
6. In the RabbitMQ web console, at the top of the page, choose Admin.
7. On the Admin page, in the right navigation pane, choose Policies.
8. On the Policies page, you can see a list of the broker's current User policies. Below User policies, expand Add / update a policy.
Note
By default, Amazon MQ for RabbitMQ clusters are created with an initial broker policy named `ha-all-AWS-OWNED-DO-NOT-DELETE`. Amazon MQ manages this policy to ensure that every queue on the broker is replicated to all three nodes and that queues are synchronized automatically.

9. To create a new broker policy, under Add / update a policy, do the following:
   a. For Name, enter a name for your policy, for example, `batch-size-policy`.
   b. For Pattern, enter the regexp pattern `.*` so that the policy matches all queues on the broker.
   c. For Apply to, choose Exchanges and queues from the dropdown list.
   d. For Priority, enter an integer greater than all other policies in applied to the vhost. You can apply exactly one set of policy definitions to RabbitMQ queues and exchanges at any given time. RabbitMQ chooses the matching policy with the highest priority value. For more information about policy priorities and how to combine policies, see Policies in the RabbitMQ Server Documentation.
   e. For Definition, add the following key-value pairs:
      - `ha-sync-batch-size=100`. Choose Number from the dropdown list.
        Note
        You might need to adjust and calibrate the value of `ha-sync-batch-size` based on the number and size of unsynchronized messages in your queues.
      - `ha-mode=all`. Choose String from the dropdown list.
        Important
        The `ha-mode` definition is required for all HA-related policies. Omitting it results in a validation failure.
      - `ha-sync-mode=automatic`. Choose String from the dropdown list.
        Note
        The `ha-sync-mode` definition is required for all custom policies. If it is omitted, Amazon MQ automatically appends the definition.
   f. Choose Add / update policy.

10. Confirm that the new policy appears in the list of User policies.

To apply an ha-sync-batch-size policy using the RabbitMQ management API

1. Sign in to the Amazon MQ console.
2. In the left navigation pane, choose Brokers.
3. From the list of brokers, choose the name of the broker to which you want to apply the new policy.
4. On the broker's page, in the Connections section, note the RabbitMQ web console URL. This is the broker endpoint that you use in an HTTP request.
5. Open a new terminal or command line window of your choice.
6. To create a new broker policy, enter the following `curl` command. This command assumes a queue on the default / vhost, which is encoded as `%2F`.
   Note
   Replace `username` and `password` with your broker administrator user name and password. You might need to adjust and calibrate the value of `ha-sync-batch-size` based on the number and size of unsynchronized messages in your queues. Replace the broker endpoint with the URL that you noted previously.

   ```
curl -i -u username:password -H "content-type:application/json" -XPUT -d '{"pattern": ".*", "priority":1, "definition":{"ha-sync-batch-size":100, "ha-mode":"all", "ha-sync-mode":"automatic"}}'
```
7. To confirm that the new policy is added to your broker's user policies, enter the following `curl` command to list all broker policies.

```
curl -i -u username:password https://b-589c045f-f8ln-4ab0-a89c-co62e1c32ef8.mq.us-west-2.amazonaws.com/api/policies
```

### Step 2: Restart the queue sync

After applying a new `ha-sync-batch-size` policy to your broker, restart the queue sync.

**To restart the queue sync using the RabbitMQ web console**

*Note*
To open the RabbitMQ web console, see the previous instructions in Step 1 of this tutorial.

1. In the RabbitMQ web console, at the top of the page, choose **Queues**.
2. On the **Queues** page, under **All queues**, locate your paused queue. In the **Features** column, your queue should list the name of the new policy that you created (for example, `batch-size-policy`).
3. To restart the synchronization process with a reduced batch size, choose **Restart sync**.

*Note*
If synchronization pauses and doesn't finish successfully, try reducing the `ha-sync-batch-size` value and restarting the queue sync again.

### Next steps

- Once your queue synchronizes successfully, you can monitor the amount of memory that your RabbitMQ nodes use by viewing the Amazon CloudWatch metric `RabbitMQMemUsed`. You can also view the `RabbitMQMemLimit` metric to monitor a node's memory limit. For more information, see Accessing CloudWatch metrics for Amazon MQ (p. 168) and Logging and monitoring Amazon MQ for RabbitMQ brokers (p. 176).
- To prevent paused queue synchronization, we recommend keeping queues short and processing messages. For workloads with larger message sizes, we also recommend upgrading your broker instance type to a larger instance size with more memory. For more information about broker instance types and editing broker preferences, see Amazon MQ for RabbitMQ instance types (p. 139) and Editing broker preferences (p. 60).
- When you create a new Amazon MQ for RabbitMQ broker, Amazon MQ applies a set of default policies and virtual host limits to optimize broker performance. If your broker does not have the recommended default policies and limits, we recommend creating them yourself. For more information about creating default policies and vhost limits, see the section called “Broker defaults” (p. 125).

### Related resources

- **UpdateBrokerInput** – Use this broker property to update a broker instance type using the Amazon MQ API.
- **Parameters and Policies** (RabbitMQ Server Documentation) – Learn more about RabbitMQ parameters and policies on the RabbitMQ website.
- **RabbitMQ Management HTTP API** – Learn more about the RabbitMQ management API.
Best practices for Amazon MQ

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

Topics
- Amazon MQ for ActiveMQ best practices (p. 69)
- Amazon MQ for RabbitMQ best practices (p. 74)

Amazon MQ for ActiveMQ best practices

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

Topics
- Connecting to Amazon MQ (p. 69)
- Ensuring effective Amazon MQ performance (p. 71)
- Avoid slow restarts by recovering prepared XA transactions (p. 73)

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 (p. 69)
- Always Use Connection Pooling (p. 70)
- Always Use the Failover Transport to Connect to Multiple Broker Endpoints (p. 71)
- Avoid Using Message Selectors (p. 71)
- Prefer Virtual Destinations to Durable Subscriptions (p. 71)
- If using Amazon VPC peering, avoid client IPs in CIDR range 10.0.0.0/16 (p. 71)

Never Modify or Delete the Amazon MQ Elastic Network Interface

When you first create an Amazon MQ broker (p. 30), 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. 156). The network interface allows your client (producer or consumer) to communicate with the Amazon MQ broker. The network interface is considered to be within the service scope of Amazon MQ, despite being part of your account's VPC.

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

In a scenario with a single producer and single consumer (such as the Getting Started with Amazon MQ (p. 4) tutorial), you can use a single `ActiveMQConnectionFactory` class for every producer and consumer. For example:

```java
// 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.

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

// Pass the username and password.
connectionFactory.setUserName(activeMqUsername);
connectionFactory.setPassword(activeMqPassword);
```
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 (p. 30) deployment mode or when you migrate from an on-premises message broker to Amazon MQ—use the Failover Transport to allow your consumers to randomly connect to either one. For example:

```
failover:(ssl://b-1234a5b6-78cd-901e-2fgh-3i45j6k17819-1.mq.us-east-2.amazonaws.com:61617,ssl://b-9876l5k4-32ji-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.

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

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

Ensuring effective Amazon MQ performance

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

Topics

- Disable Concurrent Store and Dispatch for Queues with Slow Consumers (p. 72)
- Choose the Correct Broker Instance Type for the Best Throughput (p. 72)
- Choose the correct broker storage type for the best throughput (p. 73)
Ensure effective Amazon MQ performance

Configure Your Network of Brokers Correctly (p. 73)

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. 113).

Choose the Correct Broker Instance Type for the Best Throughput

The message throughput of a broker instance type (p. 136) depends on your application's use case and the following factors:

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

Understanding the relationship between message size, latency, and throughput

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

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

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

To determine the best broker instance type for your application, we recommend testing different broker instance types. For more information, see Broker instance types (p. 136) and also Measuring the Throughput for Amazon MQ using the JMS Benchmark.

Use cases for larger broker instance types

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

- **Non-persistent mode** – When your application is less sensitive to losing messages during broker instance failover (p. 85) (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. 113) flag is enabled, ActiveMQ allows messages to flow directly from producer to consumer without sending messages to storage (even in persistent mode). If your application can consume messages quickly (or if you can design your consumers to do this), your application can benefit from a larger broker instance type. To let your application consume messages more quickly, add consumer threads to your application instances or scale up your application instances vertically or horizontally.

- **Batched transactions** – When you use persistent mode and send multiple messages per transaction, you can achieve an overall higher message throughput by using larger broker instance types. For more information, see Should I Use Transactions? in the ActiveMQ documentation.

### Choose the correct broker storage type for the best throughput

To take advantage of high durability and replication across multiple Availability Zones, use Amazon EFS. To take advantage of low latency and high throughput, use Amazon EBS. For more information, see Storage (p. 82).

### Configure Your Network of Brokers Correctly

When you create a network of brokers (p. 86), configure it correctly for your application:

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

  If persistent mode is disabled, the first broker acknowledges the producer without persisting the message to storage. For more information, see Replicated Message Store and What is the difference between persistent and non-persistent delivery? in the Apache ActiveMQ documentation.

- **Don't disable advisory messages for broker instances** – For more information, see Advisory Message in the Apache ActiveMQ documentation.

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

### Avoid slow restarts by recovering prepared XA transactions

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

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

To monitor your unresolved prepared XA transactions, you can use the `JournalFilesForFastRecovery` metric in Amazon CloudWatch Logs. If this number is increasing, or
is consistently higher than 1, you should recover your unresolved transactions with code similar to the following example. For more information, see Quotas in Amazon MQ (p. 188).

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

```java
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) {
            // Handle exception
        }
    }
}
```

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().

Amazon MQ for RabbitMQ best practices

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

Topics

- Enable lazy queues (p. 75)
- Use persistent and durable queues (p. 75)
- Keep queues short (p. 75)
- Configure acknowledgement and confirmation (p. 76)
- Configure pre-fetching (p. 77)
- Automatically recover from network failures (p. 77)
Enable lazy queues

If you are working with very long queues that process large volumes of messages, enabling lazy queues can improve your broker's overall performance.

RabbitMQ's default behavior is to cache messages in memory and to move them to disk only when the broker needs more available memory. This process of moving messages from memory to disk can take time and stops the queue from processing messages. Enabling lazy queues can have a significant impact on speeding up the process of moving messages to disk as lazy queues store messages to disk as soon as possible, resulting in fewer messages cached in memory.

You can enable lazy queues by setting the queue.declare arguments at the time of declaration, or by configuring a policy via the RabbitMQ management console. The following example demonstrates declaring a lazy queue using the RabbitMQ Java client library.

```java
Map<String, Object> args = new HashMap<String, Object>();
args.put("x-queue-mode", "lazy");
channel.queueDeclare("myqueue", false, false, false, args);
```

**Note**

Enabling lazy queues can increase disk I/O operations.

Use persistent and durable queues

Persistent messages can help prevent data loss in situations where a broker crashes or restarts. Persistent messages are written to disk as soon as they arrive. Unlike lazy queues, however, persistent messages are cached both in memory and in disk unless more memory is needed by the broker. In cases where more memory is needed, messages are removed from memory by the RabbitMQ broker mechanism that manages storing messages to disk, commonly referred to as the persistence layer.

To enable message persistence, you can declare your queues as durable and set message delivery mode to persistent. The following example demonstrates using the RabbitMQ Java client library to declare a durable queue.

```java
boolean durable = true;
channel.queueDeclare("my_queue", durable, false, false, null);
```

Once you have configured your queue as durable, you can send a persistent message to your queue by setting MessageProperties to PERSISTENT_TEXT_PLAIN as shown in the following example.

```java
import com.rabbitmq.client.MessageProperties;

channel.basicPublish("", "my_queue",
    MessageProperties.PERSISTENT_TEXT_PLAIN,
    message.getBytes());
```

Keep queues short

In cluster deployments, queues with a large number of messages can lead to resource overutilization. When a broker is overutilized, rebooting an Amazon MQ for RabbitMQ broker can cause further degradation of performance. If rebooted, overutilized brokers might become unresponsive in the REBOOT_IN_PROGRESS state.

During maintenance windows (p. 60), Amazon MQ performs all maintenance work one node at a time to ensure that the broker remains operational. As a result, queues might need to synchronize as each node
resumes operation. During synchronization, messages that need to be replicated to mirrors are loaded into memory from the corresponding Amazon Elastic Block Store (Amazon EBS) volume to be processed in batches. Processing messages in batches lets queues synchronize faster.

If queues are kept short and messages are small, the queues successfully synchronize and resume operation as expected. However, if the amount of data in a batch approaches the node's memory limit, the node raises a high memory alarm, pausing the queue sync. You can confirm memory usage by comparing the RabbitMemUsed and RabbitMqMemLimit broker node metrics in CloudWatch (p. 168). Synchronization can't complete until messages are consumed or deleted, or the number of messages in the batch is reduced.

If queue synchronization is paused for a cluster deployment, we recommend consuming or deleting messages to lower the number of messages in queues. Once queue depth is reduced and queue sync completes, the broker status will change to RUNNING. To resolve a paused queue sync, you can also apply a policy to reduce the queue synchronization batch-size (p. 65).

Warning
Do not reboot a broker that is running high on resources. If you reboot a broker when queue synchronization is paused, the broker will reinitiate the synchronization process, which can further degrade broker resources as messages are transferred from storage to node memory, and result in the broker becoming unresponsive in the REBOOT_IN_PROGRESS state.

Configure acknowledgement and confirmation

When a client application sends confirmation of delivery and consumption of messages back to the broker, it is known as consumer acknowledgment. Similarly, the process of sending confirmation to a publisher is known as publisher confirm. Both acknowledgement and confirmation are essential to ensuring data safety when working with RabbitMQ brokers.

Consumer delivery acknowledgement is typically configured on the client application. When working with AMQP 0-9-1, acknowledgement can be enabled by configuring the basic.consume or when a message is fetched using the basic.code method.

Typically, delivery acknowledgement is enabled in a channel. For example, when working with the RabbitMQ Java client library, you can use the Channel#basicAck to set up a simple basic.ack positive acknowledgement as shown in the following example.

```java
// this example assumes an existing channel instance

boolean autoAck = false;
channel.basicConsume(queueName, autoAck, "a-consumer-tag",
    new DefaultConsumer(channel) {
        @Override
        public void handleDelivery(String consumerTag,
            Envelope envelope,
            AMQP.BasicProperties properties,
            byte[] body)
            throws IOException
        {
            long deliveryTag = envelope.getDeliveryTag();
            // positively acknowledge a single delivery, the message will
            // be discarded
            channel.basicAck(deliveryTag, false);
        }
    });
```

Note
Unacknowledged messages must be cached in memory. You can limit the number of messages that a consumer pre-fetches by configuring pre-fetch (p. 77) settings for a client application.
Configure pre-fetching

You can use the RabbitMQ pre-fetch value to optimize how your consumers consume messages. RabbitMQ implements the channel pre-fetch mechanism provided by AMQP 0-9-1 by applying the pre-fetch count to consumers as opposed to channels. The pre-fetch value is used to specify how many messages are being sent to the consumer at any given time. By default, RabbitMQ sets an unlimited buffer size for client applications.

There are a variety of factors to consider when setting a pre-fetch count for your RabbitMQ consumers. First, consider your consumers' environment and configuration. Because consumers need to keep all messages in memory as they are being processed, a high pre-fetch value can have a negative impact on your consumers' performance, and in some cases, can result in a consumer potentially crashing all together. Similarly, the RabbitMQ broker itself keeps all messages that it sends cached in memory until it receives consumer acknowledgement. A high pre-fetch value can cause your RabbitMQ server to run out of memory quickly if automatic acknowledgement is not configured for consumers, and if consumers take a relatively long time to process messages.

With the above considerations in mind, we recommend always setting a pre-fetch value in order to prevent situations where a RabbitMQ broker or its consumers run out of memory due to a large number of unprocessed, or unacknowledged messages. If you need to optimize your brokers to process large volumes of messages, you can test your brokers and consumers using a range of pre-fetch counts to determine the value at which point network overhead becomes largely insignificant compared to the time it takes a consumer to process messages.

Note

- If your client applications have configured to automatically acknowledge delivery of messages to consumers, setting a pre-fetch value will have no effect.
- All pre-fetched messages are removed from the queue.

The following example demonstrate setting a pre-fetch value of 10 for a single consumer using the RabbitMQ Java client library.

```java
ConnectionFactory factory = new ConnectionFactory();
Connection connection = factory.newConnection();
Channel channel = connection.createChannel();
channel.basicQos(10, false);
QueueingConsumer consumer = new QueueingConsumer(channel);
channel.basicConsume("my_queue", false, consumer);
```

Note

In the RabbitMQ Java client library, the default value for the `global` flag is set to `false`, so the above example can be written simply as `channel.basicQos(10)`.

Automatically recover from network failures

We recommend always enabling automatic network recovery to prevent significant downtime in cases where client connections to RabbitMQ nodes fail. The RabbitMQ Java client library supports automatic network recovery by default, beginning with version 4.0.0.

Automatic connection recovery is triggered if an unhandled exception is thrown in the connection's I/O loop, if a socket read operation timeout is detected, or if the server misses a heartbeat.

In cases where the initial connection between a client and a RabbitMQ node fails, automatic recovery will not be triggered. We recommend writing your application code to account for initial connection failures.
by retrying the connection. The following example demonstrates retrying initial network failures using the RabbitMQ Java client library.

```java
ConnectionFactory factory = new ConnectionFactory();
// enable automatic recovery if using RabbitMQ Java client library prior to version 4.0.0.
factory.setAutomaticRecoveryEnabled(true);
// configure various connection settings
try {
    Connection conn = factory.newConnection();
} catch (java.net.ConnectException e) {
    Thread.sleep(5000);
    // apply retry logic
}
```

**Note**
If an application closes a connection by using the Connection.Close method, automatic network recovery will not be enabled or triggered.
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 for ActiveMQ and RabbitMQ engine types, lists available Amazon MQ broker instance types and their statuses, and provides an overview of broker architecture and configuration options.

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

Topics

- ActiveMQ engine (p. 79)
- RabbitMQ engine (p. 124)
- Instance types (p. 136)
- Broker statuses (p. 140)
- Tagging resources (p. 140)

ActiveMQ engine

This section describes the basic elements of an ActiveMQ broker, provides an overview of ActiveMQ broker architecture options, explains broker configuration parameters, and offers a working example using Java Message Service (JMS).

Topics

- Basic elements (p. 79)
- Broker architecture (p. 83)
- ActiveMQ broker configuration parameters (p. 97)
- Managing Amazon MQ for ActiveMQ engine versions (p. 115)
- Working examples of using Java Message Service (JMS) with ActiveMQ (p. 116)

Basic elements

This section introduces key concepts essential to understanding ActiveMQ on Amazon MQ.

Topics

- Broker (p. 79)
- Configuration (p. 81)
- User (p. 82)
- Storage (p. 82)

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, t3) and size (large, micro)
is a *broker instance type* (for example, *mq.m5.large*). For more information, see Broker instance types (p. 136).

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

For more information, see Broker Architecture (p. 83).

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. 30)
- Brokers (p. 188)
- Broker statuses (p. 140)

**Supported wire-level protocols**

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

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

**Attributes**

An ActiveMQ broker has several attributes, for example:

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

- `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

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. 42) or reboot the broker (p. 27). For more information, see Amazon MQ Broker Configuration Lifecycle (p. 96). Currently, you can't delete a configuration.

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

- Creating and applying broker configurations (p. 38)
- Editing and Managing Broker Configurations (p. 40)
- Configurations (p. 188)
- Amazon MQ Broker Configuration Parameters (p. 97)

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. 38) and Editing and Managing Broker Configurations (p. 40).

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

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. 42) or reboot the broker (p. 27). For more information, see Amazon MQ Broker Configuration Lifecycle (p. 96).

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 ActiveMQ broker users (p. 58)
• Users (p. 189)

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

Storage

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

- You can use Amazon EBS only with the `mq.m5` broker instance type family.
- Although you can change the `broker instance type`, you can't change the `broker storage type` after you create the broker.
- Amazon EBS replicates data within a single Availability Zone and doesn't support the ActiveMQ active/standby (p. 85) deployment mode.

Differences between Storage Types

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

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

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

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

Broker architecture

Amazon MQ for ActiveMQ 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.

**Note**

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

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

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

Topics
- Amazon MQ single-instance broker (p. 84)
- Amazon MQ active/standby broker for high availability (p. 85)
- Amazon MQ Network of brokers (p. 86)
- Amazon MQ broker configuration lifecycle (p. 96)

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 Amazon EBS or Amazon EFS storage volume. Amazon EFS storage volumes are designed to provide the highest level of durability and availability by storing data redundantly across multiple Availability Zones (AZs). Amazon EBS provides block level storage optimized for low-latency and high throughput. For more information about storage options, see Storage (p. 82).

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

The following diagram illustrates a single-instance broker with Amazon EBS storage replicated across multiple servers within a single AZ.
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 Amazon EFS. Amazon EFS storage volumes are designed to provide the highest level of durability, and availability by storing data redundantly across multiple Availability Zones (AZs). For more information, see Storage (p. 82).

Usually, only one of the broker instances is active at any time, while the other broker instance is on standby. If one of the broker instances malfunctions or undergoes maintenance, it takes Amazon MQ a short while to take the inactive instance out of service. This allows the healthy standby instance to become active and to begin accepting incoming communications. 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 with Amazon EFS storage replicated across multiple AZs.
Amazon MQ supports ActiveMQ’s network of brokers feature.

A network of brokers is comprised of multiple simultaneously active single-instance brokers (p. 84) or active/standby brokers (p. 85). You can configure networks of brokers in a variety of topologies (p. 88) (for example, concentrator, hub-and-spokes, tree, or mesh), depending on your application’s needs, such as high availability and scalability. For instance, a hub and spoke (p. 91) network of brokers can increase resiliency, preserving messages if one broker is not reachable. A network of brokers with a concentrator (p. 92) topology can collect messages from a larger number of brokers accepting incoming messages, and concentrate them to more central brokers, to better handle the load of many incoming messages.

For a tutorial and detailed configuration information, see the following:

- Creating and Configuring a Network of Brokers (p. 35)
- Configure Your Network of Brokers Correctly (p. 73)
- networkConnector (p. 112)
- networkConnectionStartAsync (p. 108)
- Networks of Brokers in the ActiveMQ documentation

The following are benefits of using a network of brokers:

- Creating a network of brokers allows you to increase your aggregate throughput and maximum producer and consumer connection count by adding broker instances.
• You can ensure better availability by allowing your producers and consumers to be aware of multiple active broker instances. This allows them to reconnect to a new instance if the one they’re currently connected to becomes unavailable.

• Because producers and consumers can reconnect to another node in the network of brokers immediately, and because there’s no need to wait for a standby broker instance to become promoted, client reconnection within a network of brokers is faster than for an active/standby broker for high availability (p. 85).

How does a network of brokers work?

Amazon MQ supports the ActiveMQ network of brokers feature in a number of ways. First, you can edit the parameters within each broker's configuration to create a network of brokers, just as you would with native ActiveMQ. Second, Amazon MQ has sample blueprints that use AWS CloudFormation to automate the creation of a network of brokers. You can deploy these sample blueprints directly from the Amazon MQ console, or you can edit the related AWS CloudFormation templates to create your own topologies and configurations.

A network of brokers is established by connecting one broker to another using network connectors. Once connected, these brokers provide message forwarding. For instance, if Broker1 establishes a network connector to Broker2, messages on Broker1 are forwarded to Broker2 if there is a consumer on that broker for the queue or topic. If the network connector is configured as duplex, messages are also forwarded from Broker2 to Broker1. Network connectors are configured in the broker configuration. See, Configuration (p. 81). For instance, here is an example networkConnector entry in a broker configuration:

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

A network of brokers ensures that messages flow from one broker instance to another, forwarding messages only to the broker instances that have corresponding consumers. For the benefit of broker instances adjacent to each other within the network, ActiveMQ sends messages to advisory topics about producers and consumers connecting to and disconnecting from the network. When a broker instance receives information about a producer that consumes from a particular destination, the broker instance begins to forward messages. For more information, see Advisory Topics in the ActiveMQ documentation.

How Does a Network of Brokers Handle Credentials?

For broker A to connect to broker B in a network, broker A must use valid credentials, like any other producer or consumer. Instead of providing a password in broker A's <networkConnector> configuration, you must first create a user on broker A with the same values as another user on broker B (these are separate, unique users that share the same username and password values). When you specify the userName attribute in the <networkConnector> configuration, Amazon MQ will add the password automatically at runtime.
Important
Don't specify the password attribute for the `<networkConnector>`. We don't recommend storing plaintext passwords in broker configuration files, because this makes the passwords visible in the Amazon MQ console. For more information, see Configure Network Connectors for Your Broker (p. 36).

Brokers must be in the same VPC or in peered VPCs. For more information, see Prerequisites (p. 35) in the Creating and Configuring a Network of Brokers (p. 35) tutorial.

Sample blueprints

To get started using a Network of Brokers, Amazon MQ provides sample blueprints. These sample blueprints create a Network of Brokers deployment, and all related resources using, AWS CloudFormation. The two sample blueprints available are:

1. Mesh network of single instance brokers
2. Mesh network of active/standby brokers

From the Create brokers page, select one of the sample blueprints and choose Next. Once the resources have been created, review the generated brokers and their configurations in the Amazon MQ console.

By creating brokers and configuring different networkConnector elements in the broker configurations, you can create a network of brokers in many different topologies. For more information on configuring a network of brokers, see Networks of Brokers in the ActiveMQ documentation.

Network of brokers topologies

By deploying brokers, and then configuring networkConnector entries in their configurations, you can build a network of brokers using different network topologies. A network connector provides on-demand message forwarding between connected brokers. Connections can be configured as duplex, where messages are forwarded both ways between brokers, or not duplex, where the forwarding only propagates from one broker to the other. For example, if we have a duplex connection between Broker1 and Broker2, messages will be forwarded from each to the other if there is a consumer.
With a duplex network connector, messages are forwarded from each broker to the other. These are forwarded *on-demand*: if there is a consumer on Broker2 for a message on Broker1, the message is forwarded. Similarly, if there is a consumer on Broker1 for a message on Broker2 the message is also forwarded.

For non-duplex connections, messages are forwarded only from one broker to the other. In this example, if there is a consumer on Broker2 for a message on Broker1, the message is forwarded. But messages will not be forwarded from Broker2 to Broker1.

Using both duplex and non-duplex network connectors, it is possible to build a network of brokers in any number of network topologies.

**Note**

In each of the network topology examples, the `networkConnector` elements reference the endpoint of the brokers they connect to. Replace the broker endpoint entries in the `uri` attributes with the endpoints of your brokers. See, Listing brokers and viewing broker details (p. 19).

**Mesh topology**

A mesh topology provides multiple brokers that are all connected to each other. This simple example connects three single-instance brokers, but you can configure more brokers as a mesh.
This topology, and one that includes a mesh of active/standby pairs of brokers, can be created using sample blueprints in the Amazon MQ console. You can create these sample blueprint deployment to see a working network of brokers, and review how they are configured.

You can configure a three broker mesh network like this by adding a network connector to Broker1 that makes duplex connections to both Broker2 and Broker3, and a single duplex connection between Broker2 and Broker3.

**Network connectors for Broker1:**

```xml
<networkConnectors>
    <networkConnector name="connector_1_to_2" userName="myCommonUser" duplex="true"
        uri="static:(ssl://b-987615k4-32ji-109h-8gte-7d65c4b132a1-2.mq.us-east-2.amazonaws.com:61617)"/>
    <networkConnector name="connector_1_to_3" userName="myCommonUser" duplex="true"
        uri="static:(ssl://b-743c885d-2244-4c95-af67-a85017ff234e-3.mq.us-east-2.amazonaws.com:61617)"/>
</networkConnectors>
```

**Network connectors for Broker2:**

```xml
<networkConnectors>
    <networkConnector name="connector_2_to_3" userName="myCommonUser" duplex="true"
        uri="static:(ssl://b-743c885d-2244-4c95-af67-a85017ff234e-3.mq.us-east-2.amazonaws.com:61617)"/>
</networkConnectors>
```

By adding the above connectors to the configurations of Broker1 and Broker2, you can create a mesh between these three brokers that forwards message between all the brokers on demand. For more information, see Amazon MQ Broker Configuration Parameters (p. 97).
Hub and Spoke Topology

In a hub and spoke topology, messages are preserved if there is a disruption to any broker on a spoke. Messages are forwarded throughout, and only the central Broker 1 is critical to the network’s operation.

To configure the hub and spoke network of brokers in this example, you could add a `<networkConnector>` to each of the brokers on the spokes in the configuration of Broker 1.

