Distributed Message Service for RabbitMQ

User Guide

Issue 01

Date 2023-07-20

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Trademarks and Permissions


and other Huawei trademarks are the property of Huawei Technologies Co., Ltd.
All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between Huawei Cloud and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Contents

1 Permissions Management

1.1 Creating a User and Granting DMS for RabbitMQ Permissions

Use **Identity and Access Management (IAM)** to implement fine-grained permissions control over your Distributed Message Service (DMS) for RabbitMQ resources. With IAM, you can:

- Create IAM users for personnel based on your enterprise's organizational structure. Each IAM user has their own identity credentials for accessing DMS for RabbitMQ resources.
- Grant users only the permissions required to perform a given task based on their job responsibilities.
- Entrust a HUAWEI ID or a cloud service to perform efficient O&M on your DMS for RabbitMQ resources.

If your HUAWEI ID meets your permissions requirements, you can skip this section.

This section describes the procedure for granting user permissions. **Figure 1-1** shows the process flow.

NOTE

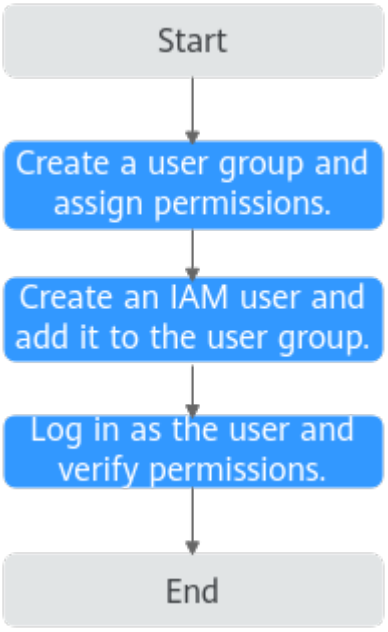
DMS for RabbitMQ permissions policies are based on DMS. Therefore, when assigning permissions, select DMS permissions policies.

Prerequisites

Learn about the permissions (see **System-defined roles and policies supported by DMS for RabbitMQ**) supported by DMS for RabbitMQ and choose policies according to your requirements. For the system policies of other services, see **System Permissions**.

Process Flow

Figure 1-1 Process of granting DMS for RabbitMQ permissions



1. On the IAM console, **create a user group and grant it permissions**.
   DMS ReadOnlyAccess is used as an example.
2. **Create an IAM user**.
3. **Log in as the IAM user** and verify permissions.
   In the authorized region, perform the following operations:
   - Choose **Service List** > **Distributed Message Service (for RabbitMQ)**. Then click **Buy Instance** on the console of DMS for RabbitMQ. If a message appears indicating that you have insufficient permissions to perform the operation, the **DMS ReadOnlyAccess** policy is in effect.
   - Choose **Service List** > **Elastic Volume Service**. If a message appears indicating that you have insufficient permissions to access the service, the **DMS ReadOnlyAccess** policy is in effect.

1.2 DMS for RabbitMQ Custom Policies

Custom policies can be created to supplement the system-defined policies of DMS for RabbitMQ. For the actions that can be added for custom policies, see **Permissions Policies and Supported Actions**.

You can create custom policies in either of the following ways:

- Visual editor: Select cloud services, actions, resources, and request conditions. This does not require knowledge of policy syntax.
- JSON: Edit JSON policies from scratch or based on an existing policy.

For details, see **Creating a Custom Policy**. The following section contains examples of common DMS for RabbitMQ custom policies.

NOTE

DMS for RabbitMQ permissions policies are based on DMS. Therefore, when assigning permissions, select DMS permissions policies.

Example Custom Policies

- Example 1: Allowing users to delete and restart instances

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dms:instance:modifyStatus",
                "dms:instance:delete"
            ]
        }
    ]
}
```

- Example 2: Denying instance deletion

  A policy with only "Deny" permissions must be used in conjunction with other policies to take effect. If the permissions assigned to a user contain both "Allow" and "Deny", the "Deny" permissions take precedence over the "Allow" permissions.

  For example, if you want to assign all of the permissions of the **DMS FullAccess** policy to a user, except for deleting instances, you can create a custom policy to deny only instance deletion. When you apply both the **DMS FullAccess** policy and the custom policy denying instance deletion, since "Deny" always takes precedence over "Allow", the "Deny" will be applied for that one conflicting permission. The user will then be able to perform all operations on instances except deleting instances. The following is an example of a deny policy:

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "dms:instance:delete"
            ]
        }
    ]
}
```

1.3 DMS for RabbitMQ Resources

A resource is an object that exists within a service. DMS for RabbitMQ resources are **rabbitmq**. You can select them by specifying their paths.

Table 1-1 DMS for RabbitMQ resources and their paths

Resource	Resource Name	Path
rabbitmq	Instance	Format: DMS:*:*: rabbitmq: *instance ID* Note: For instance resources, IAM automatically generates the prefix (**DMS:*:*:rabbitmq:**) of the resource path. For the path of a specific instance, add the *instance ID* to the end. You can also use an asterisk * to indicate any instance. For example: **DMS:*:*:rabbitmq:*** indicates any RabbitMQ instance.

1.4 DMS for RabbitMQ Request Conditions

Request conditions are useful for fine tuning when a custom policy takes effect. A request condition consists of a condition key and operator. Condition keys are either global or service-level and are used in the Condition element of a policy statement. **Global condition keys** (starting with **g:**) are available for operations of all services, while service-level condition keys (starting with a service name such as *dms:*) are available only for operations of a specific service. An operator is used together with a condition key to form a complete condition statement.

DMS for RabbitMQ has a group of predefined condition keys that can be used in IAM. For example, to define an "Allow" permission, you can use the condition key **dms:ssl** to check whether SSL is enabled for a RabbitMQ instance. The following table lists the predefined condition keys of DMS for RabbitMQ.

Table 1-2 Predefined condition keys of DMS for RabbitMQ

Condition Key	Operator	Description
dms:publicIP	Bool IsNullOrEmpty BoolIfExists	Whether public access is enabled
dms:ssl	Bool IsNullOrEmpty BoolIfExists	Whether SSL is enabled

2 Preparing the Environment

Before creating RabbitMQ instances, you must create a VPC and configure security groups and subnets for it. A VPC creates an isolated virtual network environment for you to configure and manage RabbitMQ instances, improving resource security and simplifying network deployment.

Once you have created a VPC, you can use it for all instances you subsequently create.

Creating a VPC

Step 1 Log in to the management console.

Step 2 In the upper left corner, click and select a region.

NOTE

Select the region where your RabbitMQ instance is.

Step 3 Click and choose **Networking** > **Virtual Private Cloud**.

Step 4 Click **Create VPC**.

Step 5 Create a VPC as prompted, For details on how to create a VPC, see the **Virtual Private Cloud User Guide**.

After a VPC is created, a subnet is also created. If the VPC needs more subnets, go to **Step 6**. Otherwise, go to **Step 7**.

Step 6 In the navigation pane, choose **Subnets**. Click **Create Subnet**. Create a subnet as prompted,

For details on how to create a subnet, see the **Virtual Private Cloud User Guide**.

Step 7 In the navigation pane, choose **Access Control** > **Security Groups**. Create a security group as prompted,

For details on how to create a security group, see the **Virtual Private Cloud User Guide**.