```xml
<networkConnectors>
    <networkConnector name="connector_hub_and_spoke_2" userName="myCommonUser" duplex="true"
        uri="static:(ssl://b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-east-2.amazonaws.com:61617)"/>
    <networkConnector name="connector_hub_and_spoke_3" userName="myCommonUser"
        duplex="true"
```
Concentrator topology

In this example topology, the three brokers on the bottom can handle a large number of connections, and those messages are concentrated to Broker 1 and Broker 2. Each of the other brokers has a non-duplex connection to the more central brokers. To scale the capacity of this topology, you can add additional brokers that receive messages and concentrate those messages in Broker 1 and Broker 2.

To configure this topology, each of the brokers on the bottom would contain a network connector to each of the brokers they are concentrating messages to.

Network connectors for Broker 3:

```
<networkConnectors>
  <networkConnector name="3_to_1" userName="myCommonUser" duplex="false"
    uri="static:(ssl://b-123456b-78cd-901e-2fg-h-3i45j6k178l9-1.mq.us-
    east-2.amazonaws.com:61617)"/>
  <networkConnector name="3_to_2" userName="myCommonUser" duplex="false"
    uri="static:(ssl://b-9876i5k4-32ji-109h-8gfe-7d65c4b132a1-2.mq.us-
    east-2.amazonaws.com:61617)"/>
</networkConnectors>
```
Network connectors for Broker4:

```xml
<networkConnectors>
  <networkConnector name="4_to_1" userName="myCommonUser" duplex="false"
      uri="static:(ssl://b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-east-2.amazonaws.com:61617)"/>
  <networkConnector name="4_to_2" userName="myCommonUser" duplex="false"
      uri="static:(ssl://b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-east-2.amazonaws.com:61617)"/>
</networkConnectors>
```

Network connectors for Broker5:

```xml
<networkConnectors>
  <networkConnector name="5_to_1" userName="myCommonUser" duplex="false"
      uri="static:(ssl://b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-east-2.amazonaws.com:61617)"/>
  <networkConnector name="5_to_2" userName="myCommonUser" duplex="false"
      uri="static:(ssl://b-9876l5k4-32ji-109h-8gfe-7d65c4b132a1-2.mq.us-east-2.amazonaws.com:61617)"/>
</networkConnectors>
```

Cross region

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

**Network connectors for Broker1:**

```xml
<networkConnectors>
  <networkConnector name="1_to_2" userName="myCommonUser" duplex="true"
       uri="static:(ssl://b-987615k4-32ji-109h-8gefe-7d65c4b132a1-2.mq.us-
       east-2.amazonaws.com:61617)"/>
  <networkConnector name="1_to_3" userName="myCommonUser" duplex="true"
       uri="static:(ssl://b-743c885d-2244-4c95-af67-a85017ff234e-3.mq.us-
       east-2.amazonaws.com:61617)"/>
</networkConnectors>
```
Network connector for Broker2:

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

Network connectors for Broker4:

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

Dynamic Failover With Transport Connectors

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

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

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

Available options:

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

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

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

When a new broker connects, it will receive a list of URIs of all brokers in the network. If the connection to the broker fails, it can dynamically switch to one of the brokers provided when it connected.
For more information on failover, see Broker-side Options for Failover in the Active MQ documentation.

**Amazon MQ 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. 42) or reboot the broker (p. 27). For more information, see Amazon MQ Broker Configuration Lifecycle (p. 96).

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. 38)
- Editing and Managing Broker Configurations (p. 40)
- Amazon MQ Broker Configuration Parameters (p. 97)

For information about creating, editing, and deleting ActiveMQ users, see the following:
ActiveMQ 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. 81)
- Creating and applying broker configurations (p. 38)
- Editing and Managing Broker Configurations (p. 40)
- Configurations (p. 188)

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 schemas. Download zip files of the supported schemas using the following links.

- amazon-mq-active-mq-5.16.3.xsd.zip
- amazon-mq-active-mq-5.16.2.xsd.zip
- amazon-mq-active-mq-5.15.15.xsd.zip
- amazon-mq-active-mq-5.15.14.xsd.zip
- amazon-mq-active-mq-5.15.13.xsd.zip
- amazon-mq-active-mq-5.15.12.xsd.zip
- amazon-mq-active-mq-5.15.10.xsd.zip
- amazon-mq-active-mq-5.15.9.xsd.zip
- amazon-mq-active-mq-5.15.8.xsd.zip
- amazon-mq-active-mq-5.15.6.xsd.zip
- amazon-mq-active-mq-5.15.0.xsd.zip

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

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

Topics
- Elements Permitted in Amazon MQ Configurations (p. 98)
- Elements and Their Attributes Permitted in Amazon MQ Configurations (p. 100)
- Elements, Child Collection Elements, and Their Child Elements Permitted in Amazon MQ Configurations (p. 108)
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.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abortSlowAckConsumerStrategy</td>
<td>(attributes)</td>
</tr>
<tr>
<td>abortSlowConsumerStrategy</td>
<td>(attributes)</td>
</tr>
<tr>
<td>authorizationEntry</td>
<td>(attributes)</td>
</tr>
<tr>
<td>authorizationMap</td>
<td>(child collection elements)</td>
</tr>
<tr>
<td>authorizationPlugin</td>
<td>(child collection elements)</td>
</tr>
<tr>
<td>broker</td>
<td>(attributes)</td>
</tr>
<tr>
<td>cachedMessageGroupMapFactory</td>
<td>(attributes)</td>
</tr>
<tr>
<td>compositeQueue</td>
<td>(attributes)</td>
</tr>
<tr>
<td>compositeTopic</td>
<td>(attributes)</td>
</tr>
<tr>
<td>constantPendingMessageLimitStrategy</td>
<td>(attributes)</td>
</tr>
<tr>
<td>discarding</td>
<td>(attributes)</td>
</tr>
<tr>
<td>discardingDLQBrokerPlugin</td>
<td>(attributes)</td>
</tr>
<tr>
<td>fileCursor</td>
<td></td>
</tr>
<tr>
<td>fileDurableSubscriberCursor</td>
<td></td>
</tr>
<tr>
<td>fileQueueCursor</td>
<td></td>
</tr>
<tr>
<td>filteredDestination</td>
<td>(attributes)</td>
</tr>
<tr>
<td>fixedCountSubscriptionRecoveryPolicy</td>
<td>(attributes)</td>
</tr>
<tr>
<td>fixedSizedSubscriptionRecoveryPolicy</td>
<td>(attributes)</td>
</tr>
<tr>
<td>forcePersistencyModeBrokerPlugin</td>
<td>(attributes)</td>
</tr>
<tr>
<td>individualDeadLetterStrategy</td>
<td>(attributes)</td>
</tr>
<tr>
<td>lastImageSubscriptionRecoveryPolicy</td>
<td></td>
</tr>
<tr>
<td>messageGroupHashBucketFactory</td>
<td>(attributes)</td>
</tr>
<tr>
<td>mirroredQueue</td>
<td>(attributes)</td>
</tr>
<tr>
<td>noSubscriptionRecoveryPolicy</td>
<td></td>
</tr>
<tr>
<td>oldestMessageEvictionStrategy</td>
<td>(attributes)</td>
</tr>
<tr>
<td>oldestMessageWithLowestPriorityEvictionStrategy</td>
<td>(attributes)</td>
</tr>
<tr>
<td>policyEntry</td>
<td>(attributes)</td>
</tr>
<tr>
<td>policyMap</td>
<td>(child collection elements)</td>
</tr>
<tr>
<td>Element</td>
<td></td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>prefetchRatePendingMessageLimitStrategy <em>(attributes)</em> <em>(p. 105)</em></td>
<td></td>
</tr>
<tr>
<td>priorityDispatchPolicy</td>
<td></td>
</tr>
<tr>
<td>priorityNetworkDispatchPolicy</td>
<td></td>
</tr>
<tr>
<td>queryBasedSubscriptionRecoveryPolicy <em>(attributes)</em> <em>(p. 105)</em></td>
<td></td>
</tr>
<tr>
<td>queue <em>(attributes)</em> <em>(p. 105)</em></td>
<td></td>
</tr>
<tr>
<td>redeliveryPlugin <em>(attributes</em> <em>(p. 105)</em></td>
<td>child collection elements* <em>(p. 110)</em></td>
</tr>
<tr>
<td>redeliveryPolicy <em>(attributes)</em> <em>(p. 105)</em></td>
<td></td>
</tr>
<tr>
<td>redeliveryPolicyMap <em>(child collection elements)</em> <em>(p. 110)</em></td>
<td></td>
</tr>
<tr>
<td>retainedMessageSubscriptionRecoveryPolicy <em>(child collection elements)</em> <em>(p. 110)</em></td>
<td></td>
</tr>
<tr>
<td>roundRobinDispatchPolicy</td>
<td></td>
</tr>
<tr>
<td>sharedDeadLetterStrategy <em>(attributes</em> <em>(p. 106)</em></td>
<td>child collection elements* <em>(p. 111)</em></td>
</tr>
<tr>
<td>simpleDispatchPolicy</td>
<td></td>
</tr>
<tr>
<td>simpleMessageGroupMapFactory</td>
<td></td>
</tr>
<tr>
<td>statisticsBrokerPlugin</td>
<td></td>
</tr>
<tr>
<td>storeCursor</td>
<td></td>
</tr>
<tr>
<td>storeDurableSubscriberCursor <em>(attributes)</em> <em>(p. 106)</em></td>
<td></td>
</tr>
<tr>
<td>strictOrderDispatchPolicy</td>
<td></td>
</tr>
<tr>
<td>tempDestinationAuthorizationEntry <em>(attributes)</em> <em>(p. 106)</em></td>
<td></td>
</tr>
<tr>
<td>tempQueue <em>(attributes)</em> <em>(p. 106)</em></td>
<td></td>
</tr>
<tr>
<td>tempTopic <em>(attributes)</em> <em>(p. 106)</em></td>
<td></td>
</tr>
<tr>
<td>timedSubscriptionRecoveryPolicy <em>(attributes)</em> <em>(p. 106)</em></td>
<td></td>
</tr>
<tr>
<td>timeStampingBrokerPlugin</td>
<td></td>
</tr>
<tr>
<td>topic <em>(attributes)</em> <em>(p. 106)</em></td>
<td></td>
</tr>
<tr>
<td>transportConnector <em>(attributes)</em> <em>(p. 107)</em></td>
<td></td>
</tr>
<tr>
<td>uniquePropertyMessageEvictionStrategy <em>(attributes)</em> <em>(p. 107)</em></td>
<td></td>
</tr>
<tr>
<td>virtualDestinationInterceptor <em>(child collection elements)</em> <em>(p. 111)</em></td>
<td></td>
</tr>
<tr>
<td>virtualTopic <em>(attributes)</em> <em>(p. 107)</em></td>
<td></td>
</tr>
<tr>
<td>vmCursor</td>
<td></td>
</tr>
<tr>
<td>vmDurableCursor</td>
<td></td>
</tr>
<tr>
<td>vmQueueCursor</td>
<td></td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Element</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>abortSlowAckConsumerStrategy</td>
<td>abortConnection</td>
</tr>
<tr>
<td></td>
<td>checkPeriod</td>
</tr>
<tr>
<td></td>
<td>ignoreIdleConsumers</td>
</tr>
<tr>
<td></td>
<td>ignoreNetworkConsumers</td>
</tr>
<tr>
<td></td>
<td>maxSlowCount</td>
</tr>
<tr>
<td></td>
<td>maxSlowDuration</td>
</tr>
<tr>
<td></td>
<td>maxTimeSinceLastAck</td>
</tr>
<tr>
<td></td>
<td>name</td>
</tr>
<tr>
<td>abortSlowConsumerStrategy</td>
<td>abortConnection</td>
</tr>
<tr>
<td></td>
<td>checkPeriod</td>
</tr>
<tr>
<td></td>
<td>ignoreNetworkConsumers</td>
</tr>
<tr>
<td></td>
<td>maxSlowCount</td>
</tr>
<tr>
<td></td>
<td>maxSlowDuration</td>
</tr>
<tr>
<td></td>
<td>name</td>
</tr>
<tr>
<td>authorizationEntry</td>
<td>admin</td>
</tr>
<tr>
<td></td>
<td>queue</td>
</tr>
<tr>
<td></td>
<td>read</td>
</tr>
<tr>
<td></td>
<td>tempQueue</td>
</tr>
<tr>
<td></td>
<td>tempTopic</td>
</tr>
<tr>
<td></td>
<td>topic</td>
</tr>
<tr>
<td></td>
<td>write</td>
</tr>
<tr>
<td>broker</td>
<td>advisorySupport</td>
</tr>
<tr>
<td></td>
<td>allowTempAutoCreationOnSend</td>
</tr>
<tr>
<td></td>
<td>cacheTempDestinations</td>
</tr>
<tr>
<td></td>
<td>consumerSystemUsagePortion</td>
</tr>
<tr>
<td></td>
<td>dedicatedTaskRunner</td>
</tr>
<tr>
<td></td>
<td>deleteAllMessagesOnStartup</td>
</tr>
</tbody>
</table>
## Broker configuration parameters

<table>
<thead>
<tr>
<th>Element</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>keepDurableSubsActive</td>
<td>keepDurableSubsActive</td>
</tr>
<tr>
<td>enableMessageExpirationOnActiveDurableSubs</td>
<td>enableMessageExpirationOnActiveDurableSubs</td>
</tr>
<tr>
<td><strong>Supported in</strong></td>
<td><strong>Supported in</strong></td>
</tr>
<tr>
<td>Apache ActiveMQ 15.16.x and above</td>
<td>Apache ActiveMQ 15.16.x and above</td>
</tr>
<tr>
<td>maxPurgedDestinationsPerSweep</td>
<td>maxPurgedDestinationsPerSweep</td>
</tr>
<tr>
<td>maxSchedulerRepeatAllowed</td>
<td>maxSchedulerRepeatAllowed</td>
</tr>
<tr>
<td><strong>Supported in</strong></td>
<td><strong>Supported in</strong></td>
</tr>
<tr>
<td>Apache ActiveMQ 15.16.x and above</td>
<td>Apache ActiveMQ 15.16.x and above</td>
</tr>
<tr>
<td>monitorConnectionSplits</td>
<td>monitorConnectionSplits</td>
</tr>
<tr>
<td>networkConnectorStartAsync</td>
<td>networkConnectorStartAsync (p. 108)</td>
</tr>
<tr>
<td>offlineDurableSubscriberTaskSchedule</td>
<td>offlineDurableSubscriberTaskSchedule</td>
</tr>
<tr>
<td>offlineDurableSubscriberTimeout</td>
<td>offlineDurableSubscriberTimeout</td>
</tr>
<tr>
<td>persistenceThreadPriority</td>
<td>persistenceThreadPriority</td>
</tr>
<tr>
<td>persistent</td>
<td>persistent</td>
</tr>
<tr>
<td>populateJMSXUserID</td>
<td>populateJMSXUserID</td>
</tr>
<tr>
<td>producerSystemUsagePortion</td>
<td>producerSystemUsagePortion</td>
</tr>
<tr>
<td>rejectDurableConsumers</td>
<td>rejectDurableConsumers</td>
</tr>
<tr>
<td>rollbackOnlyOnAsyncException</td>
<td>rollbackOnlyOnAsyncException</td>
</tr>
<tr>
<td>schedulePeriodForDestinationPurge</td>
<td>schedulePeriodForDestinationPurge</td>
</tr>
<tr>
<td>schedulerSupport</td>
<td>schedulerSupport</td>
</tr>
<tr>
<td>splitSystemUsageForProducersConsumers</td>
<td>splitSystemUsageForProducersConsumers</td>
</tr>
<tr>
<td>taskRunnerPriority</td>
<td>taskRunnerPriority</td>
</tr>
<tr>
<td>timeBeforePurgeTempDestinations</td>
<td>timeBeforePurgeTempDestinations</td>
</tr>
<tr>
<td>useAuthenticatedPrincipalForJMSXUserID</td>
<td>useAuthenticatedPrincipalForJMSXUserID</td>
</tr>
<tr>
<td>useMirroredQueues</td>
<td>useMirroredQueues</td>
</tr>
<tr>
<td>useTempMirroredQueues</td>
<td>useTempMirroredQueues</td>
</tr>
<tr>
<td>useVirtualDestSubs</td>
<td>useVirtualDestSubs</td>
</tr>
<tr>
<td>useVirtualDestSubsOnCreation</td>
<td>useVirtualDestSubsOnCreation</td>
</tr>
<tr>
<td>useVirtualTopics</td>
<td>useVirtualTopics</td>
</tr>
<tr>
<td>cachedMessageGroupMapFactory</td>
<td>cacheSize</td>
</tr>
<tr>
<td>compositeQueue</td>
<td>concurrentSend</td>
</tr>
<tr>
<td></td>
<td>copyMessage</td>
</tr>
<tr>
<td>Element</td>
<td>Attribute</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td></td>
<td>forwardOnly</td>
</tr>
<tr>
<td></td>
<td>name</td>
</tr>
<tr>
<td></td>
<td>sendWhenNotMatched</td>
</tr>
<tr>
<td><strong>Supported in</strong></td>
<td>Apache ActiveMQ 15.16.x and above</td>
</tr>
<tr>
<td>compositeTopic</td>
<td>concurrentSend</td>
</tr>
<tr>
<td></td>
<td>copyMessage</td>
</tr>
<tr>
<td></td>
<td>forwardOnly</td>
</tr>
<tr>
<td></td>
<td>name</td>
</tr>
<tr>
<td></td>
<td>sendWhenNotMatched</td>
</tr>
<tr>
<td><strong>Supported in</strong></td>
<td>Apache ActiveMQ 15.16.x and above</td>
</tr>
<tr>
<td>conditionalNetworkBridgeFilterFactory</td>
<td>rateDuration</td>
</tr>
<tr>
<td></td>
<td>rateLimit</td>
</tr>
<tr>
<td></td>
<td>replayDelay</td>
</tr>
<tr>
<td></td>
<td>replayWhenNoConsumers</td>
</tr>
<tr>
<td></td>
<td>selectorAware</td>
</tr>
<tr>
<td><strong>Supported in</strong></td>
<td>Apache ActiveMQ 15.16.x</td>
</tr>
<tr>
<td>constantPendingMessageLimitStrategy</td>
<td>limit</td>
</tr>
<tr>
<td>discarding</td>
<td>deadLetterQueue</td>
</tr>
<tr>
<td></td>
<td>enableAudit</td>
</tr>
<tr>
<td></td>
<td>expiration</td>
</tr>
<tr>
<td></td>
<td>maxAuditDepth</td>
</tr>
<tr>
<td></td>
<td>maxProducersToAudit</td>
</tr>
<tr>
<td></td>
<td>processExpired</td>
</tr>
<tr>
<td></td>
<td>processNonPersistent</td>
</tr>
<tr>
<td>discardingDLQBrokerPlugin</td>
<td>dropAll</td>
</tr>
<tr>
<td></td>
<td>dropOnly</td>
</tr>
<tr>
<td></td>
<td>dropTemporaryQueues</td>
</tr>
<tr>
<td></td>
<td>dropTemporaryTopics</td>
</tr>
<tr>
<td></td>
<td>reportInterval</td>
</tr>
<tr>
<td>filteredDestination</td>
<td>queue</td>
</tr>
</tbody>
</table>
## Broker configuration parameters

<table>
<thead>
<tr>
<th>Element</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>selector</td>
<td></td>
</tr>
<tr>
<td>topic</td>
<td></td>
</tr>
<tr>
<td>fixedCountSubscriptionRecoveryPolicy</td>
<td>maximumSize</td>
</tr>
<tr>
<td>fixedSizedSubscriptionRecoveryPolicy</td>
<td>maximumSize</td>
</tr>
<tr>
<td></td>
<td>useSharedBuffer</td>
</tr>
<tr>
<td>forcePersistencyModeBrokerPlugin</td>
<td>persistenceFlag</td>
</tr>
<tr>
<td>individualDeadLetterStrategy</td>
<td>destinationPerDurableSubscriber</td>
</tr>
<tr>
<td></td>
<td>enableAudit</td>
</tr>
<tr>
<td></td>
<td>expiration</td>
</tr>
<tr>
<td></td>
<td>maxAuditDepth</td>
</tr>
<tr>
<td></td>
<td>maxProducersToAudit</td>
</tr>
<tr>
<td></td>
<td>processExpired</td>
</tr>
<tr>
<td></td>
<td>processNonPersistent</td>
</tr>
<tr>
<td></td>
<td>queuePrefix</td>
</tr>
<tr>
<td></td>
<td>queueSuffix</td>
</tr>
<tr>
<td></td>
<td>topicPrefix</td>
</tr>
<tr>
<td></td>
<td>topicSuffix</td>
</tr>
<tr>
<td></td>
<td>useQueueForQueueMessages</td>
</tr>
<tr>
<td></td>
<td>useQueueForTopicMessages</td>
</tr>
<tr>
<td>messageGroupHashBucketFactory</td>
<td>bucketCount</td>
</tr>
<tr>
<td></td>
<td>cacheSize</td>
</tr>
<tr>
<td>mirroredQueue</td>
<td>copyMessage</td>
</tr>
<tr>
<td></td>
<td>postfix</td>
</tr>
<tr>
<td></td>
<td>prefix</td>
</tr>
<tr>
<td>oldestMessageEvictionStrategy</td>
<td>evictExpiredMessagesHighWatermark</td>
</tr>
<tr>
<td>oldestMessageWithLowestPriorityEvictionStrategy</td>
<td>evictExpiredMessagesHighWatermark</td>
</tr>
<tr>
<td>policyEntry</td>
<td>advisoryForConsumed</td>
</tr>
<tr>
<td></td>
<td>advisoryForDelivery</td>
</tr>
<tr>
<td></td>
<td>advisoryForDiscardingMessages</td>
</tr>
<tr>
<td></td>
<td>advisoryForFastProducers</td>
</tr>
<tr>
<td></td>
<td>advisoryForSlowConsumers</td>
</tr>
<tr>
<td>Element</td>
<td>Attribute</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>advisoryWhenFull</td>
<td></td>
</tr>
<tr>
<td>allConsumersExclusiveByDefault</td>
<td></td>
</tr>
<tr>
<td>alwaysRetroactive</td>
<td></td>
</tr>
<tr>
<td>blockedProducerWarningInterval</td>
<td></td>
</tr>
<tr>
<td>consumersBeforeDispatchStarts</td>
<td></td>
</tr>
<tr>
<td>cursorMemoryHighWaterMark</td>
<td></td>
</tr>
<tr>
<td>doOptimizeMessageStorage</td>
<td></td>
</tr>
<tr>
<td>durableTopicPrefetch</td>
<td></td>
</tr>
<tr>
<td>enableAudit</td>
<td></td>
</tr>
<tr>
<td>expireMessagesPeriod</td>
<td></td>
</tr>
<tr>
<td>gcInactiveDestinations</td>
<td></td>
</tr>
<tr>
<td>gcWithNetworkConsumers</td>
<td></td>
</tr>
<tr>
<td>inactiveTimeoutBeforeGC</td>
<td></td>
</tr>
<tr>
<td>inactiveTimeoutBeforeGC</td>
<td></td>
</tr>
<tr>
<td>includeBodyForAdvisory</td>
<td></td>
</tr>
<tr>
<td>lazyDispatch</td>
<td></td>
</tr>
<tr>
<td>maxAuditDepth</td>
<td></td>
</tr>
<tr>
<td>maxBrowsePageSize</td>
<td></td>
</tr>
<tr>
<td>maxDestinations</td>
<td></td>
</tr>
<tr>
<td>maxExpirePageSize</td>
<td></td>
</tr>
<tr>
<td>maxPageSize</td>
<td></td>
</tr>
<tr>
<td>maxProducersToAudit</td>
<td></td>
</tr>
<tr>
<td>maxQueueAuditDepth</td>
<td></td>
</tr>
<tr>
<td>memoryLimit</td>
<td></td>
</tr>
<tr>
<td>messageGroupMapFactoryType</td>
<td></td>
</tr>
<tr>
<td>minimumMessageSize</td>
<td></td>
</tr>
<tr>
<td>optimizedDispatch</td>
<td></td>
</tr>
<tr>
<td>optimizeMessageStoreInFlightLimit</td>
<td></td>
</tr>
<tr>
<td>persistJMSRedelivered</td>
<td></td>
</tr>
<tr>
<td>prioritizedMessages</td>
<td></td>
</tr>
<tr>
<td>producerFlowControl</td>
<td></td>
</tr>
</tbody>
</table>

---

104
## Broker configuration parameters

<table>
<thead>
<tr>
<th>Element</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue</td>
<td></td>
</tr>
<tr>
<td></td>
<td>queueBrowserPrefetch</td>
</tr>
<tr>
<td></td>
<td>queuePrefetch</td>
</tr>
<tr>
<td></td>
<td>reduceMemoryFootprint</td>
</tr>
<tr>
<td></td>
<td>sendAdvisoryIfNoConsumers</td>
</tr>
<tr>
<td></td>
<td>sendFailIfNoSpace</td>
</tr>
<tr>
<td></td>
<td>Supported in</td>
</tr>
<tr>
<td></td>
<td>Apache ActiveMQ 15.16.x and above</td>
</tr>
<tr>
<td></td>
<td>sendFailIfNoSpaceAfterTimeout</td>
</tr>
<tr>
<td></td>
<td>storeUsageHighWaterMark</td>
</tr>
<tr>
<td></td>
<td>strictOrderDispatch</td>
</tr>
<tr>
<td></td>
<td>tempQueue</td>
</tr>
<tr>
<td></td>
<td>tempTopic</td>
</tr>
<tr>
<td></td>
<td>timeBeforeDispatchStarts</td>
</tr>
<tr>
<td></td>
<td>topic</td>
</tr>
<tr>
<td></td>
<td>topicPrefetch</td>
</tr>
<tr>
<td></td>
<td>useCache</td>
</tr>
<tr>
<td></td>
<td>useConsumerPriority</td>
</tr>
<tr>
<td></td>
<td>usePrefetchExtension</td>
</tr>
<tr>
<td></td>
<td>prefetchRatePendingMessageLimitStrategy</td>
</tr>
<tr>
<td></td>
<td>multiplier</td>
</tr>
<tr>
<td></td>
<td>queryBasedSubscriptionRecoveryPolicy</td>
</tr>
<tr>
<td></td>
<td>query</td>
</tr>
<tr>
<td></td>
<td>queue</td>
</tr>
<tr>
<td></td>
<td>DLQ</td>
</tr>
<tr>
<td></td>
<td>physicalName</td>
</tr>
<tr>
<td></td>
<td>redeliveryPlugin</td>
</tr>
<tr>
<td></td>
<td>fallbackToDeadLetter</td>
</tr>
<tr>
<td></td>
<td>sendToDlqIfMaxRetriesExceeded</td>
</tr>
<tr>
<td></td>
<td>redeliveryPolicy</td>
</tr>
<tr>
<td></td>
<td>backOffMultiplier</td>
</tr>
<tr>
<td></td>
<td>collisionAvoidancePercent</td>
</tr>
<tr>
<td></td>
<td>initialRedeliveryDelay</td>
</tr>
<tr>
<td></td>
<td>maximumRedeliveries</td>
</tr>
<tr>
<td></td>
<td>maximumRedeliveryDelay</td>
</tr>
<tr>
<td></td>
<td>preDispatchCheck</td>
</tr>
<tr>
<td></td>
<td>queue</td>
</tr>
<tr>
<td>Element</td>
<td>Attribute</td>
</tr>
<tr>
<td>---------</td>
<td>-----------</td>
</tr>
<tr>
<td></td>
<td>redeliveryDelay</td>
</tr>
<tr>
<td></td>
<td>tempQueue</td>
</tr>
<tr>
<td></td>
<td>tempTopic</td>
</tr>
<tr>
<td></td>
<td>topic</td>
</tr>
<tr>
<td></td>
<td>useCollisionAvoidance</td>
</tr>
<tr>
<td></td>
<td>useExponentialBackOff</td>
</tr>
<tr>
<td>sharedDeadLetterStrategy</td>
<td>enableAudit</td>
</tr>
<tr>
<td></td>
<td>expiration</td>
</tr>
<tr>
<td></td>
<td>maxAuditDepth</td>
</tr>
<tr>
<td></td>
<td>maxProducersToAudit</td>
</tr>
<tr>
<td></td>
<td>processExpired</td>
</tr>
<tr>
<td></td>
<td>processNonPersistent</td>
</tr>
<tr>
<td>storeDurableSubscriberCursor</td>
<td>immediatePriorityDispatch</td>
</tr>
<tr>
<td></td>
<td>useCache</td>
</tr>
<tr>
<td>tempDestinationAuthorizationEntry</td>
<td>admin</td>
</tr>
<tr>
<td></td>
<td>queue</td>
</tr>
<tr>
<td></td>
<td>read</td>
</tr>
<tr>
<td></td>
<td>tempQueue</td>
</tr>
<tr>
<td></td>
<td>tempTopic</td>
</tr>
<tr>
<td></td>
<td>topic</td>
</tr>
<tr>
<td></td>
<td>write</td>
</tr>
<tr>
<td>tempQueue</td>
<td>DLQ</td>
</tr>
<tr>
<td></td>
<td>physicalName</td>
</tr>
<tr>
<td>tempTopic</td>
<td>DLQ</td>
</tr>
<tr>
<td></td>
<td>physicalName</td>
</tr>
<tr>
<td>timedSubscriptionRecoveryPolicy</td>
<td>zeroExpirationOverride</td>
</tr>
<tr>
<td>timeStamchingBrokerPlugin</td>
<td>recoverDuration</td>
</tr>
<tr>
<td></td>
<td>futureOnly</td>
</tr>
<tr>
<td></td>
<td>processNetworkMessages</td>
</tr>
<tr>
<td></td>
<td>ttlCeiling</td>
</tr>
<tr>
<td>topic</td>
<td>DLQ</td>
</tr>
</tbody>
</table>
### Amazon MQ Parent Element Attributes

The following is a detailed explanation of parent element attributes. For more information, see [XML Configuration](https://activemq.apache.org/configuration.html) in the Apache ActiveMQ documentation.

**Topics**
- broker (p. 107)

**broker**

broker is a parent collection element.
Attributes

**networkConnectionStartAsync**

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

**Default:** false

Example Configuration

```xml
<broker networkConnectionStartAsync="false"/>
```

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

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

<table>
<thead>
<tr>
<th>Element</th>
<th>Child Collection Element</th>
<th>Child Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>authorizationMap</td>
<td>authorizationEntries</td>
<td>authorizationEntry (p. 111)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tempDestinationAuthorizationEntry</td>
</tr>
<tr>
<td></td>
<td>defaultEntry</td>
<td>authorizationEntry</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tempDestinationAuthorizationEntry</td>
</tr>
<tr>
<td></td>
<td>tempDestinationAuthorizationEntry</td>
<td></td>
</tr>
<tr>
<td>authorizationPlugin</td>
<td>map</td>
<td>authorizationMap</td>
</tr>
<tr>
<td>broker</td>
<td>destinationInterceptors</td>
<td>mirroredQueue</td>
</tr>
<tr>
<td></td>
<td></td>
<td>virtualDestinationInterceptor</td>
</tr>
<tr>
<td></td>
<td>destinationPolicy</td>
<td>policyMap</td>
</tr>
<tr>
<td></td>
<td>destinations</td>
<td>queue</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tempQueue</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tempTopic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>topic</td>
</tr>
<tr>
<td></td>
<td>networkConnectors</td>
<td>networkConnector (p. 112)</td>
</tr>
<tr>
<td>persistenceAdapter</td>
<td>kahaDB (p. 113)</td>
<td></td>
</tr>
<tr>
<td>plugins</td>
<td>authorizationPlugin</td>
<td></td>
</tr>
<tr>
<td></td>
<td>discardingDLQBrokerPlugin</td>
<td></td>
</tr>
<tr>
<td></td>
<td>forcePersistencyModeBrokerPlugin</td>
<td></td>
</tr>
<tr>
<td></td>
<td>redeliveryPlugin</td>
<td></td>
</tr>
<tr>
<td>Element</td>
<td>Child Collection Element</td>
<td>Child Element</td>
</tr>
<tr>
<td>--------------------------</td>
<td>--------------------------</td>
<td>---------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>statisticsBrokerPlugin</td>
</tr>
<tr>
<td></td>
<td></td>
<td>timeStampingBrokerPlugin</td>
</tr>
<tr>
<td>systemUsage</td>
<td></td>
<td>systemUsage (p. 114)</td>
</tr>
<tr>
<td>transportConnector</td>
<td></td>
<td>name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>updateClusterClients</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rebalanceClusterClients</td>
</tr>
<tr>
<td></td>
<td></td>
<td>updateClusterClientsOnRemove</td>
</tr>
<tr>
<td>compositeQueue</td>
<td>forwardingTo</td>
<td>queue</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tempQueue</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tempTopic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>topic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>filteredDestination</td>
</tr>
<tr>
<td>compositeTopic</td>
<td>forwardingTo</td>
<td>queue</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tempQueue</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tempTopic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>topic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>filteredDestination</td>
</tr>
<tr>
<td>policyEntry</td>
<td>deadLetterStrategy</td>
<td>discarding</td>
</tr>
<tr>
<td></td>
<td></td>
<td>individualDeadLetterStrategy</td>
</tr>
<tr>
<td></td>
<td></td>
<td>sharedDeadLetterStrategy</td>
</tr>
<tr>
<td></td>
<td>destination</td>
<td>queue</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tempQueue</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tempTopic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>topic</td>
</tr>
<tr>
<td>dispatchPolicy</td>
<td></td>
<td>priorityDispatchPolicy</td>
</tr>
<tr>
<td></td>
<td></td>
<td>priorityNetworkDispatchPolicy</td>
</tr>
<tr>
<td></td>
<td></td>
<td>roundRobinDispatchPolicy</td>
</tr>
<tr>
<td></td>
<td></td>
<td>simpleDispatchPolicy</td>
</tr>
<tr>
<td></td>
<td></td>
<td>strictOrderDispatchPolicy</td>
</tr>
<tr>
<td></td>
<td></td>
<td>clientIdFilterDispatchPolicy</td>
</tr>
<tr>
<td>messageEvictionStrategy</td>
<td></td>
<td>oldestMessageEvictionStrategy</td>
</tr>
</tbody>
</table>
### Broker configuration parameters

<table>
<thead>
<tr>
<th>Element</th>
<th>Child Collection Element</th>
<th>Child Element</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>oldestMessageWithLowestPriorityEvictionStrategy</td>
<td></td>
</tr>
<tr>
<td></td>
<td>uniquePropertyMessageEvictionStrategy</td>
<td></td>
</tr>
<tr>
<td>messageGroupMapFactory</td>
<td>cachedMessageGroupMapFactory</td>
<td></td>
</tr>
<tr>
<td></td>
<td>messageGroupHashBucketFactory</td>
<td></td>
</tr>
<tr>
<td></td>
<td>simpleMessageGroupMapFactory</td>
<td></td>
</tr>
<tr>
<td></td>
<td>fileDurableSubscriberCursor</td>
<td></td>
</tr>
<tr>
<td></td>
<td>storeDurableSubscriberCursor</td>
<td></td>
</tr>
<tr>
<td></td>
<td>vmDurableCursor</td>
<td></td>
</tr>
<tr>
<td></td>
<td>pendingMessageLimitStrategy</td>
<td>constantPendingMessageLimitStrategy</td>
</tr>
<tr>
<td></td>
<td></td>
<td>prefetchRatePendingMessageLimitStrategy</td>
</tr>
<tr>
<td>pendingDurableSubscriberPolicy</td>
<td>fileQueueCursor</td>
<td></td>
</tr>
<tr>
<td></td>
<td>storeCursor</td>
<td></td>
</tr>
<tr>
<td></td>
<td>vmQueueCursor</td>
<td></td>
</tr>
<tr>
<td>pendingSubscriberPolicy</td>
<td>fileCursor</td>
<td></td>
</tr>
<tr>
<td></td>
<td>vmCursor</td>
<td></td>
</tr>
<tr>
<td>slowConsumerStrategy</td>
<td>abortSlowAckConsumerStrategy</td>
<td></td>
</tr>
<tr>
<td></td>
<td>abortSlowConsumerStrategy</td>
<td></td>
</tr>
<tr>
<td>subscriptionRecoveryPolicy</td>
<td>fixedCountSubscriptionRecoveryPolicy</td>
<td></td>
</tr>
<tr>
<td></td>
<td>fixedSizedSubscriptionRecoveryPolicy</td>
<td></td>
</tr>
<tr>
<td></td>
<td>lastImageSubscriptionRecoveryPolicy</td>
<td></td>
</tr>
<tr>
<td></td>
<td>noSubscriptionRecoveryPolicy</td>
<td></td>
</tr>
<tr>
<td></td>
<td>queryBasedSubscriptionRecoveryPolicy</td>
<td></td>
</tr>
<tr>
<td></td>
<td>retainedMessageSubscriptionRecoveryPolicy</td>
<td></td>
</tr>
<tr>
<td>timedSubscriptionRecoveryPolicy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>policyMap</td>
<td>defaultEntry</td>
<td>policyEntry</td>
</tr>
<tr>
<td></td>
<td>policyEntries</td>
<td>policyEntry</td>
</tr>
<tr>
<td>redeliveryPlugin</td>
<td>redeliveryPolicyMap</td>
<td>redeliveryPolicyMap</td>
</tr>
<tr>
<td>redeliveryPolicyMap</td>
<td>defaultEntry</td>
<td>redeliveryPolicy</td>
</tr>
<tr>
<td></td>
<td>redeliveryPolicyEntries</td>
<td>redeliveryPolicy</td>
</tr>
<tr>
<td>retainedMessageSubscriptionRecoveryPolicy</td>
<td></td>
<td>fixedCountSubscriptionRecoveryPolicy</td>
</tr>
<tr>
<td></td>
<td></td>
<td>fixedSizedSubscriptionRecoveryPolicy</td>
</tr>
</tbody>
</table>
## Broker configuration parameters

<table>
<thead>
<tr>
<th>Element</th>
<th>Child Collection Element</th>
<th>Child Element</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>lastImageSubscriptionRecoveryPolicy</td>
</tr>
<tr>
<td></td>
<td></td>
<td>noSubscriptionRecoveryPolicy</td>
</tr>
<tr>
<td></td>
<td></td>
<td>queryBasedSubscriptionRecoveryPolicy</td>
</tr>
<tr>
<td></td>
<td></td>
<td>retainedMessageSubscriptionRecoveryPolicy</td>
</tr>
<tr>
<td></td>
<td></td>
<td>timedSubscriptionRecoveryPolicy</td>
</tr>
<tr>
<td></td>
<td>sharedDeadLetterStrategy</td>
<td>deadLetterQueue</td>
</tr>
<tr>
<td></td>
<td></td>
<td>queue</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tempQueue</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tempTopic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>topic</td>
</tr>
<tr>
<td>virtualDestinationInterceptor</td>
<td>virtualDestinations</td>
<td>compositeQueue</td>
</tr>
<tr>
<td></td>
<td></td>
<td>compositeTopic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>virtualTopic</td>
</tr>
</tbody>
</table>

### Amazon MQ Child Element Attributes

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

#### Topics

- [authorizationEntry](#) (p. 111)
- [networkConnector](#) (p. 112)
- [kahaDB](#) (p. 113)
- [systemUsage](#) (p. 114)

#### authorizationEntry

authorizationEntry is a child of the authorizationEntries child collection element.

**Attributes**

- admin|read|write

The permissions granted to a group of users. For more information, see [Always configure an authorization map](#) (p. 166).

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

**Default:** null

#### Example Configuration

```xml
<authorizationPlugin>
  <map>
    <authorizationMap>
      <!-- Configuration content -->
    </authorizationMap>
  </map>
</authorizationPlugin>
```
<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>

networkConnector

networkConnector is a child of the networkConnectors child collection element.

Topics
- Attributes (p. 112)
- Example Configurations (p. 113)

Attributes

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

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

   Default: true

duplex

Specifies whether the connection in the network of brokers is used to produce and consume messages. For example, if broker A creates a connection to broker B in non-duplex mode, messages can be forwarded only from broker A to broker B. However, if broker A creates a duplex connection to broker B, then broker B can forward messages to broker A without having to configure a <networkConnector>.

   Default: false

name

The name of the bridge in the network of brokers.

   Default: bridge

uri

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

   Default: null

username

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

   Default: null
Example Configurations

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

A Network of Brokers with Two Brokers

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

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

For more information, see Configure Network Connectors for Your Broker (p. 36).

A Network of Brokers with Multiple Brokers

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

```
<networkConnectors>
  <networkConnector name="connector_1_to_2" userName="myCommonUser" duplex="true"
    url="masterslave:(ssl://b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-east-2.amazonaws.com:61617,
    ssl://b-9876l5k4-32ji-109h-8gfe-7d65c4b132al-2.mq.us-east-2.amazonaws.com:61617)="/>
</networkConnectors>
```

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

kahaDB

kahaDB is a child of the persistenceAdapter child collection element.

Attributes

concurrentStoreAndDispatchQueues

Specifies whether to use concurrent store and dispatch for queues. For more information, see Disable Concurrent Store and Dispatch for Queues with Slow Consumers (p. 72).

Default: true

cleanupOnStop

Supported in

Apache ActiveMQ 15.16.x and above

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

**journalDiskSyncInterval**

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

**Default:** 1000

**journalDiskSyncStrategy**

**Supported in**

Apache ActiveMQ 15.14.x and above

Configures the disk sync policy. For more information, see the [Apache ActiveMQ kahaDB documentation](#).

**Default:** always

**Note**

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

**preallocationStrategy**

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

**Default:** sparse_file

**Example Configuration**

**Example**

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

**systemUsage**

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

**Child Element**

**memoryUsage**

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

**Child Element**

**Attribute**

**percentOfJvmHeap**

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

Attributes

sendFailIfNoSpace

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

Default: false

sendFailIfNoSpaceAfterTimeout

Default: null

Example Configuration

Example

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

Managing Amazon MQ for ActiveMQ engine versions

Apache ActiveMQ organizes version numbers according to semantic versioning specification as X.Y.Z. In Amazon MQ for ActiveMQ implementations, X.Y denotes the major version, and Z represents the minor version number. Amazon MQ considers a version change to be major if the major version numbers change. For example, upgrading from version 5.15 to 5.16 is considered a major version upgrade. A version change is considered minor if only the minor version number changes. For example, upgrading from version 5.15.14 to 5.15.15 is considered a minor version upgrade.

Amazon MQ for ActiveMQ currently supports the following engine versions of Apache ActiveMQ.

<table>
<thead>
<tr>
<th>Major versions</th>
<th>Minor versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActiveMQ 5.16</td>
<td>• 5.16.3 (recommended)</td>
</tr>
<tr>
<td></td>
<td>• 5.16.2</td>
</tr>
<tr>
<td>ActiveMQ 5.15</td>
<td>• 5.15.15</td>
</tr>
<tr>
<td></td>
<td>• 5.15.14</td>
</tr>
<tr>
<td></td>
<td>• 5.15.13</td>
</tr>
<tr>
<td></td>
<td>• 5.15.12</td>
</tr>
<tr>
<td></td>
<td>• 5.15.10</td>
</tr>
<tr>
<td></td>
<td>• 5.15.9</td>
</tr>
<tr>
<td></td>
<td>• 5.15.8</td>
</tr>
<tr>
<td></td>
<td>• 5.15.6</td>
</tr>
<tr>
<td></td>
<td>• 5.15.0</td>
</tr>
</tbody>
</table>
**Note**

ActiveMQ 5.15.15 is the last minor version planned for the 5.15.x release. We recommend upgrading your brokers to the latest supported ActiveMQ major engine version, 5.16.x.

When you create a new Amazon MQ for ActiveMQ broker, you can specify any supported ActiveMQ engine version. If you use the AWS Management Console to create a broker, Amazon MQ automatically defaults to the latest engine version number. If you use the AWS CLI or the Amazon MQ API to create a broker, the engine version number is required. If you don't provide a version number, the operation will result in an exception. To learn more, see `create-broker` in the *AWS CLI Command Reference* and `CreateBroker` in the *Amazon MQ REST API Reference*.

**Topics**

- Major and minor version upgrades (p. 116)
- Listing supported engine versions (p. 116)

**Major and minor version upgrades**

With Amazon MQ, you control when to upgrade your brokers to new versions. When automatic minor version upgrade is activated, Amazon MQ will automatically upgrade your broker engine to new ActiveMQ minor versions as they are released and supported by Amazon MQ.

To perform a major version upgrade, you must manually upgrade your broker's engine version number. Minor and major version upgrades occur at the same time as other broker patching operations, during your scheduled maintenance window (p. 21). If you opt out of automatic minor version upgrades, you can manually upgrade your broker to a new supported minor version by following the same procedure as a major upgrade.

For more information about updating your broker preferences to activate or deactivate minor version upgrades, and manually upgrading your broker, see the section called "Upgrading the engine version" (p. 24).

**Listing supported engine versions**

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

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

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

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

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

**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. 43).
• The MQTT example Java code connects to a broker, creates a topic, and publishes and receives a message.
• The STOMP+WSS example Java code connects to a broker, creates a queue, and publishes and receives a message.

Prerequisites

Enable VPC Attributes

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

Enable inbound Connections

1. Sign in to the Amazon MQ console.
2. From the broker list, choose the name of your broker (for example, MyBroker).
3. On the MyBroker page, in the Connections section, note the addresses and ports of the broker's web console URL and wire-level protocols.
4. In the Details section, under Security and network, choose the name of your security group or
5. The Security Groups page of the EC2 Dashboard is displayed.
6. From the security group list, choose Inbound, and then choose Edit.
7. In the Edit inbound rules dialog box, add a rule for every URL or endpoint that you want to be publicly accessible (the following example shows how to do this for a broker web console).
   a. Choose Add Rule.
   b. For Type, select Custom TCP.
   c. For Port Range, type the web console port (8162).
   d. For Source, leave Custom selected and then type the IP address of the system that you want to be able to access the web console (for example, 192.0.2.1).
   e. Choose Save.

Your broker can now accept inbound connections.

Add Java dependencies

OpenWire

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

```xml
<dependencies>
    <dependency>
        <groupId>org.apache.activemq</groupId>
        <artifactId>activemq-client</artifactId>
        <version>5.15.8</version>
    </dependency>
    <dependency>
        <groupId>org.apache.activemq</groupId>
        <artifactId>activemq-pool</artifactId>
        <version>5.15.8</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.

```xml
<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.

```xml
<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>
</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

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

import org.apache.activemq.ActiveMQConnectionFactory;
import org.apache.activemq.jms.pool.PooledConnectionFactory;
import javax.jms.*;

public class AmazonMQExample {

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

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

        sendMessage(pooledConnectionFactory);
        receiveMessage(connectionFactory);

        pooledConnectionFactory.stop();
    }

    private static void sendMessage(PooledConnectionFactory pooledConnectionFactory) throws JMSException {
        // ...
    }

    private static void receiveMessage(ActiveMQConnectionFactory connectionFactory) throws JMSException {
        // ...
    }

    private static void receiveMessage(PooledConnectionFactory pooledConnectionFactory) throws JMSException {
        // ...
    }

    private static void createActiveMQConnectionFactory() {
        // ...
    }

    private static void createPooledConnectionFactory(ActiveMQConnectionFactory connectionFactory) {
        // ...
    }

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

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

public class AmazonMQExampleMqtt implements MqttCallback {

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

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

    private void run() throws MqttException, InterruptedException {

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

        // Create the MQTT client and specify the connection options.
        final String clientId = "abc123";
        final MqttClient client = new MqttClient(WIRE_LEVEL_ENDPOINT, clientId);
        final MqttConnectOptions connOpts = new MqttConnectOptions();
// Pass the 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.");
}
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-12345b6-78cd-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) {

RabbitMQ engine

This section describes the basic elements of a RabbitMQ broker and its supported plugins, and provides an overview of RabbitMQ broker architecture options on Amazon MQ.

Topics
- Basic elements (p. 124)
- Broker architecture (p. 132)
- Managing Amazon MQ for RabbitMQ engine versions (p. 134)

Basic elements

This section introduces key concepts essential to understanding RabbitMQ on Amazon MQ.

Topics
- Broker (p. 124)
- Broker defaults (p. 125)
- User (p. 130)
- Plugins (p. 131)

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, t3) and size (large, micro) is a broker instance type (for example, mq.m5.large). For more information, see Broker instance types (p. 136).

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

For more information, see Broker architecture (p. 132).

You can enable automatic minor version upgrades to new minor versions of the broker engine, as new versions of the RabbitMQ engine are released. 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).

Supported protocols

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

- AMQP (0-9-1)
Listener ports

Amazon MQ managed RabbitMQ brokers support the following listener ports for application-level connectivity via amqps, as well as client connections using the RabbitMQ web console and the management API.

- **Listener port 5671** - Used for connections made via the secure AMQP URL. For example, given a broker with broker ID `b-c8352341-ec91-4a78-ad9c-a43f23d325bb`, deployed in the `us-west-2` region, the following is the broker's full amqp URL: `b-c8352341-ec91-4a78-ad9c-a43f23d325bb.mq.us-west-2.amazonaws.com:5671`.

- **Listener ports 443 and 15671** - Both listener ports can be used interchangeably to access a broker via the RabbitMQ web console or the management API.

Attributes

A RabbitMQ broker has several attributes:

- A name. For example, `MyBroker`.
- An ID. For example, `b-1234a5b6-78cd-901e-2fgh-3i45j6k17819`.
- A RabbitMQ web console URL. For example, `https://b-1234a5b6-78cd-901e-2fgh-3i45j6k17819-1.mq.us-east-2.amazonaws.com`.

For more information, see RabbitMQ web console in the RabbitMQ documentation.

- A secure AMQP endpoint. For example, `amqps://b-1234a5b6-78cd-901e-2fgh-3i45j6k17819-1.mq.us-east-2.amazonaws.com`.

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

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

Broker defaults

When you create an Amazon MQ for RabbitMQ broker, Amazon MQ applies a default set of broker policies and vhost limits to optimize your broker's performance. Amazon MQ applies vhost limits only to the default (/) vhost. Amazon MQ will not apply default policies to newly created vhosts. We recommend keeping these defaults for all new and existing brokers. However, you can modify, override, or delete these defaults at any time.

Amazon MQ creates policies and limits based on the instance type and broker deployment mode that you choose when you create your broker. The default policies are named according to the deployment mode, as follows:

- **Single-instance** – `AWS-DEFAULT-POLICY-SINGLE-INSTANCE`
- **Cluster deployment** – `AWS-DEFAULT-POLICY-CLUSTER-MULTI-AZ`

For single-instance brokers (p. 133), Amazon MQ sets the policy priority value to 0. To override the default priority value, you can create your own custom policies with higher priority values. For cluster deployments (p. 133), Amazon MQ sets the priority value to 1 for broker defaults. To create your own custom policy for clusters, assign a priority value greater than 1.
Note
In cluster deployments, `ha-mode` and `ha-sync-mode` broker policies are required for classic mirroring and high availability (HA).
If you delete the default `AWS-DEFAULT-POLICY-CLUSTER-MULTI-AZ` policy, Amazon MQ uses the `ha-all-AWS-OWNED-DO-NOT-DELETE` policy with a priority value of 0. This ensures that the required `ha-mode` and `ha-sync-mode` policies are still in effect. If you create your own custom policy, Amazon MQ automatically appends `ha-mode` and `ha-sync-mode` to your policy definitions.

Topics
- Policy and limit descriptions (p. 126)
- Recommended default values (p. 127)
- Manually applying default policies and limits (p. 127)

Policy and limit descriptions

The following list describes the default policies and limits that Amazon MQ applies to a newly created broker. The values for `max-length`, `max-queues`, and `max-connections` vary based on your broker's instance type and deployment mode. These values are listed in the Recommended default values (p. 127) section.

- **queue-mode**: lazy (policy) – Enables lazy queues. By default, queues keep an in-memory cache of messages, enabling the broker to deliver messages to consumers as fast as possible. This can lead to the broker running out of memory and raising a high-memory alarm. Lazy queues attempt to move messages to disk as early as is practical. This means that fewer messages are kept in memory under normal operating conditions. Using lazy queues, Amazon MQ for RabbitMQ can support much larger messaging loads and longer queues. Note that for certain use cases, brokers with lazy queues might perform marginally slower. This is because messages are moved from disk to broker, as opposed to delivering messages from an in-memory cache.

  Deployment modes
  Single-instance, cluster

- **max-length**: `number-of-messages` (policy) – Sets a limit for the number of messages in a queue. In cluster deployments, the limit prevents paused queue synchronization in cases such as broker reboots, or following a maintenance window.

  Deployment modes
  Cluster

- **overflow**: reject-publish (policy) – Enforces queues with a `max-length` policy to reject new messages after the number of messages in the queue reaches the `max-length` value. To ensure that messages aren't lost if a queue is in an overflow state, client applications that publish messages to the broker must implement publisher confirms (p. 76). For information about implementing publisher confirms, see Publisher Confirms on the RabbitMQ website.

  Deployment modes
  Cluster

- **max-queues**: `number-of-queues-per-vhost` (vhost limit) – Sets the limit for the number of queues in a broker. Similar to the `max-length` policy definition, limiting the number of queues in cluster deployments prevents paused queue synchronization following broker reboots or maintenance windows. Limiting queues also prevents excessive amounts of CPU usage for maintaining queues.

  Deployment modes
  Single-instance, cluster

- **max-connections**: `number-of-connections-per-vhost` (vhost limit) – Sets the limit for the number of client connections to the broker. Limiting the number of connections according to the recommended values prevents excessive broker memory usage, which could result in the broker raising a high memory alarm and pausing operations.
Deployment modes
Single-instance, cluster

Recommended default values

Note
The max-length and max-queue default limits are tested and evaluated based on an average message size of 5 kB. If your messages are significantly larger than 5 kB, you will need to adjust and reduce the max-length and max-queue limits.

The following table lists the default limit values for a newly created broker. Amazon MQ applies these values according to the broker's instance type and deployment mode.

<table>
<thead>
<tr>
<th>Instance type</th>
<th>Deployment mode</th>
<th>max-length</th>
<th>max-queues</th>
<th>max-connections</th>
</tr>
</thead>
<tbody>
<tr>
<td>t3.micro</td>
<td>Single-instance</td>
<td>N/A</td>
<td>500</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>Cluster</td>
<td>8,000,000</td>
<td>4,000</td>
<td>15,000</td>
</tr>
<tr>
<td>m5.large</td>
<td>Single-instance</td>
<td>N/A</td>
<td>20,000</td>
<td>4,000</td>
</tr>
<tr>
<td></td>
<td>Cluster</td>
<td>9,000,000</td>
<td>5,000</td>
<td>20,000</td>
</tr>
<tr>
<td>m5.xlarge</td>
<td>Single-instance</td>
<td>N/A</td>
<td>30,000</td>
<td>8,000</td>
</tr>
<tr>
<td></td>
<td>Cluster</td>
<td>10,000,000</td>
<td>6,000</td>
<td>40,000</td>
</tr>
<tr>
<td>m5.2xlarge</td>
<td>Single-instance</td>
<td>N/A</td>
<td>60,000</td>
<td>15,000</td>
</tr>
<tr>
<td></td>
<td>Cluster</td>
<td>12,000,000</td>
<td>10,000</td>
<td>100,000</td>
</tr>
<tr>
<td>m5.4xlarge</td>
<td>Single-instance</td>
<td>N/A</td>
<td>150,000</td>
<td>30,000</td>
</tr>
<tr>
<td></td>
<td>Cluster</td>
<td>12,000,000</td>
<td>10,000</td>
<td>100,000</td>
</tr>
</tbody>
</table>

Manually applying default policies and limits

The following section describes applying custom policies and limits with Amazon MQ recommended default values. If you have deleted the recommended default policies and limits, and want to re-create them, or you have created additional vhosts and want to apply the default policies and limits to your new vhosts, you can use the following steps.

Important
To perform the following steps, you must have an Amazon MQ for RabbitMQ broker user with administrator permissions. You can use the administrator user created when you first created the broker, or another user that you might have created afterwards. The following table provides the necessary administrator user tag and permissions as regular expression (regexp) patterns.

<table>
<thead>
<tr>
<th>Tags</th>
<th>Read regexp</th>
<th>Configure regexp</th>
<th>Write regexp</th>
</tr>
</thead>
<tbody>
<tr>
<td>administrator</td>
<td>.*</td>
<td>.*</td>
<td>.*</td>
</tr>
</tbody>
</table>

For more information about creating RabbitMQ users and managing user tags and permissions, see User (p. 130).
To apply default policies and virtual host limits using the RabbitMQ web console

1. Sign in to the Amazon MQ console.
2. In the left navigation pane, choose Brokers.
3. From the list of brokers, choose the name of the broker to which you want to apply the new policy.
4. On the broker details page, in the Connections section, choose the RabbitMQ web console URL. The RabbitMQ web console opens in a new browser tab or window.
5. Log in to the RabbitMQ web console with your broker administrator user name and password.
6. In the RabbitMQ web console, at the top of the page, choose Admin.
7. On the Admin page, in the right navigation pane, choose Policies.
8. On the Policies page, you can see a list of the broker's current User policies. Below User policies, expand Add / update a policy.
9. To create a new broker policy, under Add / update a policy, do the following:
   a. For Virtual host, choose the name of the vhost to which you want to attach the policies from the dropdown list. To choose the default vhost, choose /.
      
      Note
      If you have not created additional vhosts, the Virtual host option is not shown in the RabbitMQ console, and the policies are applied only to the default vhost.
   b. For Name, enter a name for your policy, for example, policy-defaults.
   c. For Pattern, enter the regexp pattern .* so that the policy matches all queues on the broker.
   d. For Apply to, choose Exchanges and queues from the dropdown list.
   e. For Priority, enter an integer greater than all other policies applied to the vhost. You can apply exactly one set of policy definitions to RabbitMQ queues and exchanges at any given time. RabbitMQ chooses the matching policy with the highest priority value. For more information about policy priorities and how to combine policies, see Policies in the RabbitMQ Server Documentation.
   f. For Definition, add the following key-value pairs:
      - queue-mode=lazy. Choose String from the dropdown list.
      - overflow=reject-publish. Choose String from the dropdown list.
      
      Note
      Does not apply to single-instance brokers.
      
      • max-length=number-of-messages. Replace number-of-messages with the Amazon MQ recommended value (p. 127) according to the broker's instance size and deployment mode, for example, 8000000 for an mq.m5.large cluster. Choose Number from the dropdown list.
      
      Note
      Does not apply to single-instance brokers.
   g. Choose Add / update policy.
10. Confirm that the new policy appears in the list of User policies.

   Note
   For cluster brokers, Amazon MQ automatically applies the ha-mode: all and ha-sync-mode: automatic policy definitions.
11. From the right navigation pane, choose Limits.
12. On the Limits page, you can see a list of the broker's current Virtual host limits. Below Virtual host limits, expand Set / update a virtual host limit.
13. To create a new vhost limit, under Set / update a virtual host limit, do the following:
   a. For Virtual host, choose the name of the vhost to which you want to attach the policies from the dropdown list. To choose the default vhost, choose /.
b. For Limit, choose **max-connections** from the dropdown options.

c. For Value, enter the Amazon MQ recommended value (p. 127) according to the broker's instance size and deployment mode, for example, **15000** for an `mq.m5.large` cluster.

d. Choose **Set / update limit**.

e. Repeat the steps above, and for Limit, choose **max-queues** from the dropdown options.

14. Confirm that the new limits appear in the list of **Virtual host limits**.

**To apply default policies and virtual host limits using the RabbitMQ management API**

1. Sign in to the Amazon MQ console.

2. In the left navigation pane, choose **Brokers**.

3. From the list of brokers, choose the name of the broker to which you want to apply the new policy.

4. On the broker's page, in the **Connections** section, note the **RabbitMQ web console** URL. This is the broker endpoint that you use in an HTTP request.

5. Open a new terminal or command line window of your choice.

6. To create a new broker policy, enter the following *curl* command. This command assumes a queue on the default `/vhost`, which is encoded as `%2F`. To apply the policy to another vhost, replace `%2F` with the vhost's name.

   ```bash
   curl -i -u username:password -H "content-type:application/json" -XPUT
   -d '{"pattern": ".*", "priority": 1, "definition":{"queue-mode": "lazy", "overflow": "reject-publish", "max-length": "number-of-messages"}}' \
   broker-endpoint/api/policies/%2F/policy-name
   ```

   **Note**
   
   Replace `username` and `password` with your administrator user name and password.

   Replace `number-of-messages` with the Amazon MQ recommended value (p. 127) according to the broker's instance size and deployment mode. Replace `policy-name` with a name for your policy. Replace `broker-endpoint` with the URL that you noted previously.

7. To confirm that the new policy is added to your broker's user policies, enter the following *curl* command to list all broker policies.

   ```bash
   curl -i -u username:password broker-endpoint/api/policies
   ```

8. To create a new `max-connections` virtual host limits, enter the following *curl* command. This command assumes a queue on the default `/vhost`, which is encoded as `%2F`. To apply the policy to another vhost, replace `%2F` with the vhost's name.

   ```bash
   curl -i -u username:password -H "content-type:application/json" -XPUT
   -d '{"value": "number-of-connections"}' \
   broker-endpoint/api/vhost-limits/%2F/max-connections
   ```

   **Note**
   
   Replace `username` and `password` with your administrator user name and password.

   Replace `max-connections` with the Amazon MQ recommended value (p. 127) according to the broker's instance size and deployment mode. Replace the broker endpoint with the URL that you noted previously.

9. To create a new `max-queues` virtual host limit, repeat the previous step, but modify the *curl* command as shown in the following.

   ```bash
   curl -i -u username:password -H "content-type:application/json" -XPUT
   -d '{"value": "number-of-queues"}' \
   broker-endpoint/api/vhost-limits/%2F/max-queues
   ```
Basic elements

```
-d '{"value":"number-of-queues"}' \
broker-endpoint/api/vhost-limits/%2F/max-queues
```

10. To confirm that the new limits are added to your broker's virtual host limits, enter the following `curl` command to list all broker virtual host limits.

```
curl -i -u username:password broker-endpoint/api/vhost-limits
```

**User**

Every AMQP 0-9-1 client connection has an associated user which must be authenticated. Each client connection also targets a virtual host (vhost) for which the user must have a set of permissions. A user may have permission to **configure**, **write** to, and **read** from queues and exchanges in a vhost. User credentials, and the target vhost are specified at the time the connection is established.

When you first create an Amazon MQ for RabbitMQ broker, Amazon MQ uses the username and password you provide to create a RabbitMQ user with the `administrator` tag. You can then add and manage users via the RabbitMQ management API or the RabbitMQ web console. You can also use the RabbitMQ web console or the management API to set or modify user permissions and tags.

**Note**

RabbitMQ users will not be stored or displayed via the Amazon MQ Users API.

To create a new user with the RabbitMQ management API, use the following API endpoint and request body. Replace `username` and `password` with your new username and password. When creating users via the RabbitMQ web console or the management API, avoid `guest` as a username. Amazon MQ for RabbitMQ prohibits users with the `guest` username from accessing the broker remotely via the RabbitMQ web console, the management API, or via an application-level connection.

```
POST /api/users/username HTTP/1.1
{"password":"password","tags":"administrator"}
```

**Important**

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

The `tags` key is mandatory, and is a comma-separated list of tags for the user. Amazon MQ supports `administrator`, `management`, and `monitoring` user tags.

You can set permissions for an individual user by using the following API endpoint and request body. Replace `vhost` and `username` with your information. For the default vhost `/`, use `2f%`.

```
POST /api/users/vhost/username HTTP/1.1
{"configure":".*","write":".*","read":".*"}
```

**Note**

The `configure`, `read`, and `write` keys are all mandatory.

By using the wildcard `.*` value, this operation will grant read, write, and configure permissions for all queues in the specified vhost to the user. For more information about managing users via the RabbitMQ management API, see RabbitMQ Management HTTP API.
Plugins

Amazon MQ for RabbitMQ supports the RabbitMQ management plugin which powers the management API and the RabbitMQ web console. You can use the web console and the management API to create and manage broker users and policies.

In addition to the management plugin, Amazon MQ for RabbitMQ also supports the following plugins.

Topics
- Shovel plugin (p. 131)
- Federation plugin (p. 131)
- Consistent Hash exchange plugin (p. 132)

Shovel plugin

Amazon MQ managed brokers support RabbitMQ shovel, allowing you to move messages from queues and exchanges on one broker instance to another. You can use shovel to connect loosely coupled brokers and distribute messages away from nodes with heavier message loads.

Amazon MQ managed RabbitMQ brokers support dynamic shovels. Dynamic shovels are configured using runtime parameters, and can be started and stopped at any time programatically by a client connection. For example, using the RabbitMQ management API, you can create a PUT request to the following API endpoint to configure a dynamic shovel. In the example, {vhost} can be replaced by the name of the broker's vhost, and {name} replaced by the name of the new dynamic shovel.

```
/api/parameters/shovel/{vhost}/{name}
```

In the request body, you must specify either a queue or an exchange but not both. This example below configures a dynamic shovel between a local queue specified in src-queue and a remote queue defined in dest-queue. Similarly, you can use src-exchange and dest-exchange parameters to configure a shovel between two exchanges.

```
{
  "value": {
    "src-protocol": "amqp091",
    "src-uri": "amqp://localhost",
    "src-queue": "source-queue-name",
    "dest-protocol": "amqp091",
    "dest-uri": "amqps://b-c8352341-ec91-4a78-ad9c-a43f23d325bb.mq.us-west-2.amazonaws.com:5671",
    "dest-queue": "destination-queue-name"
  }
}
```

For more information about using dynamic shovels, see RabbitMQ dynamic shovel plugin.

Note

Amazon MQ does not support using static shovels.

Federation plugin

Amazon MQ supports federated exchanges and queues. With federation, you can replicate the flow of messages between queues, exchanges and consumers on separate brokers. Federated queues and exchanges use point-to-point links to connect to peers in other brokers. While federated exchanges, by default, route messages once, federated queues can move messages any number of times as needed by consumers.
You can use federation to allow a downstream broker to consume a message from an exchange or a queue on an upstream. You can enable federation on downstream brokers by using the RabbitMQ web console or the management API.

For example, using the management API, you can configure federation by doing the following.

1. Configure one or more upstreams that define federation connections to other nodes. You can define federation connections by using the RabbitMQ web console or the management API. Using the management API, you can create a POST request to `/api/parameters/federation-upstream/%2f/my-upstream` with the following request body.

```json
{
  "value": {
    "uri": "amqp://server-name",
    "expires": 3600000
  }
}
```

2. Configure a policy to enable your queues or exchanges to become federated. You can configure policies by using the RabbitMQ web console, or the management API. Using the management API, you can create a POST request to `/api/policies/%2f/federate-me` with the following request body.

```json
{
  "pattern": "^amq\.",
  "definition": {
    "federation-upstream-set": "all",
    "apply-to": "exchanges"
  }
}
```

**Note**
The request body assumes exchanges on the server are named beginning with `amq`. Using regular expression `^amq\.` will ensure that federation is enabled for all exchanges whose names begin with "amq." The exchanges on your RabbitMQ server can be named differently.

For more information about configuring the federation plugin, see [RabbitMQ federation plugin](https://github.com/rabbitmq/rabbitmq-server/blob/main/doc/en/interconnect/federation.md).