To use RabbitMQ instances, add the security group rules described in **Table 2-1**. Other rules can be added based on site requirements.

NOTE

After a security group is created, its default inbound rule allows communication among ECSs within the security group and its default outbound rule allows all outbound traffic. In this case, you can access a RabbitMQ instance within a VPC, and do not need to add rules according to **Table 2-1**.

Table 2-1 Security group rules

Direction	Protocol	Port	Source	Description
Inbound	TCP	5672	0.0.0.0/0	Access a RabbitMQ instance (without SSL encryption).
Inbound	TCP	5671	0.0.0.0/0	Access a RabbitMQ instance (with SSL encryption).
Inbound	TCP	15672	0.0.0.0/0	Access the management UI (without SSL encryption).
Inbound	TCP	15671	0.0.0.0/0	Access the management UI (with SSL encryption).

----End

3 Buying an Instance

Scenario

RabbitMQ instances are physically isolated and exclusively occupied by each tenant. You can customize the computing capabilities and storage space of a RabbitMQ instance based on service requirements.

RabbitMQ is an open-source service based on the Advanced Message Queuing Protocol (AMQP). It is used to store and forward messages in a distributed system. A RabbitMQ server is compiled in Erlang (supporting high concurrency, distributed deployment, and robust fault tolerance), and a RabbitMQ client can be compiled in various programming languages, including Python, Ruby, .NET, Java, JMS, C, PHP, ActionScript, XMPP, STOMP, and AJAX.

Advanced Message Queuing Protocol (AMQP) is an advanced message queue protocol that provides an open standard of application layer protocols.

Prerequisites

A VPC configured with security groups and subnets is available.

Procedure

Step 1 Log in to the management console.

Step 2 In the upper left corner, click  and select a region.

NOTE

Select the same region as your application service.

Step 3 Click  and choose **Application** > **Distributed Message Service for RabbitMQ** to open the console of DMS for RabbitMQ.

Step 4 Click **Buy Instance** in the upper right corner of the page.

Step 5 Specify **Billing Mode**, **Region**, **Project**, and **AZ**.

Step 6 Specify the instance name and the enterprise project.

Step 7 Configure the following instance parameters.

1. **Version**: RabbitMQ version. Currently, only 3.8.35 is supported.
2. **Instance Type**: Select **Single-node** or **Cluster**.
   - **Single-node**: There is only one RabbitMQ broker.
   - **Cluster**: There are multiple RabbitMQ brokers, achieving highly reliable message storage.
3. **CPU Architecture**: Currently, only x86 architecture is supported.
4. **Broker Flavor**: Select a flavor as required.

   NOTE

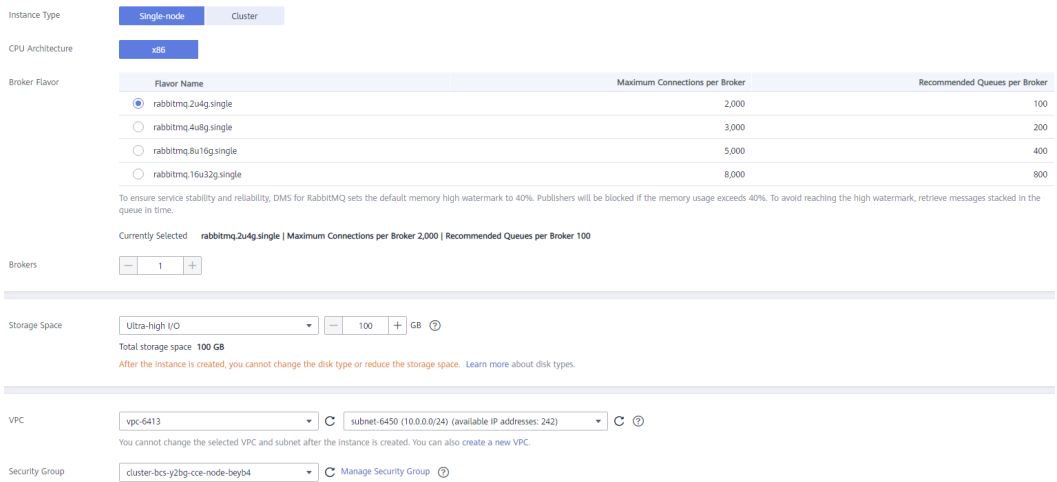
   To ensure service stability and reliability, DMS for RabbitMQ sets the default memory high watermark to 40%. Publishers will be blocked if the memory usage exceeds 40%. To avoid reaching the high watermark, retrieve messages stacked in the queue in time.
5. **Brokers**: Select the required number of brokers.
6. **Storage Space**: Indicates the disk type and total storage space of the RabbitMQ instance.

   For details on how to select a disk type, see **Disk Types and Performance**.
   - For a single-node instance, the value range is 100 GB to 30,000 GB.
   - For a cluster instance, the value range is Number of brokers x 100 GB to Number of brokers x 30,000 GB.
7. **VPC**: Select a VPC and a subnet.

   A VPC provides an isolated virtual network for your RabbitMQ instances. You can configure and manage the network as required.
8. **Security Group**: Select a security group.

   A security group is a set of rules for accessing a RabbitMQ instance. Click **Manage Security Group**. On the console that is displayed, view or create security groups.

Figure 3-1 Configuring the instance parameters



Step 8 Enter the username and password used for connecting to the RabbitMQ instance.

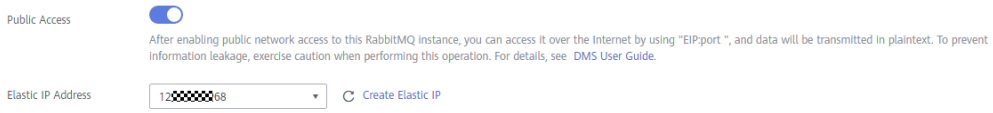
Step 9 Click **More Settings** to configure more parameters.

1. Configure **Public Access**.

   Public access can be enabled or disabled.

A RabbitMQ instance with public access enabled can be accessed by using an EIP. After you enable public access, **Elastic IP Address** is displayed. Select an EIP or click **Create Elastic IP** to view or buy EIPs.

Figure 3-2 Configuring public access for a RabbitMQ instance



NOTE

- In comparison with intra-VPC access, enabling public access might lead to packet loss and jitter. Therefore, you are advised to enable public access only during the service development and testing phases.
- If you manually unbind or delete an EIP on the VPC console, the public access function of the corresponding RabbitMQ instance is automatically disabled.

2. Configure **SSL**.

   This parameter indicates whether SSL authentication is enabled when a client is accessing an instance. If **SSL** is enabled, data will be encrypted before transmission for enhanced security.

   Once the instance is created, SSL cannot be enabled or disabled.

3. Specify tags.

   Tags are used to identify cloud resources. When you have many cloud resources of the same type, you can use tags to classify cloud resources by dimension (for example, usage, owner, or environment).

   - If you have created predefined tags, select a predefined pair of tag key and value. You can click **View predefined tags** to go to the Tag Management Service (TMS) console and view or create tags.
   - You can also create new tags by entering **Tag key** and **Tag value**.

   Up to 20 tags can be added to each RabbitMQ instance. For details about the requirements on tags, see **Managing Instance Tags**.