### Consistent Hash exchange plugin

By default, Amazon MQ for RabbitMQ supports the Consistent Hash exchange type plugin. Consistent Hash exchanges route messages to queues based on a hash value calculated from the routing key of a message. Given a reasonably even routing key, Consistent Hash exchanges can distribute messages between queues reasonably evenly.

For queues bound to a Consistent Hash exchange, the binding key is a number-as-a-string that determines the binding weight of each queue. Queues with a higher binding weight will receive a proportionally higher distribution of messages from the Consistent Hash exchange to which they are bound. In a Consistent Hash exchange topology, publishers can simply publish messages to the exchange, but consumers must be explicitly configured to consume messages from specific queues.

For more information about Consistent Hash exchanges, see [RabbitMQ Consistent Hash Exchange Type](https://github.com/rabbitmq/rabbitmq-server/blob/main/doc/en/interconnect/federation.md) on the GitHub website.

### Broker architecture

RabbitMQ brokers can be created as single-instance brokers or in a cluster deployment. For both deployment modes, Amazon MQ provides high durability by storing its data redundantly.

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

- **AMQP (0-9-1)**

**Topics**

- Single-instance broker (p. 133)
- Cluster deployment for high availability (p. 133)
**Single-instance broker**

A *single-instance broker* is comprised of one broker in one Availability Zone behind a Network Load Balancer (NLB). The broker communicates with your application and with an Amazon EBS storage volume. Amazon EBS provides block level storage optimized for low-latency and high throughput.

Using an Network Load Balancer ensures that your Amazon MQ for RabbitMQ broker endpoint remains unchanged if the broker instance is replaced during a maintenance window or because of underlying Amazon EC2 hardware failures. An Network Load Balancer allows your applications and users to continue to use the same endpoint to connect to the broker.

The following diagram illustrates an Amazon MQ for RabbitMQ single-instance broker.

---

**Cluster deployment for high availability**

A *cluster deployment* is a logical grouping of three RabbitMQ broker nodes behind a Network Load Balancer, each sharing users, queues, and a distributed state across multiple Availability Zones (AZ).

In a cluster deployment, Amazon MQ automatically manages broker policies to enable classic mirroring across all nodes, ensuring high availability (HA). Each mirrored queue consists of one *main* node and one or more *mirrors*. Each queue has its own main node. All operations for a given queue are first applied on the queue's main node and then propagated to mirrors. Amazon MQ creates a default system policy that sets the *ha-mode* to *all* and *ha-sync-mode* to *automatic*. This ensures that data is replicated to all nodes in the cluster across different Availability Zones for better durability.

**Note**

During a *maintenance window*, all maintenance to a cluster is performed one node at a time, keeping at least two running nodes at all times. Each time a node is brought down, client connections to that node are severed and need to be re-established. You must ensure that your client code is designed to automatically reconnect to your cluster. For more information about connection recovery, see *the section called “Automatically recover from network failures“* (p. 77). Because Amazon MQ sets *ha-sync-mode: automatic*, during a maintenance window, queues will synchronize when each node re-joins the cluster. Queue synchronization blocks all other queue operations. You can mitigate the impact of queue synchronization during maintenance windows by keeping queues short.

The default policy should not be deleted. If you do delete this policy, Amazon MQ will be automatically recreate it. Amazon MQ will also ensure that HA properties are applied to all other policies that you create on a clustered broker. If you add a policy without the HA properties, Amazon MQ will add them for you. If you add a policy with different high availability properties, Amazon MQ will replace them. For more information about classic mirroring, see *Classic mirrored queues*. 
**Important**
Amazon MQ does not support quorum queues. Enabling the quorum queue feature flag and creating quorum queues will result in data loss.

The following diagram illustrates a RabbitMQ cluster broker deployment with three nodes in three Availability Zones (AZ), each with its own Amazon EBS volume and a shared state. Amazon EBS provides block level storage optimized for low-latency and high throughput.

---

Managing Amazon MQ for RabbitMQ engine versions

RabbitMQ organizes version numbers according to semantic versioning specification as \( X.Y.Z \). In Amazon MQ for RabbitMQ implementations, \( X.Y \) denotes the major version, and \( Z \) represents the minor version number. Amazon MQ considers a version change to be major if the major version numbers change. A version change is considered minor if only the minor version number changes. For example, upgrading from version 3.8.11 to 3.8.17 is considered a *minor version upgrade*.

Amazon MQ for RabbitMQ currently supports the following engine versions of RabbitMQ.

<table>
<thead>
<tr>
<th>Major versions</th>
<th>Minor versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>RabbitMQ 3.8</td>
<td>• 3.8.26 (recommended)</td>
</tr>
</tbody>
</table>
When you create a new Amazon MQ for RabbitMQ broker, you can specify any supported RabbitMQ engine version. If you use the AWS Management Console to create a broker, Amazon MQ automatically defaults to the latest engine version number. If you use the AWS CLI or the Amazon MQ API to create a broker, the engine version number is required. If you don’t provide a version number, the operation will result in an exception. To learn more, see `create-broker` in the AWS CLI Command Reference and CreateBroker in the Amazon MQ REST API Reference.

### Topics
- Major and minor version upgrades (p. 135)
- Listing supported engine versions (p. 135)

### Major and minor version upgrades

With Amazon MQ, you control when to upgrade your brokers to new versions. When automatic minor version upgrade is activated, Amazon MQ will automatically upgrade your broker engine to new ActiveMQ minor versions as they are released and supported by Amazon MQ.

To perform a major version upgrade, you must manually upgrade your broker's engine version number. Minor and major version upgrades occur at the same time as other broker patching operations, during your scheduled maintenance window (p. 21). If you opt out of automatic minor version upgrades, you can manually upgrade your broker to a new supported minor version by following the same procedure as a major upgrade.

For more information about updating your broker preferences to activate or deactivate minor version upgrades, and manually upgrading your broker, see the section called “Upgrading the engine version” (p. 24).

**Note**

Amazon MQ for RabbitMQ supports one major release (3.8.x), and no major version upgrades are currently available.

### Listing supported engine versions

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

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

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

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

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

---

<table>
<thead>
<tr>
<th>Major versions</th>
<th>Minor versions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• 3.8.23</td>
</tr>
<tr>
<td></td>
<td>• 3.8.22</td>
</tr>
<tr>
<td></td>
<td>• 3.8.17 (existing brokers only)</td>
</tr>
<tr>
<td></td>
<td>• 3.8.11</td>
</tr>
<tr>
<td></td>
<td>• 3.8.6</td>
</tr>
</tbody>
</table>
Instance types

The combined description of the broker instance class (m5, t3) 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 for each supported engine type.

Topics
- Amazon MQ for ActiveMQ instance types (p. 136)
- Amazon MQ for RabbitMQ instance types (p. 139)

Amazon MQ for ActiveMQ instance types

Important
You can use Amazon EBS only with the mq.m5 broker instance type family. For more information, see Storage (p. 82).

<table>
<thead>
<tr>
<th>Instance Type</th>
<th>vCPU</th>
<th>Memory (GiB)</th>
<th>Network Performance</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>mq.t2.micro</td>
<td>1</td>
<td>1</td>
<td>Low</td>
<td>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. <strong>Note</strong> Using the mq.t2.micro instance type is subject to CPU credits and baseline performance—with the ability to burst above the baseline level (for more information, see the CpuCreditBalance metric). If your application requires...</td>
</tr>
</tbody>
</table>

136
<table>
<thead>
<tr>
<th>Instance Type</th>
<th>vCPU</th>
<th>Memory (GiB)</th>
<th>Network Performance</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>fixed performance, consider using an mq.m5.large instance type.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mq.t3.micro</td>
<td>2</td>
<td>1</td>
<td>Low</td>
<td>Use the mq.t3.micro instance type for basic evaluation of Amazon MQ. This instance type (single-instance brokers only) qualifies for the AWS Free Tier.</td>
</tr>
<tr>
<td>mq.m4.large</td>
<td>2</td>
<td>8</td>
<td>Moderate</td>
<td>Use the mq.m4.large instance type for compatibility with existing broker deployments. We recommend using an mq.m5.* instance for new brokers.</td>
</tr>
<tr>
<td>mq.m5.large</td>
<td>2</td>
<td>8</td>
<td>High</td>
<td>Use the mq.m5.large instance for regular development, testing, and production workloads.</td>
</tr>
<tr>
<td>Instance Type</td>
<td>vCPU</td>
<td>Memory (GiB)</td>
<td>Network Performance</td>
<td>Notes</td>
</tr>
<tr>
<td>----------------</td>
<td>------</td>
<td>--------------</td>
<td>---------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>mq.m5.xlarge</td>
<td>4</td>
<td>16</td>
<td>High</td>
<td>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.</td>
</tr>
<tr>
<td>mq.m5.2xlarge</td>
<td>8</td>
<td>32</td>
<td>High</td>
<td></td>
</tr>
<tr>
<td>mq.m5.4xlarge</td>
<td>16</td>
<td>64</td>
<td>High</td>
<td></td>
</tr>
</tbody>
</table>
### Amazon MQ for RabbitMQ instance types

<table>
<thead>
<tr>
<th>Instance Type</th>
<th>vCPU</th>
<th>Memory (GiB)</th>
<th>Network Performance</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>mq.t3.micro</td>
<td>2</td>
<td>1</td>
<td>Low</td>
<td>Use the mq.t3.micro instance type for basic evaluation of Amazon MQ. This instance type (single-instance brokers only) qualifies for the AWS Free Tier. <strong>Important</strong> The mq.t3.micro instance type does not support cluster deployment (p. 133).</td>
</tr>
<tr>
<td>mq.m5.large</td>
<td>2</td>
<td>8</td>
<td>High</td>
<td>Use the mq.m5.large instance for regular development, testing, and production workloads.</td>
</tr>
<tr>
<td>mq.m5.xlarge</td>
<td>4</td>
<td>16</td>
<td>High</td>
<td>Use the mq.m5.xlarge, mg.m5.2xlarge, and</td>
</tr>
<tr>
<td>mq.m5.2xlarge</td>
<td>8</td>
<td>32</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For more information about throughput considerations, see Choose the Correct Broker Instance Type for the Best Throughput (p. 72).
### Broker statuses

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

<table>
<thead>
<tr>
<th>Console</th>
<th>API</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creation failed</td>
<td>CREATION_FAILED</td>
<td>The broker couldn’t be created.</td>
</tr>
<tr>
<td>Creation in progress</td>
<td>CREATION_IN_PROGRESS</td>
<td>The broker is currently being created.</td>
</tr>
<tr>
<td>Deletion in progress</td>
<td>DELETION_IN_PROGRESS</td>
<td>The broker is currently being deleted.</td>
</tr>
<tr>
<td>Reboot in progress</td>
<td>REBOOT_IN_PROGRESS</td>
<td>The broker is currently being rebooted.</td>
</tr>
<tr>
<td>Running</td>
<td>RUNNING</td>
<td>The broker is operational.</td>
</tr>
</tbody>
</table>

### Tagging resources

Amazon MQ supports resource tagging to help track your cost allocation. You can tag resources when creating them, or by viewing the details of that resource.

**Topics**
- Tagging for Cost Allocation (p. 140)
- Managing Tags in the Amazon MQ Console (p. 141)
- Managing Using Amazon MQ API Actions (p. 142)

### Tagging for Cost Allocation

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

<table>
<thead>
<tr>
<th>Resource</th>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Broker1</td>
<td>Cost Center</td>
<td>34567</td>
</tr>
<tr>
<td></td>
<td>Stack</td>
<td>Production</td>
</tr>
<tr>
<td>Broker2</td>
<td>Cost Center</td>
<td>34567</td>
</tr>
<tr>
<td></td>
<td>Stack</td>
<td>Production</td>
</tr>
<tr>
<td>Broker3</td>
<td>Cost Center</td>
<td>12345</td>
</tr>
<tr>
<td></td>
<td>Stack</td>
<td>Development</td>
</tr>
</tbody>
</table>

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

### Managing Tags in the Amazon MQ Console

#### Adding Tags to New Resources

Amazon MQ lets you to add tags to resources as they are created. You can quickly add tags to the resources you are creating in the Amazon MQ console.

To add tags as you create a new broker:

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

To add tags as you create a configuration:

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

Amazon MQ allows you to view and manage the tags for your resources in the Amazon MQ console. You can manage tags for an individual resource by editing the tags on the details page for that resource. To edit tags on Amazon MQ resources:

1. Select either **Brokers** or **Configurations** in the Amazon MQ console.
   
   Under the **Tags** section, review the existing tags for that resource.

2. To add new or manage existing tags, select **Edit** (or **Create tag** if have no existing tags).

3. Update tags for your resource:
   
   - To modify existing tags, edit the **Key** and **Value**.
   - To remove existing tags, select **Remove**.
   - To add a new tag, select **Add tag** and enter a **Key** and **Value**.

4. Select **Save**.

Managing Using Amazon MQ API Actions

Amazon MQ allows you to view and manage the tags of your resources using the REST API.

For more information, see the Amazon MQ REST API Reference.
Security in Amazon MQ

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from data centers and network architectures that are built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security of the cloud and security in the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS Compliance Programs. To learn about the compliance programs that apply to Amazon MQ, see AWS Services in Scope by Compliance Program.

- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using Amazon MQ. The following topics show you how to configure Amazon MQ to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your Amazon MQ resources.

**Topics**

- Data protection in Amazon MQ (p. 143)
- Identity and access Management for Amazon MQ (p. 146)
- Compliance validation for Amazon MQ (p. 165)
- Resilience in Amazon MQ (p. 166)
- Infrastructure security in Amazon MQ (p. 166)
- Security best practices for Amazon MQ (p. 166)

Data protection in Amazon MQ

The AWS shared responsibility model applies to data protection in Amazon MQ. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. This content includes the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the AWS Security Blog.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual user accounts with AWS Identity and Access Management (IAM). That way each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We recommend TLS 1.2 or later.
- Set up API and user activity logging with AWS CloudTrail.
• Use AWS encryption solutions, along with all default security controls within AWS services.
• Use advanced managed security services such as Amazon Macie, which assists in discovering and securing personal data that is stored in Amazon S3.
• If you require FIPS 140-2 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-2.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form fields such as a Name field. This includes when you work with Amazon MQ or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

For both Amazon MQ for ActiveMQ and Amazon MQ for RabbitMQ brokers, do not use any personally identifiable information (PII) or other confidential or sensitive information for broker names or usernames when creating resources via the broker web console, or the Amazon MQ API. Broker names and usernames are accessible to other AWS services, including CloudWatch Logs. Broker usernames are not intended to be used for private or sensitive data.

Encryption

User data stored in Amazon MQ is encrypted at rest. Amazon MQ encryption at rest provides enhanced security by encrypting your data using encryption keys stored in the AWS Key Management Service (KMS). This service helps reduce the operational burden and complexity involved in protecting sensitive data. With encryption at rest, you can build security-sensitive applications that meet encryption compliance and regulatory requirements.

All connections between Amazon MQ brokers use Transport layer Security (TLS) to provide encryption in transit.

Amazon MQ encrypts messages at rest and in transit using encryption keys that it manages and stores securely. For more information, see the AWS Encryption SDK Developer Guide.

Encryption at rest

Amazon MQ integrates with AWS Key Management Service (KMS) to offer transparent server-side encryption. Amazon MQ always encrypts your data at rest.

Encryption at rest for ActiveMQ brokers

When you create an Amazon MQ for ActiveMQ broker, you can specify the AWS KMS key that you want Amazon MQ to use to encrypt your data at rest. If you don't specify a KMS key, Amazon MQ creates an AWS managed KMS key for you and uses it on your behalf. For more information about KMS keys, see AWS KMS keys in the AWS Key Management Service Developer Guide.

When creating a broker, you can configure what Amazon MQ uses for your encryption key by selecting one of the following.

• AWS owned KMS key — The key is owned by Amazon MQ and is not in your account.
• AWS managed KMS key — The AWS managed KMS key (aws/mq) is a KMS key in your account that is created, managed, and used on your behalf by Amazon MQ.
• Select existing customer managed KMS key — Customer managed KMS keys are created and managed by you in AWS Key Management Service (KMS).
Amazon MQ Developer Guide

Encryption in transit

**Important**
Amazon MQ uses Amazon Elastic File System (EFS) to store message data. If you revoke the grant that gives Amazon EFS permission to use the KMS keys in your account, Amazon MQ cannot access this data and your broker will stop working. When you revoke a grant for Amazon EFS, it will not take place immediately. To revoke access rights, delete your broker rather than revoking the grant.

**Encryption at rest for RabbitMQ brokers**

When you create a RabbitMQ brokers, Amazon MQ creates an AWS managed KMS key and uses it on your behalf. This AWS managed KMS key is owned by Amazon MQ and is not stored in your AWS account. Currently, Amazon MQ does not support AWS managed KMS keys owned by you and saved in your account, or customer managed KMS keys created and managed by you.

For more information about KMS keys, see [AWS KMS keys](https://docs.aws.amazon.com/kms/latest/developerguide/aws-kms-encrypting-data.html) in the AWS Key Management Service Developer Guide.

**Encryption in transit**

Amazon MQ encrypts data in transit between the brokers of your Amazon MQ deployment. All data that passes between Amazon MQ brokers is encrypted using Transport layer Security (TLS). This is true for all available protocols.

By default, Amazon MQ brokers use the recommended TLS 1.2 to encrypt data.

**Amazon MQ for ActiveMQ protocols**

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

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

**Supported TLS Cipher Suites for ActiveMQ**

ActiveMQ on Amazon MQ supports the following cipher suites:

- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
- TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
- TLS_DHE_RSA_WITH_AES_256_CBC_SHA
- TLS_RSA_WITH_AES_256_GCM_SHA384
- TLS_RSA_WITH_AES_256_CBC_SHA256
- TLS_RSA_WITH_AES_256_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
Amazon MQ for RabbitMQ protocols

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

- AMQP (0-9-1)

Supported TLS Cipher Suites for RabbitMQ

RabbitMQ on Amazon MQ supports the following cipher suites:

- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256

Identity and access Management for Amazon MQ

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be authenticated (signed in) and authorized (have permissions) to use Amazon MQ resources. IAM is an AWS service that you can use with no additional charge.

Topics

- Audience (p. 146)
- Authenticating with identities (p. 147)
- Managing access using policies (p. 149)
- How Amazon MQ works with IAM (p. 150)
- Amazon MQ Identity-based policy examples (p. 154)
- API authentication and authorization for Amazon MQ (p. 156)
- AWS managed policies for Amazon MQ (p. 158)
- Using service-linked roles for Amazon MQ (p. 159)
- Troubleshooting Amazon MQ identity and access (p. 163)

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in Amazon MQ.

Service user – If you use the Amazon MQ service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more Amazon MQ features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in Amazon MQ, see Troubleshooting Amazon MQ identity and access (p. 163).
Service administrator – If you're in charge of Amazon MQ resources at your company, you probably have full access to Amazon MQ. It's your job to determine which Amazon MQ features and resources your employees should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with Amazon MQ, see How Amazon MQ works with IAM (p. 150).

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to Amazon MQ. To view example Amazon MQ identity-based policies that you can use in IAM, see Amazon MQ Identity-based policy examples (p. 154).

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. For more information about signing in using the AWS Management Console, see Signing in to the AWS Management Console as an IAM user or root user in the IAM User Guide.

You must be authenticated (signed in to AWS) as the AWS account root user, an IAM user, or by assuming an IAM role. You can also use your company's single sign-on authentication or even sign in using Google or Facebook. In these cases, your administrator previously set up identity federation using IAM roles. When you access AWS using credentials from another company, you are assuming a role indirectly.

To sign in directly to the AWS Management Console, use your password with your root user email address or your IAM user name. You can access AWS programmatically using your root user or IAM users access keys. AWS provides SDK and command line tools to cryptographically sign your request using your credentials. If you don't use AWS tools, you must sign the request yourself. Do this using Signature Version 4, a protocol for authenticating inbound API requests. For more information about authenticating requests, see Signature Version 4 signing process in the AWS General Reference.

Regardless of the authentication method that you use, you might also be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Using multi-factor authentication (MFA) in AWS in the IAM User Guide.

AWS account root user

When you first create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account root user and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the best practice of using the root user only to create your first IAM user. Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.

IAM users and groups

An IAM user is an identity within your AWS account that has specific permissions for a single person or application. An IAM user can have long-term credentials such as a user name and password or a set of access keys. To learn how to generate access keys, see Managing access keys for IAM users in the IAM User Guide. When you generate access keys for an IAM user, make sure you view and securely save the key pair. You cannot recover the secret access key in the future. Instead, you must generate a new access key pair.

An IAM group is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named IAMAdmins and give that group permissions to administer IAM resources.
Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see When to create an IAM user (instead of a role) in the IAM User Guide.

IAM roles

An IAM role is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by switching roles. You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Using IAM roles in the IAM User Guide.

IAM roles with temporary credentials are useful in the following situations:

- **Temporary IAM user permissions** – An IAM user can assume an IAM role to temporarily take on different permissions for a specific task.
- **Federated user access** – Instead of creating an IAM user, you can use existing identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as federated users. AWS assigns a role to a federated user when access is requested through an identity provider. For more information about federated users, see Federated users and roles in the IAM User Guide.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the IAM User Guide.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
- **Principal permissions** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both actions. To see whether an action requires additional dependent actions in a policy, see Actions, Resources, and Condition Keys for Amazon MQ in the Service Authorization Reference.
- **Service role** – A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Creating a role to delegate permissions to an AWS service in the IAM User Guide.
- **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.
- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Using an IAM role to grant permissions to applications running on Amazon EC2 instances in the IAM User Guide.

To learn whether to use IAM roles or IAM users, see When to create an IAM role (instead of a user) in the IAM User Guide.
Managing access using policies

You control access in AWS by creating policies and attaching them to IAM identities or AWS resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. You can sign in as the root user or an IAM user, or you can assume an IAM role. When you then make a request, AWS evaluates the related identity-based or resource-based policies. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the IAM User Guide.

Administrators can use AWS JSON policies to specify who has access to what. That is, which principal can perform actions on what resources, and under what conditions.

Every IAM entity (user or role) starts with no permissions. In other words, by default, users can do nothing, not even change their own password. To give a user permission to do something, an administrator must attach a permissions policy to a user. Or the administrator can add the user to a group that has the intended permissions. When an administrator gives permissions to a group, all users in that group are granted those permissions.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the iam:GetRole action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM policies in the IAM User Guide.

Identity-based policies can be further categorized as inline policies or managed policies. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choosing between managed policies and inline policies in the IAM User Guide.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM role trust policies and Amazon S3 bucket policies. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Access Control Lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the Amazon Simple Storage Service Developer Guide.
Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the Principal field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the IAM User Guide.

- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see How SCPs work in the AWS Organizations User Guide.

- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session policies in the IAM User Guide.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the IAM User Guide.

How Amazon MQ works with IAM

Before you use IAM to manage access to Amazon MQ, you should understand what IAM features are available to use with Amazon MQ. To get a high-level view of how Amazon MQ and other AWS services work with IAM, see AWS Services That Work with IAM in the IAM User Guide.

Amazon MQ uses IAM for creating, updating, and deleting operations, but native ActiveMQ authentication for brokers. For more information, see Integrating ActiveMQ brokers with LDAP (p. 47).

Topics

- Amazon MQ identity-based policies (p. 150)
- Amazon MQ Resource-based policies (p. 153)
- Authorization based on Amazon MQ tags (p. 153)
- Amazon MQ IAM roles (p. 154)

Amazon MQ identity-based policies

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. Amazon MQ supports specific actions, resources, and condition keys. To learn about all of the elements that you use in a JSON policy, see IAM JSON Policy Elements Reference in the IAM User Guide.
Actions

Administrators can use AWS JSON policies to specify who has access to what. That is, which principal can perform actions on what resources, and under what conditions.

The Action element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as permission-only actions that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called dependent actions.

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

Policy actions in Amazon MQ use the following prefix before the action: mq:. For example, to grant someone permission to run an Amazon MQ instance with the Amazon MQ CreateBroker API operation, you include the mq:CreateBroker action in their policy. Policy statements must include either an Action or NotAction element. Amazon MQ defines its own set of actions that describe tasks that you can perform with this service.

To specify multiple actions in a single statement, separate them with commas as follows:

```
"Action": [  
    "mq:action1",  
    "mq:action2"
]
```

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

```
"Action": "mq:Describe*"
```

To see a list of Amazon MQ actions, see Actions Defined by Amazon MQ in the IAM User Guide.

Resources

Administrators can use AWS JSON policies to specify who has access to what. That is, which principal can perform actions on what resources, and under what conditions.

The Resource JSON policy element specifies the object or objects to which the action applies. Statements must include either a Resource or a NotResource element. As a best practice, specify a resource using its Amazon Resource Name (ARN). You can do this for actions that support a specific resource type, known as resource-level permissions.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

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

In the Amazon MQ, the primary AWS resources are an Amazon MQ message broker and its configuration. Amazon MQ brokers and configurations each have unique Amazon Resource Names (ARNs) associated with them, as shown in the following table.

<table>
<thead>
<tr>
<th>Resource Types</th>
<th>ARN</th>
<th>Condition Keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>brokers</td>
<td>arn:-opacity:mq:#{Region}:#{Account}:broker:#broker-id</td>
<td>aws:ResourceTag/ ${TagKey} (p. 153)</td>
</tr>
</tbody>
</table>
### Resource Types

<table>
<thead>
<tr>
<th>ARN</th>
<th>Condition Keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>arn:${Partition}:mq:${Region}:${Account}:configuration:${configuration-id}</td>
<td>aws:ResourceTag/${TagKey} (p. 153)</td>
</tr>
</tbody>
</table>

For more information about the format of ARNs, see Amazon Resource Names (ARNs) and AWS Service Namespaces.

For example, to specify the `i-1234567890abcdef0` broker in your statement, use the following ARN:

```
"Resource": "arn:aws:ec2:us-east-1:123456789012:broker/i-1234567890abcdef0"
```

To specify all brokers that belong to a specific account, use the wildcard (*):

```
"Resource": "arn:aws:ec2:us-east-1:123456789012:broker/*"
```

Some Amazon MQ actions, such as those for creating resources, cannot be performed on a specific resource. In those cases, you must use the wildcard (*).

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

The API action `CreateTags` requires both a broker and a configuration. To specify multiple resources in a single statement, separate the ARNs with commas.

```
"Resource": ["resource1", "resource2"
```

To see a list of Amazon MQ resource types and their ARNs, see Resources Defined by Amazon MQ in the IAM User Guide. To learn with which actions you can specify the ARN of each resource, see Actions Defined by Amazon MQ.

### Condition keys

Administrators can use AWS JSON policies to specify who has access to what. That is, which principal can perform actions on what resources, and under what conditions.

The Condition element (or Condition block) lets you specify conditions in which a statement is in effect. The Condition element is optional. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple Condition elements in a statement, or multiple keys in a single Condition element, AWS evaluates them using a logical AND operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical OR operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see IAM policy elements: variables and tags in the IAM User Guide.

AWS supports global condition keys and service-specific condition keys. To see all AWS global condition keys, see AWS global condition context keys in the IAM User Guide.

Amazon MQ does not define any service-specific condition keys, but supports using some global condition keys. To see a list of Amazon MQ condition keys, see the table below or Condition Keys for
Amazon MQ in the IAM User Guide. To learn with which actions and resources you can use a condition key, see Actions Defined by Amazon MQ.

### Condition Keys

<table>
<thead>
<tr>
<th>Condition Key</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>aws:RequestTag/${TagKey}</td>
<td>Filters actions based on the tags that are passed in the request.</td>
<td>String</td>
</tr>
<tr>
<td>aws:ResourceTag/${TagKey}</td>
<td>Filters actions based on the tags associated with the resource.</td>
<td>String</td>
</tr>
<tr>
<td>aws:TagKeys</td>
<td>Filters actions based on the tag keys that are passed in the request.</td>
<td>String</td>
</tr>
</tbody>
</table>

### Examples

To view examples of Amazon MQ identity-based policies, see Amazon MQ Identity-based policy examples (p. 154).

### Amazon MQ Resource-based policies

Currently, Amazon MQ doesn't support IAM authentication using resource-based permissions or resource-based policies.

### Authorization based on Amazon MQ tags

You can attach tags to Amazon MQ resources or pass tags in a request to Amazon MQ. To control access based on tags, you provide tag information in the condition element of a policy using the `mq:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

Amazon MQ supports policies based on tags. For instance, you could deny access to Amazon MQ resources that include a tag with the key `environment` and the value `production`:

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "mq:DeleteBroker",
                "mq:RebootBroker",
                "mq:DeleteTags"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/environment": "production"
                }
            }
        }
    ]
}
```

This policy will Deny the ability to delete or reboot an Amazon MQ broker that includes the tag `environment/production`.

For more information on tagging, see:
• Tagging resources (p. 140)
• Controlling Access Using IAM Tags

Amazon MQ IAM roles

An IAM role is an entity within your AWS account that has specific permissions.

Using Temporary Credentials with Amazon MQ

You can use temporary credentials to sign in with federation, assume an IAM role, or to assume a cross-account role. You obtain temporary security credentials by calling AWS STS API operations such as AssumeRole or GetFederationToken.

Amazon MQ supports using temporary credentials.

Service roles

This feature allows a service to assume a service role on your behalf. This role allows the service to access resources in other services to complete an action on your behalf. Service roles appear in your IAM account and are owned by the account. This means that an IAM administrator can change the permissions for this role. However, doing so might break the functionality of the service.

Amazon MQ supports service roles.

Amazon MQ Identity-based policy examples

By default, IAM users and roles don't have permission to create or modify Amazon MQ resources. They also can't perform tasks using the AWS Management Console, AWS CLI, or AWS API. An IAM administrator must create IAM policies that grant users and roles permission to perform specific API operations on the specified resources they need. The administrator must then attach those policies to the IAM users or groups that require those permissions.

To learn how to create an IAM identity-based policy using these example JSON policy documents, see Creating Policies on the JSON Tab in the IAM User Guide.

Topics
• Policy best practices (p. 154)
• Using the Amazon MQ console (p. 155)
• Allow users to view their own permissions (p. 155)

Policy best practices

Identity-based policies are very powerful. They determine whether someone can create, access, or delete Amazon MQ resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

• Get started using AWS managed policies – To start using Amazon MQ quickly, use AWS managed policies to give your employees the permissions they need. These policies are already available in your account and are maintained and updated by AWS. For more information, see Get started using permissions with AWS managed policies in the IAM User Guide.

• Grant least privilege – When you create custom policies, grant only the permissions required to perform a task. Start with a minimum set of permissions and grant additional permissions as necessary. Doing so is more secure than starting with permissions that are too lenient and then trying to tighten them later. For more information, see Grant least privilege in the IAM User Guide.
• **Enable MFA for sensitive operations** – For extra security, require IAM users to use multi-factor authentication (MFA) to access sensitive resources or API operations. For more information, see Using multi-factor authentication (MFA) in AWS in the IAM User Guide.

• **Use policy conditions for extra security** – To the extent that it's practical, define the conditions under which your identity-based policies allow access to a resource. For example, you can write conditions to specify a range of allowable IP addresses that a request must come from. You can also write conditions to allow requests only within a specified date or time range, or to require the use of SSL or MFA. For more information, see IAM JSON policy elements: Condition in the IAM User Guide.

### Using the Amazon MQ console

To access the Amazon MQ console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the Amazon MQ resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (IAM users or roles) with that policy.

To ensure that those entities can still use the Amazon MQ console, also attach the following AWS managed policy to the entities. For more information, see Adding Permissions to a User in the IAM User Guide:

AmazonMQReadOnlyAccess

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that you're trying to perform.

### Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"],
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ]
        }
    ]
}
```
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 (p. 156)
- Amazon MQ REST API permissions reference (p. 157)
- Resource-level permissions for Amazon MQ API actions (p. 158)

**IAM Permissions Required to Create an Amazon MQ Broker**

To create a broker, you must either use the **AmazonMQFullAccess** IAM policy or include the following EC2 permissions in your IAM policy.

The following custom policy is comprised of two statements (one conditional) which grant permissions to manipulate the resources which Amazon MQ requires to create an ActiveMQ broker.