4. Enter a description of the instance.

Step 10 Click **Buy Now**.

Step 11 Confirm the instance information, read and agree to the *HUAWEI CLOUD Customer Agreement*, and click **Submit**.

Step 12 Return to the instance list and check whether the instance has been created.

It takes 3 to 15 minutes to create an instance. During this period, the instance status is **Creating**.

- If the instance is created successfully, its status changes to **Running**.
- If the instance fails to be created, view **Instance Creation Failures**. Delete the instance by referring to **Deleting an Instance** and create another instance. If the RabbitMQ instance creation fails a second time, contact customer service.

----End

4 Accessing a RabbitMQ Instance

4.1 Accessing a RabbitMQ Instance Without SSL Encryption

RabbitMQ instances are compatible with the open-source RabbitMQ protocol. To access and use a RabbitMQ instance in different languages, see the tutorials at **https://www.rabbitmq.com/getstarted.html**.

The following demo shows how to access and use a RabbitMQ instance in a VPC, assuming that the RabbitMQ client is deployed in an ECS.

If SSL is enabled for the instance, see **Accessing a RabbitMQ Instance with SSL Encryption** for how to access the instance.

Prerequisites

- A RabbitMQ instance has been created following the instructions in **Buying an Instance**, and the username and password used to create the instance have been obtained.
- The **Instance Address (Private Network)** or **Instance Address (Public Network)** of the instance has been recorded from the instance details.
- An ECS has been created, and its VPC, subnet, and security group configurations are the same as those of the RabbitMQ instance.

Accessing the Instance in CLI Mode

Step 1 Log in to the ECS.

Step 2 Install JDK or JRE, and add the following lines to **.bash_profile** in the home directory to configure the environment variables **JAVA_HOME** and **PATH**:

```
export JAVA_HOME=/opt/java/jdk1.8.0_151
export PATH=$JAVA_HOME/bin:$PATH
```

Run the **source .bash_profile** command for the modification to take effect.

📖 NOTE

Use Oracle JDK instead of ECS's default JDK (for example, OpenJDK), because ECS's default JDK may not be suitable for the sample project. Obtain Oracle JDK 1.8.111 or later from **Oracle's official website**.

Step 3 Run the following command to download **RabbitMQ-Tutorial.zip**:

```
$ wget https://dms-demo.obs.cn-north-1.myhuaweicloud.com/RabbitMQ-Tutorial.zip
```

Step 4 Run the following command to decompress **RabbitMQ-Tutorial.zip**:

```
$ unzip RabbitMQ-Tutorial.zip
```

Step 5 Run the following command to navigate to the **RabbitMQ-Tutorial** directory, which contains the precompiled JAR file:

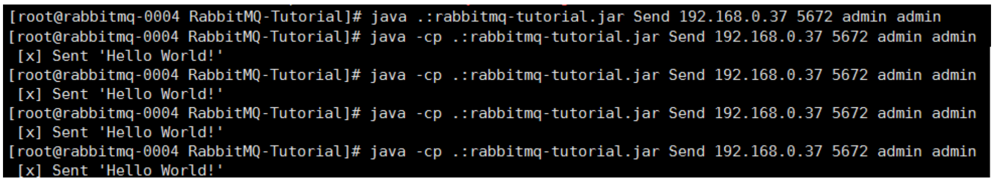
```
$ cd RabbitMQ-Tutorial
```

Step 6 Create messages using the sample project.

```
$ java -cp .:rabbitmq-tutorial.jar Send host port user password
```

host indicates the connection address for accessing the instance. *port* is the listening port of the instance, which is **5672** by default. *user* and *password* indicate the username and password used for accessing the instance.

Figure 4-1 Sample project for creating messages



```
[root@rabbitmq-0004 RabbitMQ-Tutorial]# java .:rabbitmq-tutorial.jar Send 192.168.0.37 5672 admin admin
[root@rabbitmq-0004 RabbitMQ-Tutorial]# java -cp .:rabbitmq-tutorial.jar Send 192.168.0.37 5672 admin admin
 [x] Sent 'Hello World!'
[root@rabbitmq-0004 RabbitMQ-Tutorial]# java -cp .:rabbitmq-tutorial.jar Send 192.168.0.37 5672 admin admin
 [x] Sent 'Hello World!'
[root@rabbitmq-0004 RabbitMQ-Tutorial]# java -cp .:rabbitmq-tutorial.jar Send 192.168.0.37 5672 admin admin
 [x] Sent 'Hello World!'
[root@rabbitmq-0004 RabbitMQ-Tutorial]# java -cp .:rabbitmq-tutorial.jar Send 192.168.0.37 5672 admin admin
 [x] Sent 'Hello World!'
```

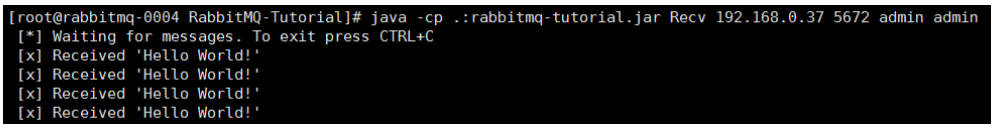
Press **Ctrl**+**C** to exit.

Step 7 Retrieve messages using the sample project.

```
$ java -cp .:rabbitmq-tutorial.jar Recv host port user password
```

host indicates the connection address for accessing the instance. *port* is the listening port of the instance, which is **5672** by default. *user* and *password* indicate the username and password used for accessing the instance.

Figure 4-2 Sample project for retrieving messages



```
[root@rabbitmq-0004 RabbitMQ-Tutorial]# java -cp .:rabbitmq-tutorial.jar Recv 192.168.0.37 5672 admin admin
 [*] Waiting for messages. To exit press CTRL+C
 [x] Received 'Hello World!'
 [x] Received 'Hello World!'
 [x] Received 'Hello World!'
 [x] Received 'Hello World!'
```

To stop retrieving messages, press **Ctrl**+**C** to exit.

----End

Java Sample Code

Accessing a RabbitMQ instance and creating messages

```
ConnectionFactory factory = new ConnectionFactory();
 factory.setHost(host);
```

```
factory.setPort(port);

factory.setUsername(user);
factory.setPassword(password);
Connection connection = factory.newConnection();
Channel channel = connection.createChannel();

channel.queueDeclare(QUEUE_NAME, false, false, false, null);

String message = "Hello World!";
channel.basicPublish("", QUEUE_NAME, null, message.getBytes("UTF-8"));
System.out.println(" [x] Sent '" + message + "'");

channel.close();
connection.close();
```

Accessing a RabbitMQ instance and retrieving messages

```
ConnectionFactory factory = new ConnectionFactory();
factory.setHost(host);
factory.setPort(port);
factory.setUsername(user);
factory.setPassword(password);
Connection connection = factory.newConnection();
Channel channel = connection.createChannel();

channel.queueDeclare(QUEUE_NAME, false, false, false, null);
System.out.println(" [*] Waiting for messages. To exit press CTRL+C");

Consumer consumer = new DefaultConsumer(channel)
{
    @Override
    public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties,
            byte[] body)
            throws IOException
    {
        String message = new String(body, "UTF-8");
        System.out.println(" [x] Received '" + message + "'");
    }
};
channel.basicConsume(QUEUE_NAME, true, consumer);
```

4.2 Accessing a RabbitMQ Instance with SSL Encryption

If SSL is enabled, data will be encrypted before transmission for enhanced security.