**Important**
- The `ec2:CreateNetworkInterface` action is required to allow Amazon MQ to create an elastic network interface (ENI) in your account on your behalf.
- The `ec2:CreateNetworkInterfacePermission` action authorizes Amazon MQ to attach the ENI to an ActiveMQ broker.
- The `ec2:AuthorizedService` condition key ensures that ENI permissions can be granted only to Amazon MQ service accounts.

```json
{
  "Version": "2012-10-17",
  "Statement": [{
    "Action": [
      "mq:*",
      "ec2:*",
      "ec2:CreateNetworkInterface",
      "ec2:DeleteNetworkInterface",
      "ec2:DetachNetworkInterface",
      "ec2:DescribeInternetGateways",
      "ec2:DescribeNetworkInterfaces",
      "ec2:DescribeRouteTables",
      "ec2:DescribeSecurityGroups",
      "ec2:DescribeSubnets",
      "ec2:DescribeVpcs"
    ],
    "Effect": "Allow",
    "Resource": "*
  },
  {
    "Resource": "*
  }
}
```
"Action": [
   "ec2:CreateNetworkInterfacePermission",
   "ec2:DeleteNetworkInterfacePermission",
   "ec2:DescribeNetworkInterfacePermissions"
],
"Effect": "Allow",
"Resource": "*",
"Condition": {
   "StringEquals": {
      "ec2:AuthorizedService": "mq.amazonaws.com"
   }
}]
}

For more information, see Step 2: create an IAM user and get your AWS credentials (p. 2) and Never Modify or Delete the Amazon MQ Elastic Network Interface (p. 69).

**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**

<table>
<thead>
<tr>
<th>Amazon MQ REST APIs</th>
<th>Required Permissions</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateBroker</td>
<td>mq:CreateBroker</td>
</tr>
<tr>
<td>CreateConfiguration</td>
<td>mq:CreateConfiguration</td>
</tr>
<tr>
<td>CreateTags</td>
<td>mq:CreateTags</td>
</tr>
<tr>
<td>CreateUser</td>
<td>mq:CreateUser</td>
</tr>
<tr>
<td>DeleteBroker</td>
<td>mq:DeleteBroker</td>
</tr>
<tr>
<td>DeleteUser</td>
<td>mq:DeleteUser</td>
</tr>
<tr>
<td>DescribeBroker</td>
<td>mq:DescribeBroker</td>
</tr>
<tr>
<td>DescribeConfiguration</td>
<td>mq:DescribeConfiguration</td>
</tr>
<tr>
<td>DescribeConfigurationRevision</td>
<td>mq:DescribeConfigurationRevision</td>
</tr>
<tr>
<td>DescribeUser</td>
<td>mq:DescribeUser</td>
</tr>
<tr>
<td>ListBrokers</td>
<td>mq:ListBrokers</td>
</tr>
<tr>
<td>ListConfigurationRevisions</td>
<td>mq:ListConfigurationRevisions</td>
</tr>
<tr>
<td>ListConfigurations</td>
<td>mq:ListConfigurations</td>
</tr>
<tr>
<td>ListTags</td>
<td>mq:ListTags</td>
</tr>
<tr>
<td>ListUsers</td>
<td>mq:ListUsers</td>
</tr>
<tr>
<td>RebootBroker</td>
<td>mq:RebootBroker</td>
</tr>
<tr>
<td>UpdateBroker</td>
<td>mq:UpdateBroker</td>
</tr>
<tr>
<td>UpdateConfiguration</td>
<td>mq:UpdateConfiguration</td>
</tr>
<tr>
<td>UpdateUser</td>
<td>mq:UpdateUser</td>
</tr>
</tbody>
</table>
Resource-level permissions for Amazon MQ API actions

The term resource-level permissions refers to the ability to specify the resources on which users are allowed to perform actions. Amazon MQ has partial support for resource-level permissions. For certain Amazon MQ actions, you can control when users are allowed to use those actions based on conditions that have to be fulfilled, or specific resources that users are allowed to use.

The following table describes the Amazon MQ API actions that currently support resource-level permissions, as well as the supported resources, resource ARNs, and condition keys for each action.

**Important**

If an Amazon MQ API action is not listed in this table, then it does not support resource-level permissions. If an Amazon MQ API action does not support resource-level permissions, you can grant users permission to use the action, but you have to specify a * wildcard for the resource element of your policy statement.

<table>
<thead>
<tr>
<th>API Action</th>
<th>Resource Types (*required)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateConfiguration</td>
<td>configurations*</td>
</tr>
<tr>
<td>CreateTags</td>
<td>brokers, configurations</td>
</tr>
<tr>
<td>CreateUser</td>
<td>brokers*</td>
</tr>
<tr>
<td>DeleteBroker</td>
<td>brokers*</td>
</tr>
<tr>
<td>DeleteUser</td>
<td>brokers*</td>
</tr>
<tr>
<td>DescribeBroker</td>
<td>brokers*</td>
</tr>
<tr>
<td>DescribeConfiguration</td>
<td>configurations*</td>
</tr>
<tr>
<td>DescribeConfigurationRevision</td>
<td>configurations*</td>
</tr>
<tr>
<td>DescribeUser</td>
<td>brokers*</td>
</tr>
<tr>
<td>ListConfigurationRevisions</td>
<td>configurations*</td>
</tr>
<tr>
<td>ListConfigurationRevisions</td>
<td>configurations*</td>
</tr>
<tr>
<td>ListTags</td>
<td>brokers, configurations</td>
</tr>
<tr>
<td>ListUsers</td>
<td>brokers*</td>
</tr>
<tr>
<td>RebootBroker</td>
<td>brokers*</td>
</tr>
<tr>
<td>UpdateBroker</td>
<td>brokers*</td>
</tr>
<tr>
<td>UpdateConfiguration</td>
<td>configurations*</td>
</tr>
<tr>
<td>UpdateUser</td>
<td>brokers*</td>
</tr>
</tbody>
</table>

AWS managed policies for Amazon MQ

To add permissions to users, groups, and roles, it is easier to use AWS managed policies than to write policies yourself. It takes time and expertise to create IAM customer managed policies that provide your team with only the permissions they need. To get started quickly, you can use our AWS managed policies. These policies cover common use cases and are available in your AWS account. For more information about AWS managed policies, see AWS managed policies in the IAM User Guide.
AWS services maintain and update AWS managed policies. You can't change the permissions in AWS managed policies. Services occasionally add additional permissions to an AWS managed policy to support new features. This type of update affects all identities (users, groups, and roles) where the policy is attached. Services are most likely to update an AWS managed policy when a new feature is launched or when new operations become available. Services do not remove permissions from an AWS managed policy, so policy updates won't break your existing permissions.

Additionally, AWS supports managed policies for job functions that span multiple services. For example, the ViewOnlyAccess AWS managed policy provides read-only access to many AWS services and resources. When a service launches a new feature, AWS adds read-only permissions for new operations and resources. For a list and descriptions of job function policies, see AWS managed policies for job functions in the IAM User Guide.

AWS managed policy: AmazonMQServiceRolePolicy

You can't attach AmazonMQServiceRolePolicy to your IAM entities. This policy is attached to a service-linked role that allows Amazon MQ to perform actions on your behalf. For more information about this permission policy and the actions it allows Amazon MQ to perform, see the section called “Service-linked role permissions for Amazon MQ” (p. 160).

Amazon MQ updates to AWS managed policies

View details about updates to AWS managed policies for Amazon MQ since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the Amazon MQ Document history (p. 206) page.

<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon MQ started tracking changes</td>
<td>Amazon MQ started tracking changes for its AWS managed policies.</td>
<td>May 5, 2021</td>
</tr>
</tbody>
</table>

Using service-linked roles for Amazon MQ

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

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

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

For information about other services that support service-linked roles, see AWS services that work with IAM and look for the services that have Yes in the Service-Linked Role column. Choose a Yes with a link to view the service-linked role documentation for that service.
Service-linked role permissions for Amazon MQ

Amazon MQ uses the service-linked role named `AWSServiceRoleForAmazonMQ` – Amazon MQ uses this service-linked role to call AWS services on your behalf.

The AWSServiceRoleForAmazonMQ service-linked role trusts the following services to assume the role:

- `mq.amazonaws.com`

Amazon MQ uses the permission policy `AmazonMQServiceRolePolicy`, which is attached to the AWSServiceRoleForAmazonMQ service-linked role, to complete the following actions on the specified resources:

- **Action:** `ec2:CreateVpcEndpoint` on the `vpc` resource.
- **Action:** `ec2:CreateVpcEndpoint` on the `subnet` resource.
- **Action:** `ec2:CreateVpcEndpoint` on the `security-group` resource.
- **Action:** `ec2:CreateVpcEndpoint` on the `vpc-endpoint` resource.
- **Action:** `ec2:DescribeVpcEndpoints` on the `vpc` resource.
- **Action:** `ec2:DescribeVpcEndpoints` on the `subnet` resource.
- **Action:** `ec2:CreateTags` on the `vpc-endpoint` resource.
- **Action:** `logs:PutLogEvents` on the `log-group` resource.
- **Action:** `logs:DescribeLogStreams` on the `log-group` resource.
- **Action:** `logs:DescribeLogGroups` on the `log-group` resource.
- **Action:** `CreateLogStream` on the `log-group` resource.
- **Action:** `CreateLogGroup` on the `log-group` resource.

When you create an Amazon MQ for RabbitMQ broker, the `AmazonMQServiceRolePolicy` permission policy allows Amazon MQ to perform the following tasks on your behalf:

- Create a Amazon VPC endpoint for the broker using the Amazon VPC, subnet, and security-group you provide. You can use the endpoint created for your broker to connect to the broker via the RabbitMQ management console, the management API, or programatically.
- Create log groups, and publish broker logs to Amazon CloudWatch Logs.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeVpcEndpoints"
      ]
    }
  ]
}
```
Using service-linked roles

```
{
  "Effect": "Allow",
  "Action": [
    "ec2:CreateVpcEndpoint"
  ],
  "Resource": [
    "arn:aws:ec2:*:*:vpc/*",
    "arn:aws:ec2:*:*:subnet/**",
    "arn:aws:ec2:*:*:security-group/**"
  ]
},
{
  "Effect": "Allow",
  "Action": [
    "ec2:CreateVpcEndpoint"
  ],
  "Resource": [
    "arn:aws:ec2:*:*:vpc-endpoint/**
  ],
  "Condition": {
    "StringEquals": {
      "aws:RequestTag/AMQManaged": "true"
    }
  }
},
{
  "Effect": "Allow",
  "Action": [
    "ec2:CreateTags"
  ],
  "Resource": "arn:aws:ec2:*:*:vpc-endpoint/**",
  "Condition": {
    "StringEquals": {
      "ec2:CreateAction": "CreateVpcEndpoint"
    }
  }
},
{
  "Effect": "Allow",
  "Action": [
    "ec2:DeleteVpcEndpoints"
  ],
  "Resource": "arn:aws:ec2:*:*:vpc-endpoint/**",
  "Condition": {
    "StringEquals": {
      "ec2:ResourceTag/AMQManaged": "true"
    }
  }
},
{
  "Effect": "Allow",
  "Action": [
    "logs:PutLogEvents",
    "logs:DescribeLogStreams",
    "logs:DescribeLogGroups",
    "logs:CreateLogStream",
    "logs:CreateLogGroup"
  ],
  "Resource": [
    "arn:aws:logs:*:*:log-group:/aws/amazonmq/**"
  ]
}
```
You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see Service-Linked Role Permissions in the IAM User Guide.

Creating a service-linked role for Amazon MQ

You don't need to manually create a service-linked role. When you first create a broker, Amazon MQ creates a service-linked role to call AWS services on your behalf. All subsequent brokers that you create will use the same role and no new role is created.

Important
This service-linked role can appear in your account if you completed an action in another service that uses the features supported by this role. To learn more, see A New Role Appeared in My IAM Account.

If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account.

You can also use the IAM console to create a service-linked role with the Amazon MQ use case. In the AWS CLI or the AWS API, create a service-linked role with the mq.amazonaws.com service name. For more information, see Creating a service-linked role in the IAM User Guide. If you delete this service-linked role, you can use this same process to create the role again.

Editing a service-linked role for Amazon MQ

Amazon MQ does not allow you to edit the AWSServiceRoleForAmazonMQ service-linked role. However, you can edit the description of the role using IAM. For more information, see Editing a service-linked role in the IAM User Guide.

Deleting a service-linked role for Amazon MQ

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don’t have an unused entity that is not actively monitored or maintained. However, you must clean up the resources for your service-linked role before you can manually delete it.

Note
If the Amazon MQ service is using the role when you try to delete the resources, then the deletion might fail. If that happens, wait for a few minutes and try the operation again.

To delete Amazon MQ resources used by the AWSServiceRoleForAmazonMQ

- Delete your Amazon MQ brokers using the AWS Management Console, Amazon MQ CLI, or Amazon MQ API. For more information about deleting brokers, see ??? (p. 28).

To manually delete the service-linked role using IAM

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

Supported regions for Amazon MQ service-linked roles

Amazon MQ supports using service-linked roles in all of the regions where the service is available. For more information, see AWS Regions and Endpoints.
<table>
<thead>
<tr>
<th>Region name</th>
<th>Region identity</th>
<th>Support in Amazon MQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>US East (N. Virginia)</td>
<td>us-east-1</td>
<td>Yes</td>
</tr>
<tr>
<td>US East (Ohio)</td>
<td>us-east-2</td>
<td>Yes</td>
</tr>
<tr>
<td>US West (N. California)</td>
<td>us-west-1</td>
<td>Yes</td>
</tr>
<tr>
<td>US West (Oregon)</td>
<td>us-west-2</td>
<td>Yes</td>
</tr>
<tr>
<td>Asia Pacific (Mumbai)</td>
<td>ap-south-1</td>
<td>Yes</td>
</tr>
<tr>
<td>Asia Pacific (Osaka)</td>
<td>ap-northeast-3</td>
<td>Yes</td>
</tr>
<tr>
<td>Asia Pacific (Seoul)</td>
<td>ap-northeast-2</td>
<td>Yes</td>
</tr>
<tr>
<td>Asia Pacific (Singapore)</td>
<td>ap-southeast-1</td>
<td>Yes</td>
</tr>
<tr>
<td>Asia Pacific (Sydney)</td>
<td>ap-southeast-2</td>
<td>Yes</td>
</tr>
<tr>
<td>Asia Pacific (Tokyo)</td>
<td>ap-northeast-1</td>
<td>Yes</td>
</tr>
<tr>
<td>Canada (Central)</td>
<td>ca-central-1</td>
<td>Yes</td>
</tr>
<tr>
<td>Europe (Frankfurt)</td>
<td>eu-central-1</td>
<td>Yes</td>
</tr>
<tr>
<td>Europe (Ireland)</td>
<td>eu-west-1</td>
<td>Yes</td>
</tr>
<tr>
<td>Europe (London)</td>
<td>eu-west-2</td>
<td>Yes</td>
</tr>
<tr>
<td>Europe (Paris)</td>
<td>eu-west-3</td>
<td>Yes</td>
</tr>
<tr>
<td>South America (São Paulo)</td>
<td>sa-east-1</td>
<td>Yes</td>
</tr>
<tr>
<td>AWS GovCloud (US)</td>
<td>us-gov-west-1</td>
<td>No</td>
</tr>
</tbody>
</table>

**Troubleshooting Amazon MQ identity and access**

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

**Topics**
- I Am Not Authorized to Perform an Action in Amazon MQ (p. 163)
- I am not authorized to perform iam:PassRole (p. 164)
- I want to view my access keys (p. 164)
- I'm an administrator and want to allow others to access Amazon MQ (p. 164)
- I want to allow people outside of my AWS account to access my Amazon MQ resources (p. 165)

**I Am Not Authorized to Perform an Action in Amazon MQ**

If the AWS Management Console tells you that you're not authorized to perform an action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a `widget` but does not have `mq:GetWidget` permissions.
In this case, Mateo asks his administrator to update his policies to allow him to access the `my-example-widget` resource using the `mq:GetWidget` action.

**I am not authorized to perform iam:PassRole**

If you receive an error that you're not authorized to perform the `iam:PassRole` action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password. Ask that person to update your policies to allow you to pass a role to Amazon MQ.

Some AWS services allow you to pass an existing role to that service, instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in Amazon MQ. However, the action requires the service to have permissions granted by a service role. Mary does not have permissions to pass the role to the service.

**I want to view my access keys**

After you create your IAM user access keys, you can view your access key ID at any time. However, you can't view your secret access key again. If you lose your secret key, you must create a new access key pair.

Access keys consist of two parts: an access key ID (for example, `AKIAIOSFODNN7EXAMPLE`) and a secret access key (for example, `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLERKEY`). Like a user name and password, you must use both the access key ID and secret access key together to authenticate your requests. Manage your access keys as securely as you do your user name and password.

**Important**

- Do not provide your access keys to a third party, even to help find your canonical user ID. By doing this, you might give someone permanent access to your account.

When you create an access key pair, you are prompted to save the access key ID and secret access key in a secure location. The secret access key is available only at the time you create it. If you lose your secret access key, you must add new access keys to your IAM user. You can have a maximum of two access keys. If you already have two, you must delete one key pair before creating a new one. To view instructions, see Managing access keys in the *IAM User Guide*.

**I'm an administrator and want to allow others to access Amazon MQ**

To allow others to access Amazon MQ, you must create an IAM entity (user or role) for the person or application that needs access. They will use the credentials for that entity to access AWS. You must then attach a policy to the entity that grants them the correct permissions in Amazon MQ.

To get started right away, see Creating your first IAM delegated user and group in the *IAM User Guide*.
I want to allow people outside of my AWS account to access my Amazon MQ resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether Amazon MQ supports these features, see How Amazon MQ works with IAM (p. 150).
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the IAM User Guide.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the IAM User Guide.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the IAM User Guide.
- To learn the difference between using roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the IAM User Guide.

Compliance validation for Amazon MQ

Third-party auditors assess the security and compliance of Amazon MQ as part of multiple AWS compliance programs. These include SOC, PCI, HIPAA, and others.

To learn whether Amazon MQ or other AWS services are in scope of specific compliance programs, see AWS Services in Scope by Compliance Program. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security and Compliance Quick Start Guides – These deployment guides discuss architectural considerations and provide steps for deploying baseline environments on AWS that are security and compliance focused.
- Architecting for HIPAA Security and Compliance Whitepaper – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.

  **Note**
  Not all services are compliant with HIPAA.

- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- Evaluating Resources with Rules in the AWS Config Developer Guide – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.
- AWS Audit Manager – This AWS service helps you continuously audit your AWS usage to simplify how you manage risk and compliance with regulations and industry standards.
Resilience in Amazon MQ

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

Infrastructure security in Amazon MQ

As a managed service, Amazon MQ is protected by the AWS global network security procedures that are described in the Amazon Web Services: Overview of Security Processes whitepaper.

You use AWS published API calls to access Amazon MQ through the network. Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

Security best practices for Amazon MQ

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

Topics
- Prefer brokers without public accessibility (p. 166)
- Always configure an authorization map (p. 166)
- Block unnecessary protocols with VPC security groups (p. 167)

For more information about how Amazon MQ encrypts your data, as well as a list of supported protocols, see Data Protection (p. 143).

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 broker web console without public accessibility (p. 29) in this guide and How to Help Prepare for DDoS Attacks by Reducing Your Attack Surface on the AWS Security Blog.

Always configure an authorization map

Because ActiveMQ has no authorization map configured by default, any authenticated user can perform any action on the broker. Thus, it is a best practice to restrict permissions by group. For more information, see authorizationEntry (p. 111).
Important

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

Block unnecessary protocols with VPC security groups

To improve security, you should restrict the connections of unnecessary protocols and ports by properly configuring your Amazon VPC Security Group. For instance, to restrict access to most protocols while allowing access to OpenWire and the web console, you could allow access to only 61617 and 8162. This limits your exposure by blocking protocols you are not using, while allowing OpenWire and the web console to function normally.

Allow only the protocol ports that you are using.

- AMQP: 5671
- MQTT: 8883
- OpenWire: 61617
- STOMP: 61614
- WebSocket: 61619

For more information see:

- Configure Additional Broker Settings (p. 32)
- Security Groups for your VPC
- Default Security Group for Your VPC
- Working with Security Groups
Logging and monitoring Amazon MQ brokers

Monitoring is an important part of maintaining the reliability, availability, and performance of your AWS solutions. You should collect monitoring data from all of the parts of your AWS solution so that you can more easily debug a multi-point failure if one occurs. AWS provides several tools for monitoring your Amazon MQ resources and responding to potential incidents:

**Topics**
- Accessing CloudWatch metrics for Amazon MQ (p. 168)
- Monitoring Amazon MQ brokers using Amazon CloudWatch (p. 171)
- Logging Amazon MQ API calls using AWS CloudTrail (p. 180)
- Configuring Amazon MQ to publish logs to Amazon CloudWatch Logs (p. 183)

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. 171).

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.

For ActiveMQ brokers, CloudWatch monitors only the first 200 destinations.

For RabbitMQ brokers, CloudWatch monitors only the first 500 destinations, ordered by number of consumers.

**Topics**
- AWS Management Console (p. 168)
- AWS Command Line Interface (p. 170)
- Amazon CloudWatch API (p. 171)

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.

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, see Accessing CloudWatch metrics for Amazon MQ (p. 168).

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

- Average
- Minimum
- Maximum
- Sum

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

Topics

- Logging and monitoring Amazon MQ for ActiveMQ brokers (p. 171)
- Logging and monitoring Amazon MQ for RabbitMQ brokers (p. 176)

Logging and monitoring Amazon MQ for ActiveMQ brokers

Amazon MQ for ActiveMQ metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AmqpMaximumConnections</td>
<td>Count</td>
<td>The maximum number of clients you can connect to your broker using AMQP. For more information on connection quotas, see Quotas in Amazon MQ (p. 188).</td>
</tr>
<tr>
<td>BurstBalance</td>
<td>Percent</td>
<td>The percentage of burst credits remaining on the Amazon EBS volume used to persist message</td>
</tr>
<tr>
<td>Metric</td>
<td>Unit</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>data for throughput-optimized brokers. If this balance reaches zero, the IOPS provided by the Amazon EBS volume will decrease until the Burst Balance refills. For more information on how Burst Balances work in Amazon EBS, see: I/O Credits and Burst Performance.</td>
</tr>
</tbody>
</table>
| CpuCreditBalance          | Credits (vCPU-minutes) | **Important**  
This metric is available only for the mq.t2.micro broker instance type.  
CPU credit metrics are available only at five-minute intervals.  
The number of earned CPU credits that an instance has accrued since it was launched or started (including the number of launch credits). The credit balance is available for the broker instance to spend on bursts beyond the baseline CPU utilization.  
Credits are accrued in the credit balance after they're earned and removed from the credit balance after they're spent. The credit balance has a maximum limit. Once the limit is reached, any newly earned credits are discarded. |
<p>| CpuUtilization            | Percent           | The percentage of allocated Amazon EC2 compute units that the broker currently uses. |
| CurrentConnectionsCount   | Count             | The current number of active connections on the current broker. |
| EstablishedConnectionsCount| Count             | The total number of connections, active and inactive, that have been established on the broker. |
| HeapUsage                 | Percent           | The percentage of the ActiveMQ JVM memory limit that the broker currently uses. |</p>
<table>
<thead>
<tr>
<th>Metric</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InactiveDurableTopicSubscribersCount</td>
<td>Count</td>
<td>The number of inactive durable topic subscribers, up to a maximum of 2000.</td>
</tr>
<tr>
<td>JobSchedulerStorePercentUsage</td>
<td>Percent</td>
<td>The percentage of disk space used by the job scheduler store.</td>
</tr>
<tr>
<td>JournalFilesForFastRecoveryCount</td>
<td>Count</td>
<td>The number of journal files that will be replayed after a clean shutdown.</td>
</tr>
<tr>
<td>JournalFilesForFullRecoveryCount</td>
<td>Count</td>
<td>The number of journal files that will be replayed after an unclean shutdown.</td>
</tr>
<tr>
<td>MqttMaximumConnections</td>
<td>Count</td>
<td>The maximum number of clients you can connect to your broker using MQTT. For more information on connection quotas, see Quotas in Amazon MQ (p. 188).</td>
</tr>
<tr>
<td>NetworkConnectorConnectionCount</td>
<td>Count</td>
<td>The number of nodes connected to the broker in a network of brokers (p. 86) using NetworkConnector.</td>
</tr>
<tr>
<td>NetworkIn</td>
<td>Bytes</td>
<td>The volume of incoming traffic for the broker.</td>
</tr>
<tr>
<td>NetworkOut</td>
<td>Bytes</td>
<td>The volume of outgoing traffic for the broker.</td>
</tr>
<tr>
<td>OpenTransactionCount</td>
<td>Count</td>
<td>The total number of transactions in progress.</td>
</tr>
<tr>
<td>OpenwireMaximumConnections</td>
<td>Count</td>
<td>The maximum number of clients you can connect to your broker using OpenWire. For more information on connection quotas, see Quotas in Amazon MQ (p. 188).</td>
</tr>
<tr>
<td>StompMaximumConnections</td>
<td>Count</td>
<td>The maximum number of clients you can connect to your broker using STOMP. For more information on connection quotas, see Quotas in Amazon MQ (p. 188).</td>
</tr>
<tr>
<td>StorePercentUsage</td>
<td>Percent</td>
<td>The percent used by the storage limit. If this reaches 100, the broker will refuse messages.</td>
</tr>
<tr>
<td>TempPercentUsage</td>
<td>Percent</td>
<td>The percentage of available temporary storage used by non-persistent messages.</td>
</tr>
</tbody>
</table>
### Metric

<table>
<thead>
<tr>
<th>Metric</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TotalConsumerCount</td>
<td>Count</td>
<td>The number of message consumers subscribed to destinations on the current broker.</td>
</tr>
<tr>
<td>TotalMessageCount</td>
<td>Count</td>
<td>The number of messages stored on the broker.</td>
</tr>
<tr>
<td>TotalProducerCount</td>
<td>Count</td>
<td>The number of message producers active on destinations on the current broker.</td>
</tr>
<tr>
<td>VolumeReadOps</td>
<td>Count</td>
<td>The number of read operations performed on the Amazon EBS volume.</td>
</tr>
<tr>
<td>VolumeWriteOps</td>
<td>Count</td>
<td>The number of write operations performed on the Amazon EBS volume.</td>
</tr>
<tr>
<td>WsMaximumConnections</td>
<td>Count</td>
<td>The maximum number of clients you can connect to your broker using WebSocket. For more information on connection quotas, see [Quotas in Amazon MQ](p. 188).</td>
</tr>
</tbody>
</table>

### Dimensions for ActiveMQ broker metrics

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Broker</td>
<td>The name of the broker</td>
</tr>
</tbody>
</table>

**Note**  
A single-instance broker has the suffix -1. An active/standby broker for high availability has the suffixes -1 and -2 for its redundant pair.

### ActiveMQ destination (queue and topic) metrics

**Important**  
The following metrics include per-minute counts for the CloudWatch polling period.

- EnqueueCount
- ExpiredCount
- DequeueCount
- DispatchCount
- InFlightCount
For example, in a five-minute CloudWatch period, EnqueueCount has five count values, each for a one-minute portion of the period. The Minimum and Maximum statistics provide the lowest and highest per-minute value during the specified period.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConsumerCount</td>
<td>Count</td>
<td>The number of consumers subscribed to the destination.</td>
</tr>
<tr>
<td>EnqueueCount</td>
<td>Count</td>
<td>The number of messages sent to the destination, per minute.</td>
</tr>
<tr>
<td>EnqueueTime</td>
<td>Time (milliseconds)</td>
<td>The end-to-end latency from when a message arrives at a broker until it is delivered to a consumer.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note</strong> EnqueueTime does not measure the end-to-end latency from when a message is sent by a producer until it reaches the broker, nor the latency from when a message is received by a broker until it is acknowledged by the broker. Rather, EnqueueTime is the number of milliseconds from the moment a message is received by the broker until it is successfully delivered to a consumer.</td>
</tr>
<tr>
<td>ExpiredCount</td>
<td>Count</td>
<td>The number of messages that couldn't be delivered because they expired, per minute.</td>
</tr>
<tr>
<td>DispatchCount</td>
<td>Count</td>
<td>The number of messages sent to consumers, per minute.</td>
</tr>
<tr>
<td>DequeueCount</td>
<td>Count</td>
<td>The number of messages acknowledged by consumers, per minute.</td>
</tr>
<tr>
<td>InFlightCount</td>
<td>Count</td>
<td>The number of messages sent to consumers that have not been acknowledged.</td>
</tr>
<tr>
<td>ReceiveCount</td>
<td>Count</td>
<td>The number of messages that have been received from the remote broker for a duplex network connector.</td>
</tr>
</tbody>
</table>
### Metric

<table>
<thead>
<tr>
<th>Metric</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MemoryUsage</td>
<td>Percent</td>
<td>The percentage of the memory limit that the destination currently uses.</td>
</tr>
<tr>
<td>ProducerCount</td>
<td>Count</td>
<td>The number of producers for the destination.</td>
</tr>
<tr>
<td>QueueSize</td>
<td>Count</td>
<td>The number of messages in the queue.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Important</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>This metric applies only to queues.</td>
</tr>
<tr>
<td>TotalEnqueueCount</td>
<td>Count</td>
<td>The total number of messages that have been sent to the broker.</td>
</tr>
<tr>
<td>TotalDequeueCount</td>
<td>Count</td>
<td>The total number of messages that have been consumed by clients.</td>
</tr>
</tbody>
</table>

**Note**

TotalEnqueueCount and TotalDequeueCount metrics include messages for advisory topics. For more information about advisory topic messages, see the [ActiveMQ documentation](#).

### Dimensions for ActiveMQ destination (queue and topic) metrics

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Broker</td>
<td>The name of the broker.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong></td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>Topic or Queue</td>
<td>The name of the topic or queue.</td>
</tr>
<tr>
<td>NetworkConnector</td>
<td>The name of the network connector.</td>
</tr>
</tbody>
</table>

### Logging and monitoring Amazon MQ for RabbitMQ brokers

#### RabbitMQ broker metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExchangeCount</td>
<td>Count</td>
<td>The total number of exchanges configured on the broker.</td>
</tr>
</tbody>
</table>
### Metric | Unit | Description
--- | --- | ---
QueueCount | Count | The total number of queues configured on the broker.
ConnectionCount | Count | The total number of connections established on the broker.
ChannelCount | Count | The total number of channels established on the broker.
ConsumerCount | Count | The total number of consumers connected to the broker.
MessageCount | Count | The total number of messages in the queues.
    **Note**
The number produced is the total sum of ready and unacknowledged messages on the broker.
MessageReadyCount | Count | The total number of ready messages in the queues.
MessageUnacknowledgedCount | Count | The total number of unacknowledged messages in the queues.
PublishRate | Count | The rate at which messages are published to the broker.
The number produced represents the number of messages per second at the time of sampling.
ConfirmRate | Count | The rate at which the RabbitMQ server is confirming published messages. You can compare this metric with PublishRate to better understand how your broker is performing.
The number produced represents the number of messages per second at the time of sampling.
AckRate | Count | The rate at which messages are being acknowledged by consumers.
The number produced represents the number of messages per second at the time of sampling.
### Metric | Unit | Description
--- | --- | ---
SystemCpuUtilization | Percent | The percentage of allocated Amazon EC2 compute units that the broker currently uses. For cluster deployments, this value represents the aggregate of all three RabbitMQ nodes' corresponding metric values.

RabbitMQMemLimit | Bytes | The RAM limit for a RabbitMQ broker. For cluster deployments, this value represents the aggregate of all three RabbitMQ nodes' corresponding metric values.

RabbitMQMemUsed | Bytes | The volume of RAM used by a RabbitMQ broker. For cluster deployments, this value represents the aggregate of all three RabbitMQ nodes' corresponding metric values.

RabbitMQDiskFreeLimit | Bytes | The disk limit for a RabbitMQ broker. For cluster deployments, this value represents the aggregate of all three RabbitMQ nodes' corresponding metric values. This metric is different per instance size. For more information about Amazon MQ instance types, see the section called “Amazon MQ for RabbitMQ instance types” (p. 139).

RabbitMQDiskFree | Bytes | The total volume of free disk space available in a RabbitMQ broker. When disk usage goes above its limit, the cluster will block all producer connections. For cluster deployments, this value represents the aggregate of all three RabbitMQ nodes' corresponding metric values.

RabbitMQFdUsed | Count | Number of file descriptors used. For cluster deployments, this value represents the aggregate of all three RabbitMQ nodes' corresponding metric values.
# Dimensions for RabbitMQ broker metrics

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Broker</td>
<td>The name of the broker.</td>
</tr>
</tbody>
</table>

## RabbitMQ node metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SystemCpuUtilization</td>
<td>Percent</td>
<td>The percentage of allocated Amazon EC2 compute units that the broker currently uses.</td>
</tr>
<tr>
<td>RabbitMQMemLimit</td>
<td>Bytes</td>
<td>The RAM limit for a RabbitMQ node.</td>
</tr>
<tr>
<td>RabbitMQMemUsed</td>
<td>Bytes</td>
<td>The volume of RAM used by a RabbitMQ node. When memory use goes above the limit, the cluster will block all producer connections.</td>
</tr>
<tr>
<td>RabbitMQDiskFreeLimit</td>
<td>Bytes</td>
<td>The disk limit for a RabbitMQ node. This metric is different per instance size. For more information about Amazon MQ instance types, see the section called “Amazon MQ for RabbitMQ instance types” (p. 139).</td>
</tr>
<tr>
<td>RabbitMQDiskFree</td>
<td>Bytes</td>
<td>The total volume of free disk space available in a RabbitMQ node. When disk usage goes above its limit, the cluster will block all producer connections.</td>
</tr>
<tr>
<td>RabbitMQFdUsed</td>
<td>Count</td>
<td>Number of file descriptors used.</td>
</tr>
</tbody>
</table>

## Dimensions for RabbitMQ node metrics

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node</td>
<td>The name of the node.</td>
</tr>
</tbody>
</table>

**Note**

A node name consists of two parts: a prefix (usually rabbit) and a hostname. For example, rabbit@ip-10-0-0-230.us-west-2.compute.internal is a node name with the prefix rabbit and
Logging API calls using CloudTrail

### RabbitMQ queue metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConsumerCount</td>
<td>Count</td>
<td>The number of consumers subscribed to the queue.</td>
</tr>
<tr>
<td>MessageReadyCount</td>
<td>Counter</td>
<td>The number of messages that are currently available to be delivered.</td>
</tr>
<tr>
<td>MessageUnacknowledgedCount</td>
<td>Count</td>
<td>The number of messages for which the server is awaiting acknowledgement.</td>
</tr>
<tr>
<td>MessageCount</td>
<td>Counter</td>
<td>The total number of MessageReadyCount and MessageUnacknowledgedCount (also known as queue depth).</td>
</tr>
</tbody>
</table>

**Dimensions for RabbitMQ queue metrics**

**Note**
Amazon MQ for RabbitMQ will not publish metrics for virtual hosts and queues with names containing blank spaces, tabs or other non-ASCII characters. For more information about dimension names, see Dimension in the Amazon CloudWatch API Reference.

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Queue</td>
<td>The name of the queue.</td>
</tr>
<tr>
<td>Virtual host</td>
<td>Name of the virtual host.</td>
</tr>
</tbody>
</table>

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. 183).

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 (***):
Amazon MQ Developer Guide
Example Amazon MQ Log File Entry

- 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.