This section describes intra-VPC access to a RabbitMQ instance with SSL enabled.

Prerequisites

- A RabbitMQ instance has been created following the instructions in **Buying an Instance**, and the username and password used to create the instance have been obtained.
- The **Instance Address (Private Network)** or **Instance Address (Public Network)** of the instance has been recorded from the instance details.
- An ECS has been created, and its VPC, subnet, and security group configurations are the same as those of the RabbitMQ instance.

Accessing the Instance in CLI Mode

Step 1 Log in to the ECS. If public network access is enabled, log in to the server for running commands.

Step 2 Install JDK or JRE, and add the following lines to **.bash_profile** in the home directory to configure the environment variables **JAVA_HOME** and **PATH**:

```
export JAVA_HOME=/opt/java/jdk1.8.0_151
export PATH=$JAVA_HOME/bin:$PATH
```

Run the **source .bash_profile** command for the modification to take effect.

NOTE

Use Oracle JDK instead of ECS's default JDK (for example, OpenJDK), because ECS's default JDK may not be suitable for the sample project. Obtain Oracle JDK 1.8.111 or later from **Oracle's official website**.

Step 3 Run the following command to download **RabbitMQ-Tutorial-SSL.zip**:

```
$ wget https://dms-demo.obs.cn-north-1.myhuaweicloud.com/RabbitMQ-Tutorial.zip
```

Step 4 Run the following command to decompress **RabbitMQ-Tutorial-SSL.zip**:

```
$ unzip RabbitMQ-Tutorial-SSL.zip
```

Step 5 Run the following command to navigate to the **RabbitMQ-Tutorial-SSL** directory, which contains the precompiled JAR file:

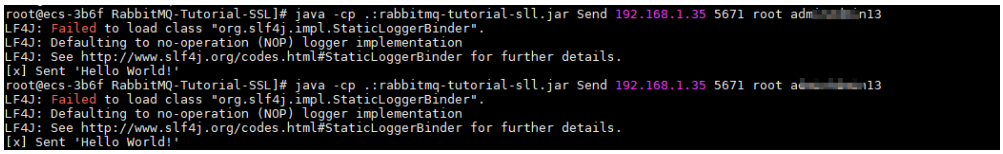
```
$ cd RabbitMQ-Tutorial-SSL
```

Step 6 Create messages using the sample project.

```
$ java -cp .:rabbitmq-tutorial-sll.jar Send host port user password
```

host indicates the connection address for accessing the instance. *port* is the listening port of the instance, which is **5671** by default. *user* and *password* indicate the username and password used for accessing the instance.

Figure 4-3 Sample project for message creation



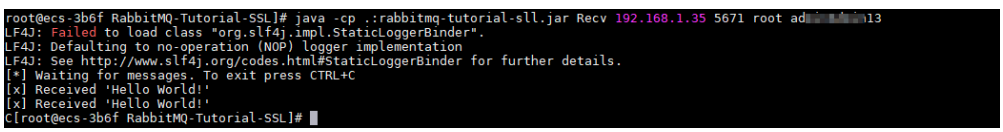
Press **Ctrl**+**C** to exit.

Step 7 Retrieve messages using the sample project.

```
$ java -cp .:rabbitmq-tutorial-sll.jar Recv host port user password
```

host indicates the connection address for accessing the instance. *port* is the listening port of the instance, which is **5671** by default. *user* and *password* indicate the username and password used for accessing the instance.

Figure 4-4 Sample project for message retrieval



To stop retrieving messages, press **Ctrl**+**C** to exit.

----End

Java Sample Code

Accessing a RabbitMQ instance and creating messages

```
ConnectionFactory factory = new ConnectionFactory();
 factory.setHost(host);
 factory.setPort(port);

 factory.setUsername(user);
 factory.setPassword(password);
 factory.useSslProtocol();
 Connection connection = factory.newConnection();
 Channel channel = connection.createChannel();

 channel.queueDeclare(QUEUE_NAME, false, false, false, null);

 String message = "Hello World!";
 channel.basicPublish("", QUEUE_NAME, null, message.getBytes("UTF-8"));
 System.out.println(" [x] Sent '" + message + "'");

 channel.close();
 connection.close();
```

Accessing a RabbitMQ instance and retrieving messages

```
ConnectionFactory factory = new ConnectionFactory();
 factory.setHost(host);
 factory.setPort(port);
 factory.setUsername(user);
 factory.setPassword(password);
 factory.useSslProtocol();
 Connection connection = factory.newConnection();
 Channel channel = connection.createChannel();

 channel.queueDeclare(QUEUE_NAME, false, false, false, null);
 System.out.println(" [*] Waiting for messages. To exit press CTRL+C");

 Consumer consumer = new DefaultConsumer(channel)
 {
    @Override
    public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties,
          byte[] body)
          throws IOException
    {
       String message = new String(body, "UTF-8");
       System.out.println(" [x] Received '" + message + "'");
    }
 };
 channel.basicConsume(QUEUE_NAME, true, consumer);
```

4.3 Connecting to the Management Address of a RabbitMQ Instance

Access the management address of a RabbitMQ instance by using the browser-based, open-source RabbitMQ cluster management tool.

Procedure

Step 1 Obtain the management address of an instance.

1. Log in to the management console.
2. In the upper left corner, click  and select a region.

NOTE

Select the same region as your application service.

3. Click ☰ and choose **Application** > **Distributed Message Service for RabbitMQ** to open the console of DMS for RabbitMQ.
4. Click the name of the instance whose management address you want to obtain. On the **Basic Information** tab page, view the **Mgmt. UI Address**, and **Username**.

Figure 4-5 Viewing the management UI address (public access disabled)

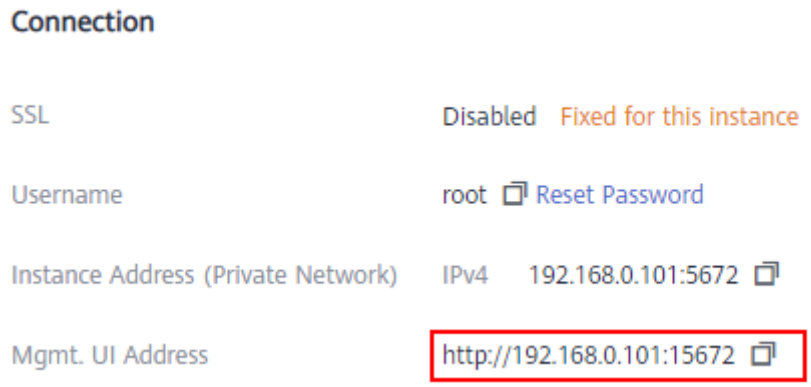


Figure 4-6 Viewing the management UI address (public access enabled)

NOTE

The username and password are customized when the RabbitMQ instance was created.

Step 2 Check whether the rules of the security group of the instance are correctly configured.

1. In the **Network** section on the **Basic Information** tab page, click the name of the security group.
2. Click the **Inbound Rules** tab to view the inbound rules of the security group.
   - SSL disabled
     - For intra-VPC access, inbound access through port 5672 must be allowed.
     - For public access, inbound access through port 15672 must be allowed.