```json
{
  "eventVersion": "1.06",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AKIAIOSFODNN7EXAMPLE",
    "arn": "arn:aws:iam::111122223333:user/AmazonMqConsole",
    "accountId": "111122223333",
    "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
    "userName": "AmazonMqConsole"
  },
  "eventSource": "amazonmq.amazonaws.com",
  "eventName": "CreateBroker",
  "awsRegion": "us-west-2",
  "sourceIPAddress": "203.0.113.0",
  "userAgent": "PostmanRuntime/7.1.5",
  "requestParameters": {
    "engineVersion": "5.15.9",
    "deploymentMode": "ACTIVE_STANDBY_MULTI_AZ",
    "maintenanceWindowStartTime": {
      "dayOfWeek": "THURSDAY",
      "timeOfDay": "22:45",
      "timeZone": "America/Los_Angeles"
    },
    "engineType": "ActiveMQ",
    "hostInstanceType": "mq.m5.large",
    "users": [
      {
        "username": "MyUsername123",
        "password": "***",
```
Configuring Amazon MQ to publish 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 ActiveMQ broker configuration (p. 97) errors. For more information about CloudWatch Logs, see the Amazon CloudWatch Logs User Guide

Topics
- Configuring Amazon MQ for ActiveMQ logs (p. 183)
- Configuring Amazon MQ for RabbitMQ logs (p. 187)

Configuring Amazon MQ for ActiveMQ logs

To allow Amazon MQ to publish logs to CloudWatch Logs, you must add a permission to your Amazon MQ user (p. 184) and also configure a resource-based policy for Amazon MQ (p. 185) before you create or restart the broker.

The following describes the steps to configure CloudWatch logs for your ActiveMQ brokers.
Understanding the structure of logging in CloudWatch Logs

You can enable general and audit logging when you configure advanced broker settings (p. 32) 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. 84) or an active/standby broker (p. 85), 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 IAM user who creates or reboots the broker has the logs:CreateLogGroup permission.

Important

If you don't add the CreateLogGroup permission to your Amazon MQ user before the user creates or reboots the broker, Amazon MQ doesn't create the log group.

The following example IAM-based policy grants permission for logs:CreateLogGroup for users to whom this policy is attached.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "logs:CreateLogGroup",
            "Resource": "arn:aws:logs:*:*:log-group:/aws/amazonmq/*"
        }
    ]
}
```
Note
Here, the term user refers to IAM Users and not Amazon MQ users, which are created when a new broker is configured. For more information regarding setting up IAM users and configuring IAM policies, please refer to the Identity Management Overview section of the IAM User Guide.

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

Configure a resource-based policy for Amazon MQ

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

To allow Amazon MQ to publish logs to your CloudWatch Logs log group, configure a resource-based policy to give Amazon MQ access to the following CloudWatch Logs API actions:

• CreateLogStream – Creates a CloudWatch Logs log stream for the specified log group.
• PutLogEvents – Delivers events to the specified CloudWatch Logs log stream.

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

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": { "Service": "mq.amazonaws.com" },
      "Action": [ "logs:CreateLogStream", "logs:PutLogEvents" ],
      "Resource": "arn:aws:logs:*:*:log-group:/aws/amazonmq/*"
    }
  ]
}
```

This resource-based policy must be configured by using the AWS CLI as shown by the following command. In the example, replace us-east-1 with your own information.

```
```

Note
Because this example uses the /aws/amazonmq/ prefix, you need to configure the resource-based policy only once per AWS account, per region.

Cross-service confused deputy prevention

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

We recommend using the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in your Amazon MQ resource-based policy to limit CloudWatch Logs access to one or more specified brokers.

**Note**

If you use both global condition context keys, the `aws:SourceAccount` value and the account in the `aws:SourceArn` value must use the same account ID when used in the same policy statement.

The following example demonstrates a resource-based policy that limits CloudWatch Logs access to a single Amazon MQ broker.

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Principal": {
            "Service": "mq.amazonaws.com"
         },
         "Action": [
            "logs:CreateLogStream",
            "logs:PutLogEvents"
         ],
         "Resource": "arn:aws:logs:*:*:log-group:/aws/amazonmq/**",
         "Condition": {
            "StringEquals": {
               "aws:SourceAccount": "123456789012",
            }
         }
      }
   ]
}
```

You can also configure your resource-based policy to limit CloudWatch Logs access to all brokers in an account, as shown in the following.

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Principal": {
            "Service": ["mq.amazonaws.com"]
         },
         "Action": ["logs:CreateLogStream", "logs:PutLogEvents"],
         "Resource": "arn:aws:logs:*:*:log-group:/aws/amazonmq/**",
         "Condition": {
            "ArnLike": {
               "aws:SourceArn": "arn:aws:mq:*:123456789012:broker:*"
            },
            "StringEquals": {
               "aws:SourceAccount": "123456789012"
            }
         }
      }
   ]
}
```
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. 184) 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. 185). This allows your broker to publish its logs.

Configuring Amazon MQ for RabbitMQ logs

When you enable CloudWatch logging for your RabbitMQ brokers, Amazon MQ uses a service-linked role to publish general logs to CloudWatch. If no Amazon MQ service-linked role exists when you first create a broker, Amazon MQ will automatically create one. All subsequent RabbitMQ brokers will use the same service-linked role to publish logs to CloudWatch.

For more information about service-linked roles, see Using service-linked roles in the AWS Identity and Access Management User Guide. For more information about how Amazon MQ uses service-linked roles, see the section called "Using service-linked roles" (p. 159).

Note
Audit logging is not supported for RabbitMQ brokers.
Quotas in Amazon MQ

This topic lists quotas within Amazon MQ. Many of the following quotas can be changed for specific AWS accounts. To request an increase for a limit, see AWS Service Quotas in the Amazon Web Services General Reference.

Topics

• Brokers (p. 188)
• Configurations (p. 188)
• Users (p. 189)
• Data Storage (p. 189)
• API Throttling (p. 190)

Brokers

The following table lists quotas related to Amazon MQ brokers.

<table>
<thead>
<tr>
<th>Limit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Broker name</td>
<td>• Must be unique in your AWS account.</td>
</tr>
<tr>
<td></td>
<td>• Must be 1-50 characters long.</td>
</tr>
<tr>
<td></td>
<td>• Must contain only characters specified in the ASCII Printable Character Set.</td>
</tr>
<tr>
<td></td>
<td>• Can contain only alphanumeric characters, dashes, periods, underscores, and tildes (- . _ ~).</td>
</tr>
<tr>
<td>Number of brokers, per region</td>
<td>20</td>
</tr>
</tbody>
</table>
| Wire-level connections per smaller broker  | Important
|                                            | Does not apply to RabbitMQ brokers.                                         |
|                                            | 100 for mq.*.micro instance type brokers.                                  |
| Wire-level connections per larger broker   | Important
|                                            | Does not apply to RabbitMQ brokers.                                         |
|                                            | 1,000 for mq.*.*large instance type brokers.                               |
| Security groups per broker                 | 5                                                                           |
| ActiveMQ destinations (queues, and topics) | CloudWatch monitors only the first 200 destinations.                        |
| monitored in CloudWatch                    |                                                                             |
| RabbitMQ destinations (queues) monitored in| CloudWatch monitors only the first 500 destinations, ordered by number of consumers. |
| CloudWatch                                 |                                                                             |
| Tags per broker                            | 50                                                                          |

Configurations

The following table lists quotas related to Amazon MQ configurations.
Important
Does not apply to RabbitMQ brokers.

<table>
<thead>
<tr>
<th>Limit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration name</td>
<td>• Must be 1-150 characters long. \n• Must contain only characters specified in the ASCII Printable Character Set. \n• Can contain only alphanumeric characters, dashes, periods, underscores, and tildes (−. _).</td>
</tr>
<tr>
<td>Revisions per configuration</td>
<td>300</td>
</tr>
</tbody>
</table>

Users

The following table lists quotas related to Amazon MQ ActiveMQ broker users.

Important
Does not apply to RabbitMQ brokers.

<table>
<thead>
<tr>
<th>Limit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Username</td>
<td>• Must be 1-100 characters long. \n• Must contain only characters specified in the ASCII Printable Character Set. \n• Can contain only alphanumeric characters, dashes, periods, underscores, and tildes (−. _). \n• Must not contain commas (,).</td>
</tr>
<tr>
<td>Password</td>
<td>• Must be 12-250 characters long. \n• Must contain only characters specified in the ASCII Printable Character Set. \n• Must contain at least 4 unique characters. \n• Must not contain commas (,).</td>
</tr>
<tr>
<td>Users per broker (simple auth)</td>
<td>250</td>
</tr>
<tr>
<td>Groups per user (simple auth)</td>
<td>20</td>
</tr>
</tbody>
</table>

Data Storage

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

<table>
<thead>
<tr>
<th>Limit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Storage capacity per smaller broker</td>
<td>20 GB for mq.*.micro instance type brokers. For more information regarding Amazon MQ instance types, see Broker instance types (p. 136).</td>
</tr>
</tbody>
</table>
### API Throttling

The following throttling quotas are aggregated per AWS account, across all Amazon MQ APIs to maintain service bandwidth. For more information about Amazon MQ APIs, see the [Amazon MQ REST API Reference](https://docs.aws.amazon.com/mq/latest/developer-guide/amazon-mq-api-reference.html).

**Important**

These quotas don't apply to Amazon MQ for ActiveMQ or Amazon MQ for RabbitMQ broker messaging APIs. For example, Amazon MQ doesn't throttle the sending or receiving of messages.

<table>
<thead>
<tr>
<th>Limit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Storage capacity per larger broker</td>
<td>200 GB for ( \text{mq.*.*large} ) instance type brokers. For more information regarding Amazon MQ instance types, see <a href="https://docs.aws.amazon.com/mq/latest/developer-guide/amazon-mq-instance-type.html">Broker instance types</a> (p. 136).</td>
</tr>
</tbody>
</table>
| Job scheduler usage limit per broker backed by Amazon EBS (p. 82) | **Important**

Does not apply to RabbitMQ brokers.

50 GB. For more information about job scheduler usage, see [JobSchedulerUsage](https://activemq.apache.org/scheduler.html) in the Apache ActiveMQ API Documentation. |
| Temporary storage capacity per smaller broker. | **Important**

Does not apply to RabbitMQ brokers.

5 GB for \( \text{mq.*.micro} \) instance type brokers. |
| Temporary storage capacity per larger broker. | **Important**

Does not apply to RabbitMQ brokers.

50 GB for \( \text{mq.*.*large} \) instance type brokers. |

<table>
<thead>
<tr>
<th>API burst limit</th>
<th>API rate limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>15</td>
</tr>
</tbody>
</table>
Troubleshooting Amazon MQ

This section describes common issues you might encounter when using Amazon MQ brokers, and the steps you can take to resolve them.

Contents

- Troubleshooting: General (p. 191)
  - I can't connect to my broker web console or endpoints. (p. 192)
  - My broker is running, and I can verify connectivity using telnet, but my clients are unable to connect and are returning SSL exceptions. (p. 195)
  - I created a broker but broker creation failed. (p. 195)
  - My broker restarted and I'm not sure why. (p. 196)
- Troubleshooting: Amazon MQ for ActiveMQ (p. 196)
  - I can't see general or audit logs for my broker in CloudWatch Logs even though I've activated logging. (p. 196)
  - After broker restart or maintenance window, I can't connect to my broker even though the status is RUNNING. Why? (p. 197)
  - I see some of my clients connecting to the broker, while others are unable to connect. (p. 197)
  - I'm seeing exception org.apache.jasper.JasperException: An exception occurred processing JSP page on the ActiveMQ console when performing operations. (p. 198)
- Troubleshooting: Amazon MQ for RabbitMQ (p. 198)
  - I can't see metrics for my queues or virtual hosts in CloudWatch. (p. 198)
  - How do I enable plugins in Amazon MQ for RabbitMQ? (p. 199)
  - I'm unable to change Amazon VPC configuration for the broker. (p. 199)
- Troubleshooting: Amazon MQ status codes (p. 199)
  - Amazon MQ for RabbitMQ: High memory alarm (p. 199)
    - Diagnosing high memory alarm using the RabbitMQ web console (p. 200)
    - Diagnosing high memory alarm using Amazon MQ metrics (p. 200)
    - Addressing high memory alarm (p. 201)
    - Reducing the number of connections and channels (p. 202)
    - Addressing paused queue synchronizations in cluster deployments (p. 202)
    - Addressing restart loops in single-instance brokers (p. 203)
    - Preventing high memory alarms (p. 203)

Troubleshooting: General

Use the information in this section to help you diagnose common issues you might encounter when working with Amazon MQ brokers, such as issues connecting to your broker, and broker reboots.

Contents

- I can't connect to my broker web console or endpoints. (p. 192)
- My broker is running, and I can verify connectivity using telnet, but my clients are unable to connect and are returning SSL exceptions. (p. 195)
- I created a broker but broker creation failed. (p. 195)
I can't connect to my broker web console or endpoints.

If you're experiencing issues connecting to your broker using the web console or wire-level endpoints, we recommend the following steps.

1. Check whether you're attempting to connect to your broker from behind a firewall. You might need to configure the firewall to allow access to your broker.
2. Check whether you're trying to connect to your broker using a FIPS endpoint. Amazon MQ only supports FIPS endpoints when using API operations, and not for wire-level connections to the broker instance itself.
3. Check if the Public Accessibility option for your broker is set to Yes. If this is set to No, check your subnet's network Access Control List (ACL) rules. If you've created custom network ACLs, you might need to change the network ACL rules to provide access to your broker. For more information about Amazon VPC networking, see Enabling internet access in the Amazon VPC User Guide
4. Check your broker's Security Group rules. Make sure that you are allowing connections to the following ports:

   Note
   The following ports are grouped according to engine types because Amazon MQ for ActiveMQ and Amazon MQ for RabbitMQ use different ports for connections.

   **Amazon MQ for ActiveMQ**
   - Web console – Port 8162
   - OpenWire – Port 61617
   - AMQP – Port 5671
   - STOPM – Port 61614
   - MQTT – Port 8883
   - WSS – Port 61619

   **Amazon MQ for RabbitMQ**
   - Web console and management API – Port 443 and 15671
   - AMQP – Port 5671

5. Run the following network connectivity tests for your broker engine type.

   Note
   For brokers without public accessibility, run the tests from an Amazon EC2 instance within the same Amazon VPC as your Amazon MQ broker and evaluate the responses.

**Amazon MQ for ActiveMQ**

1. Open a new terminal or command line window.
2. Run the following `nslookup` command to query your broker DNS record. For active/standby (p. 85) deployments, test both the active and standby endpoints. The active/standby endpoints are identified with a suffix, -1 or -2 added to the unique broker ID. Replace the endpoint with your information.

   ```
   $ nslookup b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-west-2.amazonaws.com
   ```
I can’t connect to my broker web console or endpoints.

If the query succeeds, you will see an output similar to the following.

Non-authoritative answer:
Server: dns-resolver-corp-sfo-1.sfo.corp.amazon.com
Address: 172.10.123.456
Name: ec2-12-345-123-45.us-west-2.compute.amazonaws.com
Address: 12.345.123.45
Aliases: b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-west-2.amazonaws.com

The resolved IP address should match the IP addresses provided in the Amazon MQ console. This indicates that the domain name is resolving correctly on the DNS server, and you can move on to the next step.

3. Run the following `telnet` command to test the network path for your broker. Replace the endpoint with your information. Replace `port` with port number 8162 for the web console, or other wire-level ports to test additional protocols as needed.

   Note
   For active/standby deployments, you will receive a `Connect failed` error message if you run `telnet` with the standby endpoint. This is expected, as the standby instance itself is running, but the ActiveMQ process is not running and does not have access to the broker's Amazon EFS storage volume. Run the command for both -1 and -2 endpoints to ensure you test both the active and the standby instances.

   $ telnet b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-west-2.amazonaws.com port

   For the active instance, you will see an output similar to the following.

   Connected to b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-west-2.amazonaws.com. Escape character is '^]'.

4. Do one of the following.
   - If the `telnet` command succeeds, check the `EstablishedConnectionsCount` (p. 171) metric and confirm that the broker has not reached the maximum `Wire-level connection limit` (p. 188). You can also confirm if the limit has been reached by reviewing the broker General logs. If this metric is greater than zero, then there is at least one client currently connected to the broker. If the metric shows zero connections, then perform the `telnet` path test again and wait at least one minute before disconnecting, as broker metrics are published every minute.
   - If the `telnet` command fails, check the status of your broker's elastic network interface, and confirm that the status is `in-use`. Create an Amazon VPC flow log for each instance's network interface, and review the generated flow logs. Look for the broker's IP addresses when you run the `telnet` command, and confirm the connection packets are `ACCEPTED`, including a return packet. For more information, and to see a flow log example, see Flow log record examples in the Amazon VPC Developer Guide.

5. Run the following `curl` command to check connectivity to the ActiveMQ admin web console.

   $ curl https://b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-west-2.amazonaws.com:8162/index.html

   If the command succeeds, the output should be an HTML document similar to the following.

   <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
   <html>
I can't connect to my broker web console or endpoints.

Amazon MQ for RabbitMQ

To test your Amazon MQ for RabbitMQ broker's network connectivity

1. Open a new terminal or command line window.
2. Run the following `nslookup` command to query your broker DNS record. Replace the endpoint with your information.

   ```
   $ nslookup b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-west-2.amazonaws.com
   ``

   If the query succeeds, you will see an output similar to the following.

   Non-authoritative answer:
   Server:  dns-resolver-corp-sfo-1.sfo.corp.amazon.com
   Address:  172.10.123.456

   Name:    rabbit-broker-1c23e456ca78-b9000123b4ebba5.elb.us-west-2.amazonaws.com
   Addresses:  52.12.345.678
               52.23.234.56
               41.234.567.890
               54.123.45.678
   Aliases:  b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-west-2.amazonaws.com

3. Run the following `telnet` command to test the network path for your broker. Replace the endpoint with your information. You can replace `port` with port 443 for the web console, and 5671 to test the wire-level AMQP connection.

   ```
   $ telnet b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-west-2.amazonaws.com port
   ``

   If the command succeeds, you'll see an output similar to the following.

   Connected to b-1234a5b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-west-2.amazonaws.com.
   Escape character is '^]'.

   **Note**
   The telnet connection will close automatically after a few seconds.

4. Do one of the following.
   - If the `telnet` command succeeds, check the `ConnectionCount` metric and confirm that the broker has not reached the value set in the `max-connections` default policy. You can also confirm if the limit has been reached by reviewing the broker `Connection.log` log group. If this metric is greater than zero, there is at least one client currently connected to the broker. If the metric shows zero connections, then perform the `telnet` path test again. You may need to repeat this process if the connection closes before your broker has published new connection metrics to CloudWatch. Metrics are published every minute.
   - For brokers without public accessibility, if the `telnet` command fails, check the status of your broker's elastic network interfaces, and confirm that the status is in-use. Create an Amazon VPC flow log for each network interface, and review the generated flow logs. Look for the broker's private IP addresses when you the `telnet` command was invoked.
and confirm the connection packets are ACCEPTED, including a return packet. For more information, and to see a flow log example, see Flow log record examples in the Amazon VPC Developer Guide.

Note
This step does not apply to Amazon MQ for RabbitMQ brokers with public accessibility.

5. Run the following curl command to check connectivity to the RabbitMQ admin web console.

```
$ curl https://b-123456b6-78cd-901e-2fgh-3i45j6k178l9-1.mq.us-west-2.amazonaws.com:443/index.html
```

If the command succeeds, the output should be an HTML document similar to the following.

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="X-UA-Compatible" content="IE=edge" />
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>RabbitMQ Management</title>
...
```

My broker is running, and I can verify connectivity using telnet, but my clients are unable to connect and are returning SSL exceptions.

Your broker endpoint certificate may have been updated during the broker maintenance window (p. 21). Amazon MQ broker certificates are rotated every 12 months and updated about a month before a certificate expires.

We recommend using the Amazon root certificate authority (CA) in Amazon Trust Services to authenticate against in your clients' trust store. All Amazon MQ broker certificates are signed with this root CA. By using an Amazon root CA, you will no longer need to download the new Amazon MQ broker certificate every time there is a certificate update on the broker.

Note
With Amazon MQ for ActiveMQ, the certificate is placed on the instance, while with Amazon MQ for RabbitMQ, the certificate is on the Network Load Balancer.

I created a broker but broker creation failed.

If your broker is in a CREATION_FAILED status, do the following.

- Check your IAM permissions. To create a broker must either use the AWS managed IAM policy AmazonMQFullAccess or have the correct set of Amazon EC2 permissions in your custom IAM policy. To learn more about the required Amazon EC2 permissions you need, see IAM permissions required to create an Amazon MQ broker (p. 156).
- Check if the subnet you are choosing for your broker is in a shared Amazon Virtual Private Cloud (VPC). To create an Amazon MQ broker in a shared Amazon VPC, you must create it in the account that owns the Amazon VPC.
My broker restarted and I'm not sure why.

If your broker has restarted automatically, it may be due to one of the following reasons.

- Your broker may have restarted because of a scheduled weekly maintenance window. Periodically, Amazon MQ performs maintenance to the hardware, operating system, or the engine software of a message broker. The duration of the maintenance varies, but can last up to two hours, depending on the operations that are scheduled for your message broker. Brokers might restart at any point during the two hour maintenance window. For more information about broker maintenance windows, see the section called “Maintaining a broker” (p. 21).

- Your broker instance type might not be suitable to your application workload. For example, running a production workload on a mq.t2.micro might result in the broker running out of resources. High CPU utilization, or high broker memory usage can cause a broker to unexpectedly restart. To see how much CPU and memory is being utilized by your broker, use the following CloudWatch metrics for your engine type.
  - Amazon MQ for ActiveMQ – Check CpuUtilization for the percentage of allocated Amazon EC2 compute units that the broker currently uses. Check HeapUsage for the percentage of the ActiveMQ JVM memory limit that the broker currently uses.
  - Amazon MQ for RabbitMQ – Check SystemCpuUtilization for the percentage of allocated Amazon EC2 compute units that the broker currently uses. Check RabbitMQMemUsed for the volume of RAM used in Bytes, and divide by RabbitMQMemLimit for the percentage of memory used by the RabbitMQ node.

For more information about broker instance types and how to choose the right instance type for your workload, see Broker instance types (p. 136).

Troubleshooting: Amazon MQ for ActiveMQ

Use the information in this section to help you diagnose and resolve common issues you might encounter when working with Amazon MQ for ActiveMQ brokers.

Contents

- I can't see general or audit logs for my broker in CloudWatch Logs even though I've activated logging. (p. 196)
- After broker restart or maintenance window, I can't connect to my broker even though the status is RUNNING. Why? (p. 197)
- I see some of my clients connecting to the broker, while others are unable to connect. (p. 197)
- I'm seeing exception org.apache.jasper.JasperException: An exception occurred processing JSP page on the ActiveMQ console when performing operations. (p. 198)

I can't see general or audit logs for my broker in CloudWatch Logs even though I've activated logging.

If you're unable to view logs for your broker in CloudWatch Logs, do the following.

1. Check if the IAM user who creates or reboots the broker has the logs:CreateLogGroup permission. If you don't add the CreateLogGroup permission to a user before the user creates or reboots the broker, Amazon MQ will not create the log group.

2. Check if you have configured a resource-based policy to allow Amazon MQ to publish logs to CloudWatch Logs. To allow Amazon MQ to publish logs to your CloudWatch Logs log group, configure a resource-based policy to give Amazon MQ access to the following CloudWatch Logs API actions:
- **CreateLogStream** – Creates a CloudWatch Logs log stream for the specified log group.
- **PutLogEvents** – Delivers events to the specified CloudWatch Logs log stream.

For more information about configuring Amazon MQ for ActiveMQ to publish logs to CloudWatch Logs, see Configuring logging.

**After broker restart or maintenance window, I can't connect to my broker even though the status is RUNNING. Why?**

You might be encountering connection issues after a broker restart you initiated, after a scheduled maintenance window is completed, or in a failure event, where the standby instance is activated. In either case, connection issues following a broker restart are most likely caused by unusually large numbers of messages persisted in your broker's Amazon EFS or Amazon EBS storage volume. During a restart, Amazon MQ moves persisted messages from storage to broker memory. To confirm this diagnosis, you can monitor the following metrics on CloudWatch for your Amazon MQ for ActiveMQ broker:

- **StoragePercentUsage** — Large percentages at or close to 100 percent can cause the broker to refuse connections.
- **JournalFilesForFullRecovery** — Indicates the number of journal files that will be replayed following an unclean shutdown and restart. An increasing, or consistently higher than one, value indicates unresolved transactions that can cause connection issues after restart.
- **OpenTransactionCount** — A number larger than zero following a restart indicates that the broker will attempt to store previously consumed messages, as a result causing connection issues.

To resolve this issue, we recommend resolving your XA transactions with either a `rollback()` or a `commit()`. For more information, and to see a code example of resolving XA transactions using `rollback()`, see recovering XA transactions (p. 73).

**I see some of my clients connecting to the broker, while others are unable to connect.**

If your broker is in the **RUNNING** status and some clients are able to connect to the broker successfully, while others are unable to do so, you may have reached the wire-level connections (p. 188) limit for the broker. To verify that you've reached the wire-level connections limit, do the following:

- Check the *general* broker logs for your Amazon MQ for ActiveMQ broker in CloudWatch Logs. If the limit has been reached, you will see **Reached Maximum Connections** in the broker logs. For more information on CloudWatch Logs for Amazon MQ for ActiveMQ brokers, see the section called "Understanding the structure of logging in CloudWatch Logs" (p. 184).

Once the wire-level connections limit is reached, the broker will actively refuse additional incoming connections. To resolve this issue, we recommend upgrading your broker instance type. For more information on choosing the best instance type for your workload, see Broker instance types (p. 136).

If you've confirmed that the number of your wire-level connections is less than the broker connection limit, the issue might be related to rebooting clients. Check your broker logs for numerous and frequent entries of **... Inactive for longer than 600000 ms - removing ....** The log entry is indicative of rebooting clients or connectivity issues. This effect is more evident when clients connect to
the broker via a Network Load Balancer (NLB) with clients that frequently disconnect and reconnect to the broker. This is more typically observed in container based clients.

Check your client-side logs for further details. The broker will clean up inactive TCP connections after 600000 ms, and free up the connection socket.

I'm seeing exception org.apache.jasper.JasperException: An exception occurred processing JSP page on the ActiveMQ console when performing operations.

If you are using simple authentication and configuring AuthorizationPlugin for queue and topic authorization, make sure to use the AuthorizationEntries element in your XML configuration file, and allow the activemq.webconsole group permission to all queues and topics. This ensures that the ActiveMQ web console can communicate with the ActiveMQ broker.

The following example AuthorizationEntry grants read and write permissions for all queues and topics to the activemq.webconsole group.

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

Similarly when integrating your broker with LDAP, make sure to grant permission for the amazonmq-console-admins group. For more information about LDAP integration, see the section called "How LDAP integration works" (p. 51).

**Troubleshooting: Amazon MQ for RabbitMQ**

Use the information in this section to help you diagnose and resolve common issues you might encounter when working with Amazon MQ for RabbitMQ brokers.

**Contents**

- I can’t see metrics for my queues or virtual hosts in CloudWatch. (p. 198)
- How do I enable plugins in Amazon MQ for RabbitMQ? (p. 199)
- I'm unable to change Amazon VPC configuration for the broker. (p. 199)

**I can’t see metrics for my queues or virtual hosts in CloudWatch.**

If you’re unable to view metrics for your queues or virtual hosts in CloudWatch, check if your queue or virtual host names contain any blank spaces, tabs, or other non-ASCII characters.

Amazon MQ cannot publish metrics for virtual hosts and queues with names containing blank spaces, tabs or other non-ASCII characters.
How do I enable plugins in Amazon MQ for RabbitMQ?

Amazon MQ for RabbitMQ currently only supports the RabbitMQ management, shovel, federation, consistent-hash exchange plugin, which are enabled by default. For more information on using supported plugins, see the section called “Plugins” (p. 131).

I'm unable to change Amazon VPC configuration for the broker.

Amazon MQ does not support changing Amazon VPC configuration after your broker is created. Please note that you will need to create a new broker with the new Amazon VPC configuration and update the client connection URL with the new broker connection URL.

Troubleshooting: Amazon MQ status codes

Amazon MQ returns an exception for certain API operations, such as RebootBroker, if your broker is in an unhealthy state and requires recovery. The exceptions include specific status codes that help you identify the root causes, and address the issue in order to recover your broker.

Use the following list of topics to identify the status code you have received, and learn more about the steps we recommend to resolve your issue.