- SSL enabled
  - For intra-VPC access, inbound access through port 5671 must be allowed.
  - For public access, inbound access through port 15671 must be allowed.

Step 3 In the address box of the browser, enter the URL of the management UI.

NOTE

- If public access is enabled for the RabbitMQ instance, you can use a browser to access the web page through the public network.
- If public access is not enabled for the RabbitMQ instance, you must purchase a Windows ECS that can connect to the RabbitMQ instance. Then, log in to the ECS and access the web page.

  For details on how to purchase an ECS, see **Purchasing an ECS**.

Figure 4-7 Logging in to the management UI

Step 4 Click **Login**.

----End

4.4 Enabling Heartbeats

If messages may be retrieved more than 90 seconds after they are created, enable heartbeats on the client and set the heartbeat timeout to shorter than 90 seconds, to prevent the client from being disconnected from a cluster RabbitMQ instance.

What Is a Heartbeat?

RabbitMQ heartbeats help the application layer detect interrupted connections and unresponsive peers in a timely manner. Heartbeats also prevent some network devices from disconnecting TCP connections where there is no activity for a certain period of time. **To enable heartbeats, specify the heartbeat timeout for connections.**

The heartbeat timeout defines after how long the peer TCP connection is considered closed by the server or client. The server and client negotiate the timeout value. The client must be configured with the value to request heartbeats.

The Java, .NET, and Erlang clients maintained by RabbitMQ use the following negotiation logic:

- If the heartbeat timeout set on neither the server nor the client is **0**, the smaller value is used.
- If the heartbeat timeout is set to **0** on either the server or the client, the non-zero value is used.
- If the heartbeat timeout set on both the server and the client is **0**, heartbeats are disabled.

After the heartbeat timeout is configured, the RabbitMQ server and client send AMQP heartbeat frames to each other at an interval of half the heartbeat timeout. After a client misses two heartbeats, it is considered unreachable and the TCP connection is closed. If the client detects that the server cannot be accessed due to heartbeats, the client needs to reconnect to the server.

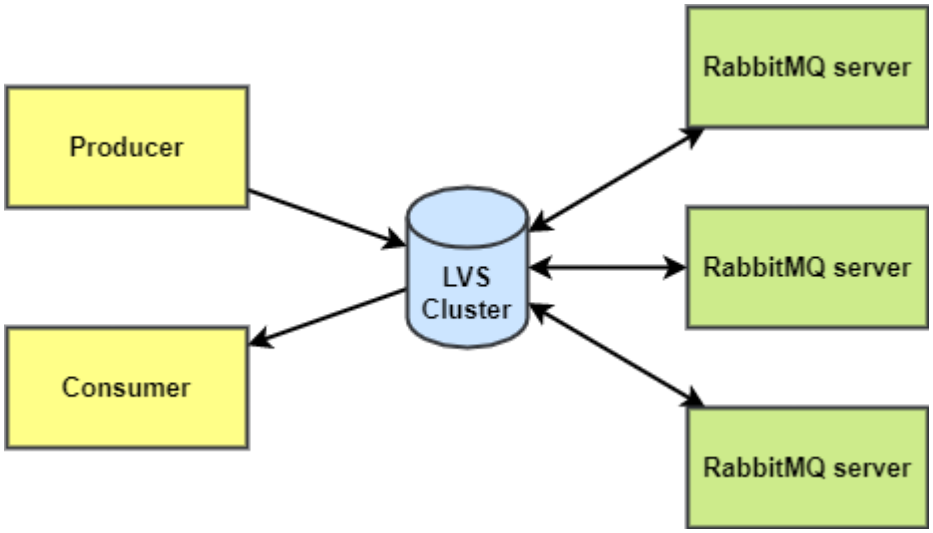
NOTICE

Some clients (such as C clients) do not have the logic for sending heartbeats. Even if the heartbeat timeout is configured and heartbeats are enabled, heartbeats still cannot be sent. In this case, an extra thread needs to be started to compile the logic for sending heartbeats.

LVS Heartbeat Timeout

Cluster RabbitMQ instances use Linux Virtual Servers (LVSs) for load balancing, as shown in **Figure 4-8**. Single-node instances do not have LVSs.

Figure 4-8 Load balancing of a cluster instance



LVS configures a heartbeat timeout of 90 seconds by default on client connections. If a client does not send a heartbeat (AMQP heartbeat frames or messages) to LVS for 90 seconds, LVS disconnects the client and the client will need to reconnect to LVS.

If messages are retrieved more than 90 seconds after they are created, enable heartbeats on the client and set the heartbeat timeout to shorter than 90 seconds.

Configuring Heartbeat Timeout on the Client

- Java client

  Before creating a connection, configure the **ConnectionFactory#setRequestedHeartbeat** parameter. Example:

  ```
  ConnectionFactory cf = new ConnectionFactory();
  // The heartbeat timeout is 60 seconds.
  cf.setRequestedHeartbeat(60);
  ```

- .NET client

  ```
  var cf = new ConnectionFactory();
  // The heartbeat timeout is 60 seconds.
  cf.RequestedHeartbeat = TimeSpan.FromSeconds(60);
  ```

- Python Pika

  ```
  // The heartbeat timeout is 60 seconds.
  params = pika.ConnectionParameters(host='host', heartbeat=60,
  credentials=pika.PlainCredentials('username', 'passwd'))
  connection = pika.BlockingConnection(params)

  while True:
      channel.basic_publish(exchange='', routing_key='hello', body='Hello World!')
      print(" [x] Sent 'Hello World!'")
      # The producer needs to use connection.sleep() to trigger a heartbeat. time.sleep() cannot trigger
  heartbeats.
      connection.sleep(200)
  ```

4.5 Viewing Client Connection Addresses

View client connection addresses on the RabbitMQ management UI.

NOTE

A client's connection addresses can be viewed only when the client is connected to a RabbitMQ instance.

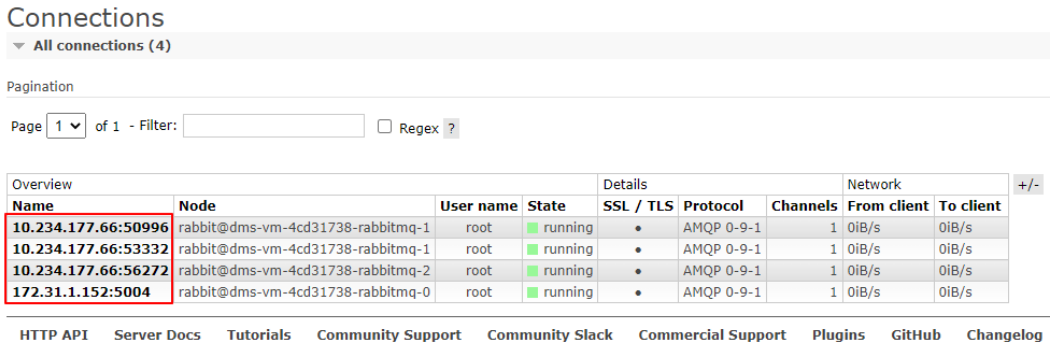
Procedure

Step 1 Log in to the RabbitMQ management UI.

Step 2 In the navigation pane, choose **Connections**.

Step 3 View client connection addresses, as shown in Figure 4-9.

Figure 4-9 Client connection addresses



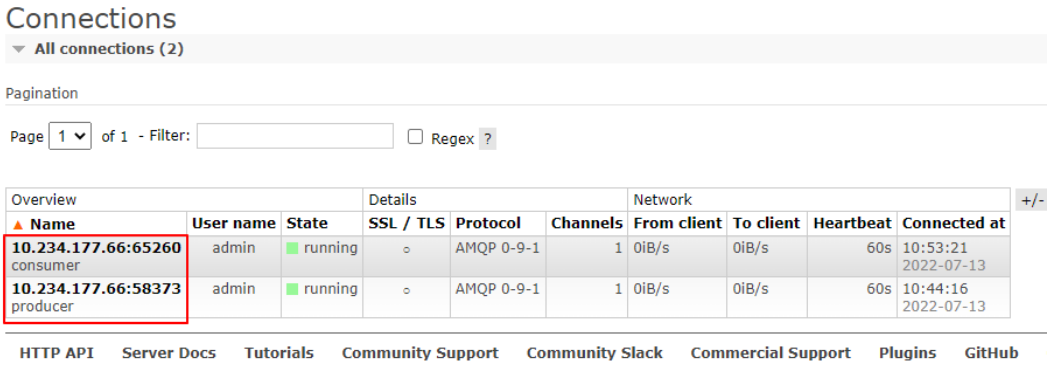
A client can function as a producer to create messages and as a consumer to retrieve messages. The producer and consumer IP addresses are the same, as shown in **Figure 4-9**, and are difficult to distinguish. To differentiate between producer and consumer IP addresses, you can set the **clientProperties** parameter on the client. The following is an example:

```
// Configure client connection parameters.
HashMap<String, Object> clientProperties = new HashMap<>();
clientProperties.put("connection_name", "producer");
connectionFactory.setClientProperties(clientProperties);

// Create a connection.
Connection connection = connectionFactory.newConnection();
```

After the **clientProperties** parameter is set, the connection addresses are displayed as shown in **Figure 4-10**.

Figure 4-10 Client connection addresses (with producer/consumer differentiated)



----End

5 Operating RabbitMQ Instances

5.1 Viewing an Instance

Scenario

View detailed information about a RabbitMQ instance on the console, for example, the connection address and port number for accessing the instance.

Prerequisites

A RabbitMQ instance has been created.

Procedure

Step 1 Log in to the management console.

Step 2 In the upper left corner, click and select a region.

NOTE

Select the region where your RabbitMQ instance is.

Step 3 Click and choose **Application** > **Distributed Message Service for RabbitMQ** to open the console of DMS for RabbitMQ.

Step 4 Search for a RabbitMQ instance by specifying the tag, status, name, connection address, or ID. **Table 5-1** describes the various possible statuses of a RabbitMQ instance.

Table 5-1 RabbitMQ instance status description

Status	Description
Creating	The instance is being created.

Status	Description
Running	The instance is running properly. Only instances in the **Running** state can provide services.
Faulty	The instance is not running properly.
Starting	The status between **Frozen** and **Running**.
Restarting	The instance is being restarted.
Changing	The instance specifications are being changed.
Change failed	The instance specifications failed to be changed.
Frozen	The instance is frozen.
Freezing	The status between **Running** and **Frozen**.
Upgrading	The instance is being upgraded.
Rolling back	The instance is being rolled back.

Step 5 Click the name of the chosen RabbitMQ instance and view the instance details on the page that is displayed.

Table 5-2 describes the parameters for connecting to an instance. For details about other parameters, see the **Basic Information** tab page of the instance on the console.

Table 5-2 Connection parameters

Parameter	Description
Instance Address (Private Network)	Address for connecting to the instance when public access is disabled.
Mgmt. UI Address	Address for connecting to the instance management UI when public access is disabled.
Public Access	Whether public access has been enabled.
Instance Address (Public Network)	Address for connecting to the instance when public access is enabled.
Mgmt. UI Address (Public Network)	Address for connecting to the instance management UI when public access is enabled.

----End

5.2 Restarting a RabbitMQ Instance

Scenario

Restart one or more RabbitMQ instances at a time on the RabbitMQ console.

NOTICE

When a RabbitMQ instance is being restarted, message retrieval and creation requests of the client will be rejected.

Prerequisites

The status of the RabbitMQ instance you want to restart is in the **Running** or **Faulty** state.

Procedure

Step 1 Log in to the management console.

Step 2 In the upper left corner, click and select a region.

NOTE

Select the region where your RabbitMQ instance is.

Step 3 Click and choose **Application** > **Distributed Message Service for RabbitMQ** to open the console of DMS for RabbitMQ.

Step 4 Restart RabbitMQ instances using one of the following methods:

- Select one or more RabbitMQ instances and click **Restart** in the upper left corner.
- In the row containing the desired RabbitMQ instance, click **Restart**.
- Click the desired RabbitMQ instance to view its details. In the upper right corner, click **Restart**.

Step 5 Click **Yes**.

It takes 3 to 15 minutes to restart a RabbitMQ instance. After it is successfully restarted, the instance should be in the **Running** state.

NOTE

Restarting a RabbitMQ instance only restarts the instance process and does not restart the VM where the instance is located.

To restart a single RabbitMQ instance, you can also click **Restart** in the row containing the chosen RabbitMQ instance on the **RabbitMQ Premium** page.

----End

5.3 Deleting an Instance

Scenario

With a few clicks on the DMS (for RabbitMQ) console, you can delete one or more RabbitMQ instances that have been created or multiple RabbitMQ instances that failed to be created.

NOTICE

Deleting a RabbitMQ instance will delete the data in the instance without any backup. Exercise caution when performing this operation.

Prerequisites

The status of the RabbitMQ instance you want to delete is **Running**, **Faulty**, or **Frozen**.

Deleting a RabbitMQ Instance

Step 1 Log in to the management console.

Step 2 In the upper left corner, click and select a region.

NOTE

Select the region where your RabbitMQ instance is.

Step 3 Click and choose **Application** > **Distributed Message Service for RabbitMQ** to open the console of DMS for RabbitMQ.

Step 4 Delete RabbitMQ instances using one of the following methods:

- Select one or more RabbitMQ instances and choose **More** > **Delete** in the upper left corner.
- In the row containing the RabbitMQ instance to be deleted, choose **More** > **Delete**.
- Click the desired RabbitMQ instance to view its details. In the upper right corner, choose **More** > **Delete**.

NOTE

RabbitMQ instances in the **Creating**, **Starting**, **Changing**, **Change failed**, or **Restarting** state cannot be deleted.

Step 5 Click **Yes**.

It takes 1 to 60 seconds to delete a RabbitMQ instance.

----End

Deleting a RabbitMQ Instance That Failed to Be Created

Step 1 Log in to the management console.

Step 2 In the upper left corner, click and select a region.

NOTE

Select the region where your RabbitMQ instance is.

Step 3 Click and choose **Application** > **Distributed Message Service for RabbitMQ** to open the console of DMS for RabbitMQ.

Step 4 If there are RabbitMQ instances that failed to be created, **Instance Creation Failures** and quantity information will be displayed. Click the icon or quantity next to **Instance Creation Failures**.