Status codes
- Amazon MQ for RabbitMQ: High memory alarm (p. 199)

Amazon MQ for RabbitMQ: High memory alarm

RabbitMQ will raise a high memory alarm when the broker’s memory usage, identified by CloudWatch metric RabbitMQMemUsed, exceeds the memory limit, identified by RabbitMQMemLimit. RabbitMQMemLimit is set by Amazon MQ and has been specifically tuned considering the memory available for each host instance type.

An Amazon MQ for RabbitMQ broker that has raised a high memory alarm will block all clients that are publishing messages. Due to high memory usage, your broker might also experience other issues that complicate diagnosis and resolution of the alarm.

Single-instance brokers that can't complete start-up due to high memory usage might enter a restart loop, during which, interactions with the broker are limited. In cluster deployments, queues might experience paused synchronization of messages between replicas on different nodes. Paused queue syncs prevent consumption of messages from queues and must be addressed separately while resolving the memory alarm.

Amazon MQ will not restart a broker experiencing a high memory alarm and will return an exception for RebootBroker API operations as long as the broker continues to raise the alarm.

Use the information in this section to help you diagnose and resolve RabbitMQ high memory alarms raised by your broker.

Topics
Diagnosing high memory alarm using the RabbitMQ web console

The RabbitMQ web console can generate and display detailed memory usage information for each node. You can find this information by doing the following:

1. Sign in to AWS Management Console and open your broker's RabbitMQ web console.
2. On the RabbitMQ console, on the **Overview** page, choose the name of a node from the **Nodes** list.
3. On the node detail page, choose **Memory details** to expand the section to view the node's memory usage information.

The memory usage information that RabbitMQ provides in the web console can help you determine which resources might be consuming too much memory and contributing to the high memory alarm. For more information about the memory usage details available via the RabbitMQ web console, see **Reasoning About Memory Use** on the RabbitMQ Server Documentation website.

Diagnosing high memory alarm using Amazon MQ metrics

Amazon MQ enables metrics for your broker by default. You can view your broker metrics (p. 168) by accessing the CloudWatch console, or by using the CloudWatch API. The following metrics are useful when diagnosing the RabbitMQ high memory alarm.

<table>
<thead>
<tr>
<th><strong>Amazon MQ CloudWatch metric</strong></th>
<th><strong>Reason for high memory use</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>MessageCount</td>
<td>Messages are stored in memory until they are consumed or discarded. A high message count might indicate overutilization of resources and can lead to a high memory alarm.</td>
</tr>
<tr>
<td>QueueCount</td>
<td>Queues are stored in memory, and a high number of queues can lead to a high memory alarm.</td>
</tr>
<tr>
<td>ConnectionCount</td>
<td>Client connections utilize memory, and too many simultaneous connections can lead to a high memory alarm.</td>
</tr>
<tr>
<td>ChannelCount</td>
<td>Similar to connections, channels established using each connection are also stored</td>
</tr>
</tbody>
</table>
Reason for high memory use

Amazon MQ recommendation

<table>
<thead>
<tr>
<th>Reason for high memory use</th>
<th>Amazon MQ recommendation</th>
</tr>
</thead>
</table>
| The number of messages in the queues is too high. | Do any of the following:

- Consume messages published to the queues.
- Purge messages from queues.
- Delete the queues from your broker. |
| The number of queues configured on the broker is too high. | Reduce the number of queues. |
| The number of connections established on the broker is too high. | Reduce the number of connections. For more information, see the section called “Reducing the number of connections and channels” (p. 202). |
| The number of channels established on the broker is too high. | Reduce the number of channels. For more information see, the section called “Reducing the |
Reason for high memory use | Amazon MQ recommendation
---|---
The number of consumers connected to the broker is too high. | Reduce the number of consumers connected to the broker.
The message publishing rate is too high. | Reduce the rate at which publishers send messages to the broker.
The client connection attempt rate is too high. | Reduce the frequency at which clients attempt to connect to the broker in order to publish or consume messages, or configure the broker.

Reducing the number of connections and channels

Connections to your Amazon MQ for RabbitMQ broker can be closed either by your client applications, or by manually closing them using the RabbitMQ web console. To close a connection using the RabbitMQ web console do the following.

1. Sign in to AWS Management Console and open your broker’s RabbitMQ web console.
2. On the RabbitMQ console, choose the **Connections** tab.
3. On the **Connections** page, under **All connections**, choose the name of the connection you want to close from the list.
4. On the connection details page, choose **Close this connection** to expand the section, then choose **Force Close**. Optionally, you can replace the default text for **Reason** with your own description. Amazon MQ for RabbitMQ will return the reason you specify to the client when you close the connection.
5. Choose **OK** on the dialog box to confirm and close the connection.

When you close a connection, any channels associated with closed connection will also be closed.

**Note**
Your client applications may be configured to automatically re-establish connections to the broker after they are closed. In this case, closing connections from the broker web console will not be sufficient for reducing connection or channel counts.

For brokers without public access, you can temporarily block connections by denying inbound traffic on the appropriate message protocol port, for example, port 5671 for AMQP connections. You can block the port in the security group that you provided to Amazon MQ when creating the broker. For more information on modifying your security group, see Adding rules to a security group in the Amazon VPC User Guide.

**Addressing paused queue synchronizations in cluster deployments**

While addressing RabbitMQ’s high memory alarms, you may find that messages on one or multiple queues cannot be consumed. These queues may be in the process of synchronizing messages between nodes, during which the respective queues become unavailable for publishing and consuming. Queue
synchronizations might become paused due to the high memory alarm, and even contribute to the memory alarm.

For information about stopping and retrying paused queue syncs, see the section called “Resolving paused queue sync” (p. 65).

### Addressing restart loops in single-instance brokers

An Amazon MQ for RabbitMQ single-instance broker that raises a high memory alarm is at risk of becoming unavailable if it restarts and does not have enough memory to start up. This can cause RabbitMQ to enter a restart loop and prevent any further interactions with the broker until the issue is resolved. If your broker is in a restart loop, you will not be able to apply the Amazon MQ recommended actions previously described in this section to resolve the high memory alarm.

To recover your broker, we recommend upgrading to a larger instance type with more memory. Unlike in cluster deployments, you can upgrade a single-instance broker while it is experiencing a high memory alarm because there are no queue synchronizations to perform between nodes during a restart.

### Preventing high memory alarms

For each contributing factor you identify, we recommends the following set of actions for preventing and reducing the occurrence of RabbitMQ high memory alarms.

<table>
<thead>
<tr>
<th>Reason high memory use</th>
<th>Amazon MQ recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td>The number of messages in the queues is too high.</td>
<td>Do the following:</td>
</tr>
<tr>
<td></td>
<td>• Enable lazy queues (p. 74).</td>
</tr>
<tr>
<td></td>
<td>• Set, or reduce the queue depth limit (p. 125).</td>
</tr>
<tr>
<td>The number of queues configured on the broker is too high.</td>
<td>Set, or reduce the queue count limit (p. 125).</td>
</tr>
<tr>
<td>The number of connections established on the broker is too high.</td>
<td>Set, or reduce the connection count limit (p. 125).</td>
</tr>
<tr>
<td>The number of channels established on the broker is too high.</td>
<td>Set a maximum number of channels per connection on client applications.</td>
</tr>
<tr>
<td>The number of consumers connected to the broker is too high.</td>
<td>Set a small consumer pre-fetch limit (p. 125).</td>
</tr>
<tr>
<td>The client connection attempt rate is too high.</td>
<td>Use longer-lived connections to reduce the number and frequency of connection attempts.</td>
</tr>
</tbody>
</table>

After your broker's memory alarm has been resolved, you can upgrade your host instance type to an instance with additional resources. For information on how to update your broker's instance type see UpdateBrokerInput in the Amazon MQ REST API Reference.

For a complete listing of broker instance types, see the section called “Amazon MQ for RabbitMQ instance types” (p. 139).
Related resources

Amazon MQ resources

The following table lists useful resources for working with Amazon MQ.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon MQ REST API Reference</td>
<td>Descriptions of REST resources, example requests, HTTP methods, schemas, parameters, and the errors that the service returns.</td>
</tr>
<tr>
<td>Amazon MQ in the AWS CLI Command Reference</td>
<td>Descriptions of the AWS CLI commands that you can use to work with message brokers.</td>
</tr>
<tr>
<td>Amazon MQ in the AWS CloudFormation User Guide</td>
<td>The <code>AWS::Amazon MQ::Broker</code> 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.</td>
</tr>
<tr>
<td></td>
<td>The <code>AWS::Amazon MQ::Configuration</code> resource lets you create Amazon MQ configurations, add configuration changes or modify users, and return information about the specified configuration.</td>
</tr>
<tr>
<td>Regions and Endpoints</td>
<td>Information about Amazon MQ regions and endpoints</td>
</tr>
<tr>
<td>Product Page</td>
<td>The primary web page for information about Amazon MQ.</td>
</tr>
<tr>
<td>Discussion Forum</td>
<td>A community-based forum for developers to discuss technical questions related to Amazon MQ.</td>
</tr>
<tr>
<td>AWS Premium Support Information</td>
<td>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</td>
</tr>
</tbody>
</table>

Amazon MQ for ActiveMQ resources

The following table lists useful resources for working with Apache ActiveMQ.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache ActiveMQ Getting Started Guide</td>
<td>The official documentation of Apache ActiveMQ.</td>
</tr>
</tbody>
</table>
## 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 for RabbitMQ resources

The following table lists useful resources for working with RabbitMQ.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>The RabbitMQ Getting Started Guide</strong></td>
<td>The official documentation of RabbitMQ.</td>
</tr>
<tr>
<td><strong>RabbitMQ Client Libraries and Developer Tools</strong></td>
<td>A guide to the officially supported client libraries and developer tools for working with RabbitMQ using a variety of programming languages and platforms.</td>
</tr>
<tr>
<td><strong>RabbitMQ Best Practices</strong></td>
<td>CloudAMQP's guide on best practices and recommendations for working with RabbitMQ.</td>
</tr>
</tbody>
</table>
# 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. 217).

<table>
<thead>
<tr>
<th>Date</th>
<th>Documentation Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>January 18, 2022</td>
<td>Amazon MQ for RabbitMQ now supports RabbitMQ version 3.8.26. For more information about the fixes and features in this release, see the following:</td>
</tr>
</tbody>
</table>
|                     | • RabbitMQ 3.8.26 release notes on the RabbitMQ server GitHub repository  
• RabbitMQ changelog                                                                                                                                                                                                                                                                                       |
|                     | For more information about supported Amazon MQ for RabbitMQ versions and broker upgrades, see Managing Amazon MQ for RabbitMQ engine versions (p. 134).                                                                                                                                                                           |
| January 13, 2022    | Amazon MQ introduces the RABBITMQ_MEMORY_ALARM status code to inform you when your broker has raised a high memory alarm and is in an unhealthy state. Amazon MQ provides detailed information and recommendations to help you diagnose, resolve and prevent high memory alarms. For more information, see the following: |
|                     | • the section called “RABBITMQ_MEMORY_ALARM” (p. 199)                                                                                                                                                                                                                                                                                         |
| January 6, 2022     | When you configure CloudWatch Logs for Amazon MQ for ActiveMQ brokers, Amazon MQ supports using the aws:SourceArn and aws:SourceAccount global condition context keys in IAM resource-based policies to prevent the confused deputy problem. For more information, see the following:                     |
|                     | • the section called “Cross-service confused deputy prevention” (p. 185)                                                                                                                                                                                                                                                                                     |
| December 20, 2021   | Amazon MQ for ActiveMQ introduces a set of new metrics, allowing you to monitor the maximum number of connections you can make to your broker using different supported transport protocols, as well as an additional new metric that allows you to monitor the number of nodes connected to your broker in a network of brokers (p. 86). For more information, see the following: |
|                     | • the section called “Logging and monitoring Amazon MQ for ActiveMQ brokers” (p. 171)                                                                                                                                                                                                                                                                               |
| November 16, 2021   | Amazon MQ for RabbitMQ now supports RabbitMQ version 3.8.23. For more information about the fixes and features in this release, see the following:                                                                                                                                                                                                                   |
|                     | • RabbitMQ 3.8.23 release notes on the RabbitMQ server GitHub repository  
• RabbitMQ changelog                                                                                                                                                                                                                                                                                       |
<p>|                     | For more information about supported Amazon MQ for RabbitMQ versions and broker upgrades, see Managing Amazon MQ for RabbitMQ engine versions (p. 134).                                                                                                                                                                           |</p>
<table>
<thead>
<tr>
<th>Date</th>
<th>Documentation Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>October 12, 2021</td>
<td>Amazon MQ now supports ActiveMQ 5.16.3, a minor engine version release. For more information, see the following:</td>
</tr>
<tr>
<td></td>
<td>• ActiveMQ 5.16.3 Release Page</td>
</tr>
<tr>
<td></td>
<td>• Managing Amazon MQ for ActiveMQ engine versions (p. 115)</td>
</tr>
<tr>
<td></td>
<td>• Upgrading an Amazon MQ broker engine version (p. 24)</td>
</tr>
<tr>
<td></td>
<td>• Working with Spring XML configuration files (p. 97)</td>
</tr>
<tr>
<td>September 8, 2021</td>
<td>Amazon MQ for RabbitMQ now supports RabbitMQ version 3.8.22.</td>
</tr>
<tr>
<td></td>
<td>This release includes a fix for an issue with queues using per-message TTL (time to live), identified in the previously supported version, RabbitMQ 3.8.17. We recommend upgrading your existing brokers to version 3.8.22.</td>
</tr>
<tr>
<td></td>
<td>For more information about the fixes and features in this release, see the following:</td>
</tr>
<tr>
<td></td>
<td>• RabbitMQ 3.8.22 release notes on the RabbitMQ server GitHub repository</td>
</tr>
<tr>
<td></td>
<td>• RabbitMQ changelog</td>
</tr>
<tr>
<td></td>
<td>For more information about supported Amazon MQ for RabbitMQ versions and broker upgrades, see Managing Amazon MQ for RabbitMQ engine versions (p. 134)</td>
</tr>
<tr>
<td>August 25, 2021</td>
<td>Amazon MQ for RabbitMQ has temporarily disabled RabbitMQ engine version 3.8.17 due to an issue identified with queues using per-message time-to-live (TTL). We recommend using version 3.8.11.</td>
</tr>
<tr>
<td>July 29, 2021</td>
<td>Amazon MQ for RabbitMQ now supports RabbitMQ version 3.8.17. For more information about the fixes and features contained in this update, see the following:</td>
</tr>
<tr>
<td></td>
<td>• RabbitMQ 3.8.17 release notes on the RabbitMQ server GitHub repository</td>
</tr>
<tr>
<td></td>
<td>• RabbitMQ changelog</td>
</tr>
<tr>
<td></td>
<td>• Managing Amazon MQ for RabbitMQ engine versions (p. 134)</td>
</tr>
<tr>
<td>July 16, 2021</td>
<td>You can now adjust the maintenance window of an Amazon MQ broker using the AWS Management Console, AWS CLI, or the Amazon MQ API. To learn more about broker maintenance windows, see the following.</td>
</tr>
<tr>
<td></td>
<td>• Maintaining an Amazon MQ broker (p. 21)</td>
</tr>
<tr>
<td>July 6, 2021</td>
<td>Amazon MQ for RabbitMQ introduces support for the Consistent Hash exchange type. Consistent Hash exchanges route messages to queues based on a hash value calculated from the routing key of a message. For more information, see the following:</td>
</tr>
<tr>
<td></td>
<td>• Consistent Hash exchange plugin (p. 132)</td>
</tr>
<tr>
<td></td>
<td>• RabbitMQ Consistent Hash Exchange Type on the RabbitMQ GitHub repository</td>
</tr>
<tr>
<td>Date</td>
<td>Documentation Update</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>June 7, 2021</td>
<td>Amazon MQ now supports ActiveMQ 5.16.2, a new major engine version release. For more information, see the following:</td>
</tr>
<tr>
<td></td>
<td>• ActiveMQ 5.16.2 Release Page</td>
</tr>
<tr>
<td></td>
<td>• Managing Amazon MQ for ActiveMQ engine versions (p. 115)</td>
</tr>
<tr>
<td></td>
<td>• Upgrading an Amazon MQ broker engine version (p. 24)</td>
</tr>
<tr>
<td></td>
<td>• Working with Spring XML configuration files (p. 97)</td>
</tr>
<tr>
<td>May 26, 2021</td>
<td>Amazon MQ for RabbitMQ is now available in the China (Beijing) and China (Ningxia) Regions. For information on available regions, see AWS Regions and Endpoints.</td>
</tr>
<tr>
<td>May 18, 2021</td>
<td>Amazon MQ for RabbitMQ implements broker defaults.</td>
</tr>
<tr>
<td></td>
<td>When you first create a broker, Amazon MQ creates a set of broker policies and vhost limits based on the instance type and deployment mode you choose, in order to optimize the broker's performance. For more information, see the following:</td>
</tr>
<tr>
<td></td>
<td>• Amazon MQ for RabbitMQ broker defaults (p. 125)</td>
</tr>
<tr>
<td>May 5, 2021</td>
<td>Amazon MQ now supports ActiveMQ 5.15.15. For more information, see the following:</td>
</tr>
<tr>
<td></td>
<td>• ActiveMQ 5.15.15 Release Page</td>
</tr>
<tr>
<td></td>
<td>• Managing Amazon MQ for ActiveMQ engine versions (p. 115)</td>
</tr>
<tr>
<td></td>
<td>• Working with Spring XML configuration files (p. 97)</td>
</tr>
<tr>
<td>May 5, 2021</td>
<td>Amazon MQ started tracking changes to AWS managed policies. For more information, see the following:</td>
</tr>
<tr>
<td></td>
<td>• the section called &quot;AWS managed policies&quot; (p. 158)</td>
</tr>
<tr>
<td>April 14, 2021</td>
<td>Amazon MQ is now available in the China (Beijing) and China (Ningxia) Regions. For information on available regions, see AWS Regions and Endpoints.</td>
</tr>
<tr>
<td>April 7, 2021</td>
<td>Amazon MQ now supports RabbitMQ 3.8.11. For more information about the fixes and features contained in this update, see the following:</td>
</tr>
<tr>
<td></td>
<td>• RabbitMQ 3.8.11 release notes on the RabbitMQ server GitHub repository</td>
</tr>
<tr>
<td></td>
<td>• RabbitMQ changelog</td>
</tr>
<tr>
<td></td>
<td>• Managing Amazon MQ for RabbitMQ engine versions (p. 134)</td>
</tr>
<tr>
<td>April 1, 2021</td>
<td>Amazon MQ is now available in the Asia Pacific (Osaka) Region. For information about available regions, see Amazon MQ regions and endpoints.</td>
</tr>
<tr>
<td>Date</td>
<td>Documentation Update</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>December 21, 2020</td>
<td>Amazon MQ now supports ActiveMQ 5.15.14. For more information, see the following:</td>
</tr>
<tr>
<td></td>
<td>• ActiveMQ 5.15.14 Release Notes</td>
</tr>
<tr>
<td></td>
<td>• Managing Amazon MQ for ActiveMQ engine versions (p. 115)</td>
</tr>
<tr>
<td></td>
<td>• Working with Spring XML configuration files (p. 97)</td>
</tr>
<tr>
<td></td>
<td>• <strong>Important</strong> Due to a known Apache ActiveMQ issue in this release, the new <strong>Pause Queue</strong> button in the ActiveMQ web console cannot be used with Amazon MQ for ActiveMQ brokers. For more information about this issue, see AMQ-8104.</td>
</tr>
<tr>
<td>November 4, 2020</td>
<td>Amazon MQ now supports RabbitMQ, a popular open source message broker. This enables you to migrate your existing RabbitMQ message brokers to AWS without having to rewrite code.</td>
</tr>
<tr>
<td></td>
<td>Amazon MQ for RabbitMQ manages both individual and clustered message brokers and handles tasks like provisioning the infrastructure, setting up the broker, and updating the software.</td>
</tr>
<tr>
<td></td>
<td>• Amazon MQ supports RabbitMQ 3.8.6. For more information about supported engine versions, see the section called “Version management” (p. 134).</td>
</tr>
<tr>
<td></td>
<td>• The <strong>AWS Free Tier</strong> includes up to 750 hours of a single-instance <code>mq.t3.micro</code> broker and up to 20GB of storage per month for one year. For more information about supported instance types, see Broker instance types (p. 136).</td>
</tr>
<tr>
<td></td>
<td>• With Amazon MQ for RabbitMQ, you can access your brokers using AMQP 0-9-1, and with any language supported by the RabbitMQ client libraries. For more information about supported protocols and cipher suites, see the section called “Amazon MQ for RabbitMQ protocols” (p. 146).</td>
</tr>
<tr>
<td></td>
<td>• Amazon MQ for RabbitMQ is available in all regions that Amazon MQ is currently available. To learn more about all of the available regions, see the AWS Region Table.</td>
</tr>
<tr>
<td></td>
<td>To get started with using Amazon MQ, create a broker, and connect a JVM-based application to your RabbitMQ broker, see the section called “Creating and connecting to a RabbitMQ broker” (p. 11).</td>
</tr>
<tr>
<td>October 22, 2020</td>
<td>Amazon MQ supports ActiveMQ 5.15.13. For more information, see the following:</td>
</tr>
<tr>
<td></td>
<td>• ActiveMQ 5.15.13 Release Notes</td>
</tr>
<tr>
<td></td>
<td>• Managing Amazon MQ for ActiveMQ engine versions (p. 115)</td>
</tr>
<tr>
<td></td>
<td>• Working with Spring XML configuration files (p. 97)</td>
</tr>
<tr>
<td>September 30, 2020</td>
<td>Amazon MQ is now available in the Europe (Milan) Region. For information about available regions, see Amazon MQ regions and endpoints.</td>
</tr>
<tr>
<td>July 27, 2020</td>
<td>You can authenticate Amazon MQ users using the credentials stored in your Active Directory or other LDAP server. You can also add, delete, and modify Amazon MQ users and assign permissions to topics and queues. For more information, see Integrate LDAP with ActiveMQ (p. 47).</td>
</tr>
<tr>
<td>July 17, 2020</td>
<td>Amazon MQ now supports the <code>mq.t3.micro</code> instance type. For more information, see Broker instance types (p. 136).</td>
</tr>
<tr>
<td>Date</td>
<td>Documentation Update</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>June 30, 2020</td>
<td>Amazon MQ supports ActiveMQ 5.15.12. For more information, see the following:</td>
</tr>
<tr>
<td></td>
<td>• ActiveMQ 5.15.12 Release Notes</td>
</tr>
<tr>
<td></td>
<td>• Managing Amazon MQ for ActiveMQ engine versions (p. 115)</td>
</tr>
<tr>
<td></td>
<td>• Working with Spring XML configuration files (p. 97)</td>
</tr>
<tr>
<td>April 30, 2020</td>
<td>Amazon MQ supports a new child collection element, <code>systemUsage</code>, on the <code>broker</code> element. For more information, see <code>systemUsage</code> (p. 114).</td>
</tr>
<tr>
<td></td>
<td>Amazon MQ also supports three new attributes on the <code>kahaDB</code> child element.</td>
</tr>
<tr>
<td></td>
<td>• <code>journalDiskSyncInterval</code> - Interval (ms) for when to perform a disk sync if <code>journalDiskSyncStrategy=periodic</code>.</td>
</tr>
<tr>
<td></td>
<td>• <code>journalDiskSyncStrategy</code> - configures the disk sync policy.</td>
</tr>
<tr>
<td></td>
<td>• <code>preallocationStrategy</code> - configures how the broker will try to preallocate the journal files when a new journal file is needed.</td>
</tr>
<tr>
<td></td>
<td>For more information, see <code>Attributes</code> (p. 113).</td>
</tr>
<tr>
<td>March 3, 2020</td>
<td>Amazon MQ supports two new CloudWatch metrics</td>
</tr>
<tr>
<td></td>
<td>• <code>TempPercentUsage</code> - The percentage of available temporary storage used by non-persistent messages.</td>
</tr>
<tr>
<td></td>
<td>• <code>JobSchedulerStorePercentUsage</code> - The percentage of disk space used by the job scheduler store.</td>
</tr>
<tr>
<td></td>
<td>For more information, see <code>Monitoring Amazon MQ using CloudWatch</code> (p. 171).</td>
</tr>
<tr>
<td>February 4, 2020</td>
<td>Amazon MQ is available in the Asia Pacific (Hong Kong) and Middle East (Bahrain) regions. For information on available regions, see <code>AWS Regions and Endpoints</code>.</td>
</tr>
<tr>
<td>January 22, 2020</td>
<td>Amazon MQ supports ActiveMQ 5.15.10. For more information, see the following:</td>
</tr>
<tr>
<td></td>
<td>• ActiveMQ 5.15.10 Release Notes</td>
</tr>
<tr>
<td></td>
<td>• Managing Amazon MQ for ActiveMQ engine versions (p. 115)</td>
</tr>
<tr>
<td></td>
<td>• Working with Spring XML configuration files (p. 97)</td>
</tr>
<tr>
<td>December 19, 2019</td>
<td>Amazon MQ is available in the Europe (Stockholm) and South America (São Paulo) regions. For information on available regions, see <code>AWS Regions and Endpoints</code>.</td>
</tr>
</tbody>
</table>
Amazon MQ supports creating throughput-optimized brokers by using Amazon Elastic Block Store (EBS)—instead of the default Amazon Elastic File System (Amazon EFS)—for broker storage. To take advantage of high durability and replication across multiple Availability Zones, use Amazon EFS. To take advantage of low latency and high throughput, use Amazon EBS.

**Important**

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

For more information, see the following:

- Storage (p. 82)
- Choose the correct broker storage type for the best throughput (p. 73)
- The `storageType` property of the `broker-instance-options` resource in the Amazon MQ REST API Reference
- The BurstBalance, VolumeReadOps, and VolumeWriteOps metrics in the Amazon MQ for ActiveMQ metrics (p. 171) section.

October 18, 2019 Two Amazon CloudWatch metrics are available: TotalEnqueueCount and TotalDequeueCount. For more information, see ActiveMQ destination (queue and topic) metrics (p. 174).

October 11, 2019 Amazon MQ now supports Federal Information Processing Standard 140-2 (FIPS) compliant endpoints in U.S. commercial regions.

For more information see the following:

- Federal Information Processing Standard (FIPS) 140-2
- Amazon MQ Regions and Endpoints

September 30, 2019 Amazon MQ now includes the ability to scale your brokers by changing the host instance type. For more information, see the `hostInstanceType` property of `UpdateBrokerInput`, and the `pendingHostInstanceType` property of `DescribeBrokerOutput`.

August 30, 2019 You can now update the security groups associated with a broker, both in the console and with `UpdateBrokerInput`.
Amazon MQ integrates with AWS Key Management Service (KMS) to offer server-side encryption. You can now select your own customer managed CMK, or use an AWS managed KMS key in your AWS KMS account. For more information, see Encryption at rest (p. 144).

Amazon MQ supports using AWS KMS keys in the following ways.

- **AWS owned KMS key** — The key is owned Amazon MQ and is not in your account.
- **AWS managed KMS key** — The AWS managed KMS key (aws/mq) is a KMS key in your account that is created, managed, and used on your behalf by Amazon MQ.
- **Select existing customer managed CMK** — Customer managed CMKs are created and managed by you in AWS Key Management Service (KMS).

---

<table>
<thead>
<tr>
<th>Date</th>
<th>Documentation Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>July 22, 2019</td>
<td>Amazon MQ integrates with AWS Key Management Service (KMS) to offer server-side encryption. You can now select your own customer managed CMK, or use an AWS managed KMS key in your AWS KMS account. For more information, see Encryption at rest (p. 144).</td>
</tr>
<tr>
<td>June 3, 2019</td>
<td>Two new Amazon CloudWatch metrics are available: EstablishedConnectionsCount and InactiveDurableSubscribers. For more information, see the following:</td>
</tr>
<tr>
<td>June 12, 2019</td>
<td>Amazon MQ is available in the Canada (Central) region. For information on available regions, see AWS Regions and Endpoints.</td>
</tr>
<tr>
<td>June 19, 2019</td>
<td>Amazon MQ is available in the Europe (Paris) and Asia Pacific (Mumbai) regions. For information on available regions, see AWS Regions and Endpoints.</td>
</tr>
<tr>
<td>May 10, 2019</td>
<td>Data storage for new mq.t2.micro instance types is limited to 20 GB. For more information, see the following:</td>
</tr>
<tr>
<td>April 29, 2019</td>
<td>You can now use tag-based policies and resource-level permissions. For more information, see the following:</td>
</tr>
<tr>
<td>April 16, 2019</td>
<td>You can now retrieve information about broker engine and broker instance options using the REST API. For more information, see the following:</td>
</tr>
<tr>
<td>April 8, 2019</td>
<td>Amazon MQ supports ActiveMQ 5.15.9. For more information, see the following:</td>
</tr>
<tr>
<td>Date</td>
<td>Documentation Update</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| March 4, 2019            | Improved the documentation for configuring dynamic failover and the rebalancing of clients for a network of brokers. Enable dynamic failover by configuring `transportConnectors` along with `networkConnectors` configuration options. For more information, see the following:  
  - Dynamic Failover With Transport Connectors (p. 95)  
  - Amazon MQ Network of brokers (p. 86)  
  - Amazon MQ Broker Configuration Parameters (p. 97) |
| February 27, 2019        | Amazon MQ is available in the Europe (London) Region in addition to the following regions:  
  - Asia Pacific (Singapore)  
  - US East (Ohio)  
  - US East (N. Virginia)  
  - US West (N. California)  
  - US West (Oregon)  
  - Asia Pacific (Tokyo)  
  - Asia Pacific (Seoul)  
  - Asia Pacific (Sydney)  
  - Europe (Frankfurt)  
  - Europe (Ireland) |
| January 24, 2019         | The default configuration now includes a policy to purge inactive destinations.        |
| January 17, 2019         | Amazon MQ `mq.t2.micro` instance types now support only 100 connections per wire-level protocol. For more information, see, Quotas in Amazon MQ (p. 188). |
| December 19, 2018        | You can configure a series of Amazon MQ brokers in a network of brokers. For more information, see the following sections:  
  - Amazon MQ Network of brokers (p. 86)  
  - Creating and Configuring a Network of Brokers (p. 35)  
  - Configure Your Network of Brokers Correctly (p. 73)  
  - `networkConnector` (p. 112)  
  - `networkConnectionStartAsync` (p. 108) |
| December 11, 2018        | Amazon MQ supports ActiveMQ 5.15.8, 5.15.6, and 5.15.0.  
  - Resolved bugs and improvements in ActiveMQ:  
    - ActiveMQ 5.15.8 Release Notes  
    - ActiveMQ 5.15.7 Release Notes |
<p>| December 5, 2018         | AWS supports resource tagging to help track your cost allocation. You can tag resources when creating them, or by viewing the details of that resource. For more information, see Tagging resources. |
| November 19, 2018        | AWS has expanded its SOC compliance program to include Amazon MQ as an SOC compliant service. |</p>
<table>
<thead>
<tr>
<th>Date</th>
<th>Documentation Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>October 15, 2018</td>
<td>• The maximum number of groups per user is 20. For more information, see Users (p. 189).</td>
</tr>
<tr>
<td></td>
<td>• The maximum number of connections per broker, per wire-level protocol is 1,000. For more information, see Brokers (p. 188).</td>
</tr>
<tr>
<td>October 2, 2018</td>
<td>AWS has expanded its HIPAA compliance program to include Amazon MQ as a HIPAA Eligible Service.</td>
</tr>
<tr>
<td>September 27, 2018</td>
<td>Amazon MQ supports ActiveMQ 5.15.6, in addition to 5.15.0. For more information, see the following:</td>
</tr>
<tr>
<td></td>
<td>• Editing broker engine version, Amazon CloudWatch Logs, and maintenance preferences (p. 34)</td>
</tr>
<tr>
<td></td>
<td>• Resolved bugs and improvements in the ActiveMQ documentation:</td>
</tr>
<tr>
<td></td>
<td>• ActiveMQ 5.15.6 Release Notes</td>
</tr>
<tr>
<td></td>
<td>• ActiveMQ 5.15.5 Release Notes</td>
</tr>
<tr>
<td></td>
<td>• ActiveMQ 5.15.4 Release Notes</td>
</tr>
<tr>
<td></td>
<td>• ActiveMQ 5.15.3 Release Notes</td>
</tr>
<tr>
<td></td>
<td>• ActiveMQ 5.15.2 Release Notes</td>
</tr>
<tr>
<td></td>
<td>• ActiveMQ 5.15.1 Release Notes</td>
</tr>
<tr>
<td></td>
<td>• ActiveMQ Client 5.15.6</td>
</tr>
<tr>
<td>August 31, 2018</td>
<td>• The following metrics are available:</td>
</tr>
<tr>
<td></td>
<td>• CurrentConnectionsCount</td>
</tr>
<tr>
<td></td>
<td>• TotalConsumerCount</td>
</tr>
<tr>
<td></td>
<td>• TotalProducerCount</td>
</tr>
<tr>
<td></td>
<td>For more information, see the Amazon MQ for ActiveMQ metrics (p. 171) section.</td>
</tr>
<tr>
<td></td>
<td>• The IP address of the broker is displayed on the Details page.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong></td>
</tr>
<tr>
<td></td>
<td>For brokers with public accessibility disabled, the internal IP address is displayed.</td>
</tr>
<tr>
<td>August 30, 2018</td>
<td>Amazon MQ is available in the Asia Pacific (Singapore) Region in addition to the following regions:</td>
</tr>
<tr>
<td></td>
<td>• US East (Ohio)</td>
</tr>
<tr>
<td></td>
<td>• US East (N. Virginia)</td>
</tr>
<tr>
<td></td>
<td>• US West (N. California)</td>
</tr>
<tr>
<td></td>
<td>• US West (Oregon)</td>
</tr>
<tr>
<td></td>
<td>• Asia Pacific (Tokyo)</td>
</tr>
<tr>
<td></td>
<td>• Asia Pacific (Seoul)</td>
</tr>
<tr>
<td></td>
<td>• Asia Pacific (Sydney)</td>
</tr>
<tr>
<td></td>
<td>• Europe (Frankfurt)</td>
</tr>
<tr>
<td></td>
<td>• Europe (Ireland)</td>
</tr>
<tr>
<td>July 30, 2018</td>
<td>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. 183).</td>
</tr>
</tbody>
</table>