Step 5 Delete RabbitMQ instances that failed to be created in either of the following ways:

- To batch delete all RabbitMQ instances that failed to be created, click **Clear Failed Instance**.
- To delete a single RabbitMQ instance that failed to be created, click **Delete** in the row containing the chosen RabbitMQ instance.

----End

5.4 Modifying the Instance Information

After creating a RabbitMQ instance, you can adjust some parameters of the instance based on your service requirements.

Procedure

Step 1 Log in to the management console.

Step 2 In the upper left corner, click and select a region.

NOTE

Select the region where your RabbitMQ instance is.

Step 3 Click and choose **Application** > **Distributed Message Service for RabbitMQ** to open the console of DMS for RabbitMQ.

Step 4 Click the desired RabbitMQ instance to view its details.

Step 5 Modify the following parameters if needed:

- Instance Name
- Description
- Enterprise Project
- Security Group
- Public Access (For details about how to change the public access configuration, see **Configuring Public Access**.)

After the parameters are modified, view the modification result in the following ways:

- If **Public Access** has been modified, you will be redirected to the **Background Tasks** page, which displays the modification progress and result.
- If **Instance Name**, **Description**, **Enterprise Project**, or **Security Group** has been modified, the modification result will be displayed on the upper right corner of the page.

----End

5.5 Resetting the Instance Password

Scenario

If you forget the password of a RabbitMQ instance, reset the password so that you can normally access the instance.

NOTE

You can reset the password of a RabbitMQ instance only if the instance in the **Running** state.

Procedure

Step 1 Log in to the management console.

Step 2 In the upper left corner, click and select a region.

NOTE

Select the region where your RabbitMQ instance is.

Step 3 Click and choose **Application** > **Distributed Message Service for RabbitMQ** to open the console of DMS for RabbitMQ.

Step 4 Reset the instance password using either of the following methods:

- In the row containing the desired instance, choose **More** > **Reset Password**.
- Click the desired RabbitMQ instance to view its details. In the upper right corner, choose **More** > **Reset Password**.

Step 5 Enter and confirm a new password, and click **OK**.

- If the password is successfully reset, a success message will be displayed.
- If the password fails to be reset, a failure message will be displayed. If you still fail to reset the password after multiple attempts, contact customer service.

NOTE

A success message is displayed only after the password is successfully reset on all brokers.

----End

5.6 Modifying Instance Specifications

Scenario

After creating a RabbitMQ instance, you can increase or decrease its specifications. Table 5-3 lists available modification options.

Table 5-3 Specification modification options

Instance Type	Modified Object	Increase	Decrease
Cluster	Broker quantity	√	×
	Storage space	√	×
	Broker flavor	√	√
Single-node	Broker quantity	×	×
	Storage space	√	×
	Broker flavor	√	√

Notes and Constraints

- To ensure that the instance runs properly, do not perform other operations on the instance during the modification.
- The price may change after the modification.

Prerequisites

A RabbitMQ instance has been created and is in the **Running** state.

Procedure

Step 1 Log in to the management console.

Step 2 In the upper left corner, click and select a region.

NOTE

Select the region where your RabbitMQ instance is.

Step 3 Click and choose **Application** > **Distributed Message Service for RabbitMQ** to open the console of DMS for RabbitMQ.

Step 4 Modify the instance specifications using either of the following methods:

- In the row containing the desired instance, choose **More** > **Modify Specifications**.
- Click the desired RabbitMQ instance to view its details. In the upper right corner, choose **More** > **Modify Specifications**.

Step 5 Specify the required storage space, number of brokers, or bandwidth.

- Expand the storage space.

  For **Modify By**, select **Storage**. For **Storage Space per Broker**, specify a new storage space, and click **Next**. Confirm the configurations and click **Submit**.

  View the new storage space (Storage space per broker x Number of brokers) in the **Used/Available Storage Space (GB)** column in the instance list.

  NOTE

  Available storage space = Actual storage space – Storage space for storing logs – Disk formatting loss

  For example, if the storage space is expanded to 700 GB, the storage space for storing logs is 100 GB, and the disk formatting loss is 7 GB, then the available storage space after capacity expansion will be 593 GB.

- Add brokers.

  For **Modify By**, select **Brokers**. Then, enter the number of brokers and click **Next**. Confirm the configurations and click **Submit**.

  View the number of brokers in the **Specifications** column in the instance list.

  NOTE

  Services may temporarily stutter during the modification. Ensure that your client can auto-reconnect. Modify specifications during off-peak hours.

- Increase or decrease the broker flavor.

  For **Modify By**, select **Broker Flavor**. Then, select a new flavor and click **Next**. Confirm the configurations and click **Submit**.

  View the broker flavor in the **Flavor** column of the instance list.

  NOTE

  - For cluster instances without mirrored/quorum queues configured and single-node instances, services may stutter for several minutes during the modification. Ensure that your client can auto-reconnect. Modify specifications during off-peak hours.
  - For cluster instances configured with mirrored/quorum queues, services may stutter for several seconds during the modification. Ensure that your client can auto-reconnect. Modify specifications during off-peak hours.

----End

5.7 Configuring Public Access

Scenario

Enable public access to a RabbitMQ instance so that you can access the instance over a public network. If you no longer need public access to the instance, you can disable it as required.

NOTICE

In comparison with intra-VPC access, packet loss and jitter may occur and the access delay increases during public access. Therefore, you are advised to enable public access to RabbitMQ instances only during the service development and testing phase.

Prerequisites

Public access can be enabled only for RabbitMQ instances in the **Running** state.

Enabling Public Access

Step 1 Log in to the management console.

Step 2 In the upper left corner, click and select a region.

NOTE

Select the region where your RabbitMQ instance is.

Step 3 Click and choose **Application** > **Distributed Message Service for RabbitMQ** to open the console of DMS for RabbitMQ.

Step 4 Click the desired instance to view its details.

Step 5 Click next to **Public Access**.

Step 6 Select an EIP from the **Elastic IP Address** drop-down list and click .

If no EIP exists in the **Elastic IP Address** drop-down list box, click **Create Elastic IP** to create an EIP on the page that is displayed. After the EIP is created, return to the RabbitMQ console, click next to **Elastic IP Address**, and select the new EIP from the drop-down list.

It takes 10s to 30s to enable public access. After public access is enabled, the **Background Tasks** page is displayed. If the task status is **Successful**, public access is enabled successfully.

NOTE

After enabling public access, configure the following settings:

- If SSL is not enabled, add an inbound rule to the security group, allowing access to ports 5672 and 15672.

  To access the RabbitMQ management plane, visit http://*{public IP address of the RabbitMQ instance}*:15672, and enter your username and password.

  To access the instance through clients, use port 5672.
- If SSL is enabled, add an inbound rule to the security group, allowing access to ports 5671 and 15671.

  To access the RabbitMQ management plane, visit https://*{public IP address of the RabbitMQ instance}*:15671, and enter your username and password.

  To access the instance through clients, use port 5671.

----End

Disabling Public Access

Step 1 Log in to the management console.

Step 2 In the upper left corner, click and select a region.

NOTE

Select the region where your RabbitMQ instance is.

Step 3 Click and choose **Application** > **Distributed Message Service for RabbitMQ** to open the console of DMS for RabbitMQ.

Step 4 Click the desired instance to view its details.

Step 5 Click next to **Public Access**.

Step 6 Click .