214
<table>
<thead>
<tr>
<th>Date</th>
<th>Documentation Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>July 25, 2018</td>
<td>Amazon MQ is available in the Asia Pacific (Tokyo) and Asia Pacific (Seoul) Regions in addition to the following regions:</td>
</tr>
<tr>
<td></td>
<td>• US East (Ohio)</td>
</tr>
<tr>
<td></td>
<td>• US East (N. Virginia)</td>
</tr>
<tr>
<td></td>
<td>• US West (N. California)</td>
</tr>
<tr>
<td></td>
<td>• US West (Oregon)</td>
</tr>
<tr>
<td></td>
<td>• Asia Pacific (Sydney)</td>
</tr>
<tr>
<td></td>
<td>• Europe (Frankfurt)</td>
</tr>
<tr>
<td></td>
<td>• Europe (Ireland)</td>
</tr>
<tr>
<td>July 19, 2018</td>
<td>You can use AWS CloudTrail to log Amazon MQ API calls. For more information, see Logging Amazon MQ API calls using CloudTrail (p. 180).</td>
</tr>
<tr>
<td>June 29, 2018</td>
<td>In addition to <code>mq.t2.micro</code> and <code>mq.m4.large</code>, the following broker instance types are available for regular development, testing, and production workloads that require high throughput:</td>
</tr>
<tr>
<td></td>
<td>• <code>mq.m5.large</code></td>
</tr>
<tr>
<td></td>
<td>• <code>mq.m5.xlarge</code></td>
</tr>
<tr>
<td></td>
<td>• <code>mq.m5.2xlarge</code></td>
</tr>
<tr>
<td></td>
<td>• <code>mq.m5.4xlarge</code></td>
</tr>
<tr>
<td></td>
<td>For more information, see Broker instance types (p. 136).</td>
</tr>
<tr>
<td>June 27, 2018</td>
<td>Amazon MQ is available in the US West (N. California) Region in addition to the following regions:</td>
</tr>
<tr>
<td></td>
<td>• US East (Ohio)</td>
</tr>
<tr>
<td></td>
<td>• US East (N. Virginia)</td>
</tr>
<tr>
<td></td>
<td>• US West (Oregon)</td>
</tr>
<tr>
<td></td>
<td>• Asia Pacific (Sydney)</td>
</tr>
<tr>
<td></td>
<td>• Europe (Frankfurt)</td>
</tr>
<tr>
<td></td>
<td>• Europe (Ireland)</td>
</tr>
<tr>
<td>Date</td>
<td>Documentation Update</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| 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 Users (p. 189).                                                                                                                                 |
| March 13, 2018       | Creating a broker takes about 15 minutes. For more information, see Finish creating the broker (p. 33).                                                                                                               |
| March 1, 2018        | - You can configure the concurrent store and dispatch (p. 72) for Apache KahaDB using the `concurrentStoreAndDispatchQueues` (p. 113) attribute.  
    - The `CpuCreditBalance` CloudWatch metric (p. 171) 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. 6) 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.                                                                                                     |
Amazon MQ Developer Guide

Document History

<table>
<thead>
<tr>
<th>Date</th>
<th>Documentation Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>November 28, 2017</td>
<td>This is the initial release of Amazon MQ and the <em>Amazon MQ Developer Guide</em>.</td>
</tr>
<tr>
<td></td>
<td>- Amazon MQ is available in the following regions:</td>
</tr>
<tr>
<td></td>
<td>- US East (Ohio)</td>
</tr>
<tr>
<td></td>
<td>- US East (N. Virginia)</td>
</tr>
<tr>
<td></td>
<td>- US West (Oregon)</td>
</tr>
<tr>
<td></td>
<td>- Asia Pacific (Sydney)</td>
</tr>
<tr>
<td></td>
<td>- Europe (Frankfurt)</td>
</tr>
<tr>
<td></td>
<td>- Europe (Ireland)</td>
</tr>
<tr>
<td></td>
<td>Using the <em>mq.t2.micro</em> instance type is subject to <em>CPU credits and baseline performance</em>—with the ability to <em>burst</em> above the baseline level (for more information, see the CpuCreditBalance (p. 171) metric). If your application requires <em>fixed performance</em>, consider using an <em>mq.m5.large</em> instance type.</td>
</tr>
<tr>
<td></td>
<td>- You can create <em>mq.m4.large</em> and <em>mq.t2.micro</em> brokers.</td>
</tr>
<tr>
<td></td>
<td>Using the <em>mq.t2.micro</em> instance type is subject to <em>CPU credits and baseline performance</em>—with the ability to <em>burst</em> above the baseline level (for more information, see the CpuCreditBalance (p. 171) metric). If your application requires <em>fixed performance</em>, consider using an <em>mq.m5.large</em> instance type.</td>
</tr>
<tr>
<td></td>
<td>- You can use the ActiveMQ 5.15.0 broker engine.</td>
</tr>
<tr>
<td></td>
<td>- You can also create and manage brokers programmatically using Amazon MQ REST API and AWS SDKs.</td>
</tr>
<tr>
<td></td>
<td>- You can access your brokers by using <em>any programming language that ActiveMQ supports</em> and by enabling TLS explicitly for the following protocols:</td>
</tr>
<tr>
<td></td>
<td>- AMQP</td>
</tr>
<tr>
<td></td>
<td>- MQTT</td>
</tr>
<tr>
<td></td>
<td>- MQTT over WebSocket</td>
</tr>
<tr>
<td></td>
<td>- OpenWire</td>
</tr>
<tr>
<td></td>
<td>- STOMP</td>
</tr>
<tr>
<td></td>
<td>- STOMP over WebSocket</td>
</tr>
<tr>
<td></td>
<td>- You can connect to ActiveMQ brokers using <em>various ActiveMQ clients</em>. We recommend using the <em>ActiveMQ Client</em>. For more information, see <em>Connecting a Java application to your broker</em> (p. 43).</td>
</tr>
<tr>
<td></td>
<td>- Your broker can send and receive messages of any size.</td>
</tr>
</tbody>
</table>

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. 206).

<table>
<thead>
<tr>
<th>Date</th>
<th>Documentation Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>January 13, 2022</td>
<td>Added a new troubleshooting section that lists the status codes that Amazon MQ returns when a broker is in an unhealthy state, along with detailed information about diagnosing, and recovering the broker.</td>
</tr>
<tr>
<td></td>
<td>- the section called “Troubleshooting: Amazon MQ status codes” (p. 199)</td>
</tr>
<tr>
<td>Date</td>
<td>Documentation Update</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>November 8, 2021</td>
<td>Added new tutorial that describes setting up a Python Pika client with Amazon MQ for RabbitMQ brokers.</td>
</tr>
<tr>
<td></td>
<td>• the section called “Using Python Pika with Amazon MQ for RabbitMQ” (p. 60)</td>
</tr>
<tr>
<td>October 8, 2021</td>
<td>Added the following troubleshooting topics for both Amazon MQ for ActiveMQ and Amazon MQ for RabbitMQ broker engines:</td>
</tr>
<tr>
<td></td>
<td>• Some clients unable to connect (p. 197)</td>
</tr>
<tr>
<td></td>
<td>• the section called “How do I enable plugins in Amazon MQ for RabbitMQ?” (p. 199)</td>
</tr>
<tr>
<td></td>
<td>• the section called “I’m unable to change Amazon VPC configuration for the broker.” (p. 199)</td>
</tr>
<tr>
<td>September 22, 2021</td>
<td>Added the following topics for troubleshooting common connection, and authorization issues with Amazon MQ for ActiveMQ brokers:</td>
</tr>
<tr>
<td></td>
<td>• Connecting to broker after a restart (p. 197)</td>
</tr>
<tr>
<td></td>
<td>• JSP exception on the web console (p. 198)</td>
</tr>
<tr>
<td>August 12, 2021</td>
<td>Added the following section to describe troubleshooting common issues when working with Amazon MQ brokers.</td>
</tr>
<tr>
<td></td>
<td>• Troubleshooting (p. 191)</td>
</tr>
<tr>
<td>July 29, 2021</td>
<td>Added the following sections to describe Amazon MQ for RabbitMQ version management and upgrading Amazon MQ brokers to new minor and major engine versions as they are supported.</td>
</tr>
<tr>
<td></td>
<td>• the section called “Version management” (p. 134)</td>
</tr>
<tr>
<td>July 21, 2021</td>
<td>Added the following sections to describe connecting an Amazon MQ broker to AWS Lambda as an event source.</td>
</tr>
<tr>
<td></td>
<td>• Connect your Amazon MQ for ActiveMQ broker to Lambda (p. 9)</td>
</tr>
<tr>
<td></td>
<td>• Connect your Amazon MQ for RabbitMQ broker to Lambda (p. 16)</td>
</tr>
<tr>
<td>July 16, 2021</td>
<td>Added the following sections to describe Amazon MQ broker maintenance windows, and how to adjust a maintenance window using the AWS Management Console, AWS CLI, or the Amazon MQ API.</td>
</tr>
<tr>
<td></td>
<td>• the section called “Maintaining a broker” (p. 21)</td>
</tr>
<tr>
<td>June 7, 2021</td>
<td>Added the following sections to describe Amazon MQ for ActiveMQ version management and upgrading Amazon MQ brokers to new minor and major engine versions as they are supported.</td>
</tr>
<tr>
<td></td>
<td>• the section called “Version management” (p. 115)</td>
</tr>
<tr>
<td></td>
<td>• the section called “Upgrading the engine version” (p. 24)</td>
</tr>
<tr>
<td>May 18, 2021</td>
<td>Added the following section to describe Amazon MQ for RabbitMQ broker defaults</td>
</tr>
<tr>
<td></td>
<td>• the section called “Broker defaults” (p. 125)</td>
</tr>
<tr>
<td>Date</td>
<td>Documentation Update</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| May 5, 2021       | Added the following section for describing AWS managed policies for Amazon MQ and updates to these policies:  
|                   | • the section called “AWS managed policies” (p. 158)                                                                                            |
| February 16, 2021 | Added the following tutorial section for Amazon MQ for RabbitMQ:  
|                   | • the section called “Resolving paused queue sync” (p. 65)                                                                                          |
| November 4, 2020  | • Added the following sections to document Amazon MQ for RabbitMQ support:  
|                   | • the section called “Creating and connecting to a RabbitMQ broker” (p. 11)                                                                    |
|                   | • the section called “RabbitMQ tutorials” (p. 60)                                                                                               |
|                   | • the section called “Amazon MQ for RabbitMQ best practices” (p. 74)                                                                       |
|                   | • the section called “RabbitMQ engine” (p. 124)                                                                                               |
|                   | • the section called “Configuring Amazon MQ for RabbitMQ logs” (p. 187)                                                                      |
|                   | • the section called “Using service-linked roles” (p. 159)                                                                                     |
|                   | • Additional revisions to existing chapters and sections of the guide were made to accurately document Amazon MQ for RabbitMQ support. |
| December 16, 2019 | • Added the following sections:  
|                   | • Storage (p. 82)                                                                                                                              |
|                   | • Choose the correct broker storage type for the best throughput (p. 73)                                                                        |
|                   | • Revised the information in the following sections:  
|                   | • Broker (p. 79)                                                                                                                               |
|                   | • Broker instance types (p. 136)                                                                                                               |
|                   | • Amazon MQ single-instance broker (p. 84)                                                                                                    |
|                   | • Amazon MQ active/standby broker for high availability (p. 85)                                                                                   |
|                   | • Create an ActiveMQ broker (p. 4)                                                                                                             |
|                   | • Creating and configuring a broker (p. 30)                                                                                                    |
| July 19, 2019     | Modified and added content on encryption management in the following sections:  
|                   | • Data protection in Amazon MQ (p. 143)                                                                                                         |
|                   | • Encryption at rest (p. 144)                                                                                                                  |
|                   | • Encryption in transit (p. 145)                                                                                                                |
|                   | • EncryptionOptions                                                                                                                           |
| April 22, 2019    | Added the following sections for tag-based policies and resource-level permissions:  
<p>|                   | • How Amazon MQ works with IAM (p. 150)                                                                                                        |
|                   | • Resource-level permissions for Amazon MQ API actions (p. 158)                                                                                 |</p>
<table>
<thead>
<tr>
<th>Date</th>
<th>Documentation Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>March 4, 2019</td>
<td>Improved the documentation for configuring dynamic failover and the rebalancing of clients for a network of brokers. Enable dynamic failover by configuring <code>transportConnectors</code> along with <code>networkConnectors</code> configuration options.</td>
</tr>
<tr>
<td></td>
<td>• Dynamic Failover With Transport Connectors (p. 95)</td>
</tr>
<tr>
<td></td>
<td>• Amazon MQ Network of brokers (p. 86)</td>
</tr>
<tr>
<td></td>
<td>• Amazon MQ Broker Configuration Parameters (p. 97)</td>
</tr>
<tr>
<td>January 5, 2019</td>
<td>Improved documentation on some per-minute metrics. For more information, see the following: <code>ActiveMQ destination (queue and topic) metrics (p. 174)</code></td>
</tr>
<tr>
<td>December 19, 2018</td>
<td>• Added the following sections:</td>
</tr>
<tr>
<td></td>
<td>• Amazon MQ Network of brokers (p. 86)</td>
</tr>
<tr>
<td></td>
<td>• Creating and Configuring a Network of Brokers (p. 35)</td>
</tr>
<tr>
<td></td>
<td>• Configure Your Network of Brokers Correctly (p. 73)</td>
</tr>
<tr>
<td></td>
<td>• <code>networkConnector</code> (p. 112)</td>
</tr>
<tr>
<td></td>
<td>• <code>networkConnectionStartAsync</code> (p. 108)</td>
</tr>
<tr>
<td></td>
<td>• Added the <code>networkConnectors</code> child collection element to the Elements, Child Collection Elements, and Their Child Elements Permitted in Amazon MQ Configurations (p. 108) section.</td>
</tr>
<tr>
<td>December 11, 2018</td>
<td>Updated documentation to reflect availability of ActiveMQ version 5.15.8.</td>
</tr>
<tr>
<td>December 5, 2018</td>
<td>Added the Tagging resources (p. 140) section.</td>
</tr>
<tr>
<td>October 26, 2018</td>
<td>Added the Avoid slow restarts by recovering prepared XA transactions (p. 73) section.</td>
</tr>
<tr>
<td>October 15, 2018</td>
<td>Updated the Quotas in Amazon MQ (p. 188) section.</td>
</tr>
<tr>
<td>October 3, 2018</td>
<td>Corrected outdated links in the Setting Up Amazon MQ (p. 2) and Amazon MQ Tutorials (p. 29) sections.</td>
</tr>
<tr>
<td>October 1, 2018</td>
<td>Corrected the information in the Next steps (p. 11) section.</td>
</tr>
<tr>
<td>September 27, 2018</td>
<td>• Added the Editing broker engine version, Amazon CloudWatch Logs, and maintenance preferences (p. 34) section.</td>
</tr>
<tr>
<td></td>
<td>• Updated the following sections:</td>
</tr>
<tr>
<td></td>
<td>• Create an ActiveMQ broker (p. 4)</td>
</tr>
<tr>
<td></td>
<td>• Configure Basic Broker Settings (p. 31)</td>
</tr>
<tr>
<td>September 18, 2018</td>
<td>Added the following note to the Creating and managing ActiveMQ broker users (p. 58) 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.</td>
</tr>
<tr>
<td>Date</td>
<td>Documentation Update</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| August 31, 2018 | • Clarified the terminology for active/standby brokers. For more information, see Amazon MQ active/standby broker for high availability (p. 85).  
• Simplified the terminology for the maintenance window. For more information, see Amazon MQ Broker Configuration Lifecycle (p. 96).  
• Rewrote the Configure Additional Broker Settings (p. 32) section.  
• Updated the Amazon MQ for ActiveMQ metrics (p. 171) and Listing brokers and viewing broker details (p. 19) sections. |
| August 15, 2018 | Corrected the information in the Create an ActiveMQ broker (p. 4) section.  
| August 13, 2018 | Added the Accessing the broker web console without public accessibility (p. 29) section.  
| August 2, 2018 | • Added the Troubleshooting CloudWatch Logs Configuration (p. 187) 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. 184)  
• Connect a Java application to your broker (p. 6)  
| July 31, 2018 | • Moved the 3-minute demo video to the Getting Started with Amazon MQ (p. 4) section.  
• Added the 3-minute getting started video to the What is Amazon MQ? (p. 1) section.  
| July 30, 2018 | • Added the Configuring Amazon MQ to publish logs to Amazon CloudWatch Logs (p. 183) section.  
• Updated the Configure Additional Broker Settings (p. 32) section.  
| July 19, 2018 | • Added the Logging Amazon MQ API calls using CloudTrail (p. 180) section.  
| July 5, 2018 | • Added an authorizationEntry child element cross-reference to the Always configure an authorization map (p. 166) section.  
• Clarified the information in the Integrating ActiveMQ brokers with LDAP (p. 47) section.  
• Clarified the information in the API Throttling (p. 190) section.  
| June 29, 2018 | • Updated the information in the Broker instance types (p. 136) section.  
• Added the Choose the Correct Broker Instance Type for the Best Throughput (p. 72) section.  

<table>
<thead>
<tr>
<th>Date</th>
<th>Documentation Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>June 4, 2018</td>
<td>In addition to GitHub, HTML, PDF, and Kindle, the <em>Amazon MQ Developer Guide</em> release notes are available as an RSS feed.</td>
</tr>
<tr>
<td>May 29, 2018</td>
<td>Made the following changes in the Working Java Example (p. 116) section:</td>
</tr>
<tr>
<td></td>
<td>• 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.</td>
</tr>
<tr>
<td></td>
<td>• Improved the MQTT Java example.</td>
</tr>
<tr>
<td></td>
<td>• Improved the OpenWire Java example.</td>
</tr>
<tr>
<td>May 24, 2018</td>
<td>Corrected the wire-level protocol endpoint port in the MQTT Java example in the Working Java Example (p. 116) section.</td>
</tr>
<tr>
<td>May 22, 2018</td>
<td>Corrected the information in all Java dependency sections.</td>
</tr>
<tr>
<td>May 17, 2018</td>
<td>Corrected the information in the Users (p. 189) section.</td>
</tr>
<tr>
<td>May 15, 2018</td>
<td>Corrected the information in the Ensuring effective Amazon MQ performance (p. 71) section.</td>
</tr>
<tr>
<td>May 8, 2018</td>
<td>• Placed the Amazon MQ REST API permissions reference (p. 157) in its own section.</td>
</tr>
<tr>
<td></td>
<td>• Created the IAM Permissions Required to Create an Amazon MQ Broker (p. 156) section with an example custom IAM policy.</td>
</tr>
<tr>
<td>May 7, 2018</td>
<td>• Clarified throughout this guide that the broker maintenance window is 2 hours long. For more information, see Amazon MQ Broker Configuration Lifecycle (p. 96).</td>
</tr>
<tr>
<td></td>
<td>• Added explanations for why the <code>ec2:CreateNetworkInterface</code> and <code>ec2:CreateNetworkInterfacePermission</code> permissions are necessary for creating a broker. For more information, see API authentication and authorization for Amazon MQ (p. 156).</td>
</tr>
<tr>
<td>May 1, 2018</td>
<td>Clarified the information about the maintenance window for active/standby brokers in the following sections:</td>
</tr>
<tr>
<td></td>
<td>• Amazon MQ active/standby broker for high availability (p. 85)</td>
</tr>
<tr>
<td></td>
<td>• Creating and configuring a broker (p. 30)</td>
</tr>
<tr>
<td></td>
<td>• Creating and applying broker configurations (p. 38)</td>
</tr>
<tr>
<td></td>
<td>• Editing and Managing Broker Configurations (p. 40)</td>
</tr>
<tr>
<td>April 27, 2018</td>
<td>Rewrote the following sections and optimized example Java code to match the recommendation to use connection pooling only for producers, not consumers:</td>
</tr>
<tr>
<td></td>
<td>• Always Use Connection Pooling (p. 70)</td>
</tr>
<tr>
<td></td>
<td>• Create a message producer and send a message (p. 7)</td>
</tr>
<tr>
<td></td>
<td>• Create a message consumer and receive the message (p. 8)</td>
</tr>
<tr>
<td></td>
<td>• AmazonMQExample.java (p. 119)</td>
</tr>
<tr>
<td>Date</td>
<td>Documentation Update</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>April 26, 2018</td>
<td>Added an MQTT Java example to the Working Java Example (p. 116) section. The MQTT example Java code connects to a broker, creates a topic, and publishes and receives a message.</td>
</tr>
<tr>
<td>April 4, 2018</td>
<td>Renamed the Communicating with Amazon MQ section to Connecting to Amazon MQ (p. 69).</td>
</tr>
<tr>
<td>April 3, 2018</td>
<td>Clarified and corrected the information in the Disable Concurrent Store and Dispatch for Queues with Slow Consumers (p. 72) section.</td>
</tr>
<tr>
<td>April 2, 2018</td>
<td>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. 72) section.</td>
</tr>
<tr>
<td>March 27, 2018</td>
<td>• Replaced the re:Invent launch video with a 3-minute demo video in the What is Amazon MQ? (p. 1) section. \n• Restructured the following sections: \n  • Broker Architecture (p. 83) \n  • How Amazon MQ Works (p. 79) \n  • Moved Amazon MQ Broker Configuration Lifecycle (p. 96) under the Broker Architecture (p. 83) section.</td>
</tr>
<tr>
<td>March 22, 2018</td>
<td>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 more information, see the AWS Encryption SDK Developer Guide.</td>
</tr>
<tr>
<td>March 19, 2018</td>
<td>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 Amazon EFS.</td>
</tr>
<tr>
<td>March 15, 2018</td>
<td>• Restructured the Amazon MQ Basic elements (p. 79) section.</td>
</tr>
<tr>
<td>March 12, 2018</td>
<td>• Clarified and corrected the information in the Security best practices for Amazon MQ (p. 166) and Connecting to Amazon MQ (p. 69) sections.</td>
</tr>
<tr>
<td></td>
<td>• Added the Disable Concurrent Store and Dispatch for Queues with Slow Consumers (p. 72) section.</td>
</tr>
<tr>
<td></td>
<td>• Grouped admonitions into a preface for the Configure advanced broker settings (p. 32) section.</td>
</tr>
<tr>
<td>March 9, 2018</td>
<td>• Clarified and corrected the information in the Always configure an authorization map (p. 166) section.</td>
</tr>
<tr>
<td></td>
<td>• Added the authorizationEntry (p. 111) section and updated the kahaDB (p. 113) section.</td>
</tr>
<tr>
<td>March 8, 2018</td>
<td>• Added the Always configure an authorization map (p. 166) section.</td>
</tr>
<tr>
<td></td>
<td>• Added notes about broker suffixes to the Monitoring Amazon MQ using CloudWatch (p. 171) section.</td>
</tr>
</tbody>
</table>
### Document History

<table>
<thead>
<tr>
<th>Date</th>
<th>Documentation Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>March 6, 2018</td>
<td>Added the following note throughout this guide: <strong>Note</strong> Using the <code>mq.t2.micro</code> instance type is subject to <em>CPU credits and baseline performance</em>—with the ability to <em>burst</em> above the baseline level (for more information, see the <code>CpuCreditBalance</code> metric). If your application requires <em>fixed performance</em>, consider using an <code>mq.m5.large</code> instance type.</td>
</tr>
</tbody>
</table>
| March 1, 2018           | • Added the `CpuCreditBalance` metric to the *Amazon MQ for ActiveMQ metrics* section.  
• Added the *Amazon MQ Child Element Attributes* section.  
• Added links from elements in the *Permitted Elements* 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.                                                                                                                                                                                                                                                                                                                   |
| 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. 166)  
• *Always Use the Failover Transport to Connect to Multiple Broker Endpoints* (p. 71)  
• API authentication and authorization for Amazon MQ (p. 156)  
• Integrating ActiveMQ brokers with LDAP (p. 47)                                                                                                                                                                                                                                    |
| February 21, 2018       | Corrected the Java code in the following sections:  
• *Working Java Example* (p. 116)  
• *Connect a Java application to your broker* (p. 6)  
• Always Use Connection Pooling (p. 70)                                                                                                                                                                                                                                               |
| February 20, 2018       | Clarified and corrected the information in the *Security in Amazon MQ* (p. 143) and *Best Practices for Amazon MQ* (p. 69) sections.                                                                                                                                                                                                                   |
| February 19, 2018       | • Corrected the Java code in the *Always Use Connection Pooling* section.  
• Restructured and expanded the *Best Practices for Amazon MQ* and *Security in Amazon MQ* sections.                                                                                                                                                                                                                           |
<table>
<thead>
<tr>
<th>Date</th>
<th>Documentation Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>February 16, 2018</td>
<td>• Added the Security best practices for Amazon MQ (p. 166) section.</td>
</tr>
<tr>
<td></td>
<td>• Updated the Connecting to Amazon MQ (p. 69) section.</td>
</tr>
<tr>
<td></td>
<td>• Corrected the Java code in the following sections:</td>
</tr>
<tr>
<td></td>
<td>• Getting Started with Amazon MQ (p. 4)</td>
</tr>
<tr>
<td></td>
<td>• AmazonMQExample.java (p. 119)</td>
</tr>
<tr>
<td>February 15, 2018</td>
<td>• Restructured and expanded the Best Practices for Amazon MQ (p. 69) section.</td>
</tr>
<tr>
<td></td>
<td>• Updated the following sections:</td>
</tr>
<tr>
<td></td>
<td>• How can I get started with Amazon MQ? (p. 1)</td>
</tr>
<tr>
<td></td>
<td>• Next steps (p. 11) (Getting Started)</td>
</tr>
<tr>
<td></td>
<td>• Related resources (p. 204)</td>
</tr>
<tr>
<td>February 14, 2018</td>
<td>Updated the following sections:</td>
</tr>
<tr>
<td></td>
<td>• Quotas in Amazon MQ (p. 188)</td>
</tr>
<tr>
<td></td>
<td>• API Throttling (p. 190)</td>
</tr>
<tr>
<td></td>
<td>• Best Practices for Amazon MQ (p. 69)</td>
</tr>
<tr>
<td></td>
<td>• Security in Amazon MQ (p. 143)</td>
</tr>
<tr>
<td>February 13, 2018</td>
<td>• Updated the Related resources (p. 204) section.</td>
</tr>
<tr>
<td></td>
<td>• Updated the Quotas in Amazon MQ (p. 188) section.</td>
</tr>
<tr>
<td></td>
<td>• Added the We want to hear from you (p. 1) section.</td>
</tr>
<tr>
<td>January 25, 2018</td>
<td>• Fixed an error in the Add Java dependencies (p. 117) subsection of the Working</td>
</tr>
<tr>
<td></td>
<td>Java Example (p. 116) section.</td>
</tr>
<tr>
<td></td>
<td>• The permission for REST operation ID <code>RebootBroker</code> is listed correctly as <code>mq:RebootBroker</code> on the IAM console.</td>
</tr>
<tr>
<td>January 24, 2018</td>
<td>• Added the Never Modify or Delete the Amazon MQ Elastic Network Interface (p. 69)</td>
</tr>
<tr>
<td></td>
<td>section.</td>
</tr>
<tr>
<td></td>
<td>• Updated all diagrams throughout this guide.</td>
</tr>
<tr>
<td></td>
<td>• Added links to the Amazon MQ REST API Reference throughout this guide and links to</td>
</tr>
<tr>
<td></td>
<td>specific REST APIs to the API authentication and authorization for Amazon MQ (p. 156)</td>
</tr>
<tr>
<td>January 19, 2018</td>
<td>Updated the information in the Amazon MQ for ActiveMQ resources (p. 204) section.</td>
</tr>
<tr>
<td>January 18, 2018</td>
<td>Clarified and corrected the information in the Quotas in Amazon MQ (p. 188) section.</td>
</tr>
<tr>
<td>January 17, 2018</td>
<td>Reinstated the recommendation to prefer virtual destinations over durable subscriptions (p. 71), with an improved explanation.</td>
</tr>
<tr>
<td>January 11, 2018</td>
<td>• The Amazon MQ Developer Guide is available in Kindle format, in addition to HTML and PDF.</td>
</tr>
<tr>
<td></td>
<td>• Clarified and corrected information in the API authentication and authorization for Amazon MQ (p. 156) and Step 2: create an IAM user and get your AWS credentials (p. 2) sections.</td>
</tr>
<tr>
<td>January 3, 2018</td>
<td>Added DescribeConfigurationRevision to the API authentication and authorization for Amazon MQ (p. 156) section.</td>
</tr>
<tr>
<td>Date</td>
<td>Documentation Update</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>December 15, 2017</td>
<td>Removed the recommendation against durable subscriptions from the Best Practices for Amazon MQ (p. 69) section.</td>
</tr>
</tbody>
</table>
| December 8, 2017  | • Added the Enable inbound connections (p. 6) prerequisite to the Connecting a Java application to your broker (p. 43) and Working Java Example (p. 116) sections.  
• Added the following note throughout this guide: Currently, you can't delete a configuration. |
| December 7, 2017  | • Improved the code in the AmazonMQExample.java (p. 119).  
• Added the API authentication and authorization for Amazon MQ (p. 156) section.                                                          |
| December 5, 2017  | • Clarified and corrected information in the Monitoring Amazon MQ using CloudWatch (p. 171) section:  
• Improved the metric descriptions.  
• Added the Amazon MQ for ActiveMQ metrics (p. 171) and Dimensions for ActiveMQ broker metrics (p. 174) 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 Data Storage (p. 189) section: Storage capacity per broker is 200 GB.  
• Added the Prerequisites (p. 117) to the Working Java Example (p. 116) 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. 43)). |
| 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. 42) or reboot the broker (p. 27). For more information, see Amazon MQ Broker Configuration Lifecycle (p. 96). |
AWS glossary

For the latest AWS terminology, see the AWS glossary in the AWS General Reference.