It takes 10s to 30s to disable public access. After public access is disabled, the **Background Tasks** page is displayed. If the task status is **Successful**, public access is disabled successfully.

----End

5.8 Configuring Queue Mirroring

In a RabbitMQ cluster, queues can be mirrored across multiple nodes. In the event of a node failure, services are still available because the mirrors will take over services.

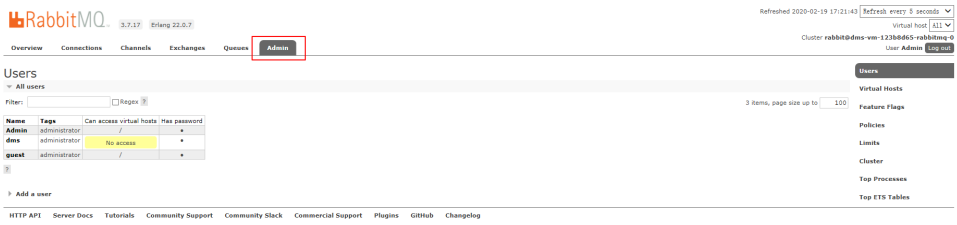
To learn more about the RabbitMQ management UI, visit the **RabbitMQ official website**. The following procedure describes how to configure queue mirroring on the RabbitMQ management UI.

Procedure

Step 1 Log in to the **management UI of a RabbitMQ instance**.

Step 2 Click the **Admin** tab.

Figure 5-1 Admin tab page



Step 3 (Optional) In the navigation tree on the right, choose **Virtual Hosts**, specify **Name**, and click **Add virtual host** to create a virtual host.

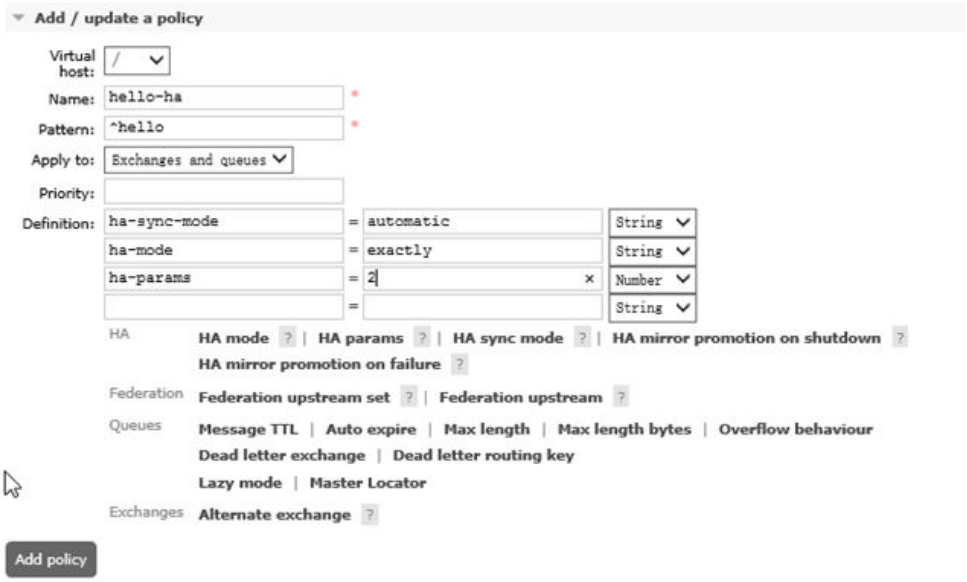
Perform this step only if you need to specify a virtual host. Otherwise, go to **Step 4**.

Figure 5-2 Creating a virtual host

Step 4 In the navigation tree on the right, choose **Policies** and set policies for the virtual host.

To set policies for a specific virtual host, select the virtual host created in **Step 3** from the **Virtual host** drop-down list box. If no virtual host has been created, the default value **/** is used.

Figure 5-3 Setting policies for a virtual host



Parameter description:

- **Name**: customized name of the policy.
- **Pattern**: regular expression that defines the pattern for matching queues.
- **Definition**: definition of the mirror, which consists of **ha-sync-mode**, **ha-mode**, and **ha-params**.
  - **ha-sync-mode**: queue synchronization mode. Options: **automatic** and **manually**.
    - **automatic**: Data is automatically synchronized from the master.
    - **manually**: Data is manually synchronized from the master.
  - **ha-mode**: queue mirroring mode. Options: **all**, **exactly**, and **nodes**.
    - **all**: Mirror the queue across all nodes in the cluster.

    - **exactly**: Mirror the queue to a specific number (determined through **ha-params**) of nodes in the cluster.
    - **nodes**: Mirror the queue to specific nodes (determined through **ha-params**).
  - **ha-params**: This parameter is used for specifying the nodes in the **ha-mode** parameter.
- (Optional) **Priority**: priority of the policy.

Step 5 Click **Add policy**.

The following figure shows a successfully added policy.

Figure 5-4 Virtual host policy

----End

5.9 Managing Instance Tags

Tags facilitate RabbitMQ instance identification and management.

You can add tags to a RabbitMQ instance when creating the instance or add tags on the details page of the created instance. Up to 20 tags can be added to an instance. Tags can be modified and deleted.

A tag consists of a tag key and a tag value. **Table 5-4** lists the tag key and value requirements.

Table 5-4 Tag key and value requirements

Name	Rules
Tag key	• Cannot be left blank. • Must be unique for the same instance. • Can contain a maximum of 36 characters. • Cannot contain the following characters: =*<>\,|/ • Cannot start or end with a space.
Tag value	• Cannot be left blank. • Can contain a maximum of 43 characters. • Cannot contain the following characters: =*<>\,|/ • Cannot start or end with a space.

Procedure

Step 1 Log in to the management console.

Step 2 In the upper left corner, click and select a region.

NOTE

Select the region where your RabbitMQ instance is.

Step 3 Click and choose **Application** > **Distributed Message Service for RabbitMQ** to open the console of DMS for RabbitMQ.

Step 4 Click the desired instance to view its details.

Step 5 Click the **Tags** tab. Tags of the instance are displayed.

Step 6 Perform the following operations as required:

- Adding a tag
  a. Click **Create/Delete Tag**.
  b. Enter a tag key and a tag value, and click **Add**.
     If you have created predefined tags, select a pair of tag key and value, and click **Add**.
  c. Click **OK**.
- Deleting a tag
  Delete a tag using either of the following methods:
  – In the row containing the tag to be deleted, click **Delete**. Click **Yes**.
  – Click **Create/Delete Tag**. In the dialog box that is displayed, click next to the tag to be deleted and click **OK**.

----End

5.10 Deleting Queues

Delete queues on the RabbitMQ management UI or by calling APIs.

- **Method 1: Deleting a Single Queue on the Management UI**: Delete a single queue on the **Queues** tab page of the management UI.
- **Method 2: Deleting Queues in Batches Using a Policy**: Add a policy to delete multiple queues at a time. The policy has the same prefix as the queues to be deleted, and the queue time-to-live (TTL) is 1 ms.
- **Method 3: Deleting a Single Queue Using an API**: Call an API to delete a queue from a RabbitMQ instance with SSL disabled.
- **Method 4: Deleting Queues in Batches Using an API**: Compile a shell script to repeatedly call an API to delete queues in batches from a RabbitMQ instance with SSL disabled.

NOTICE

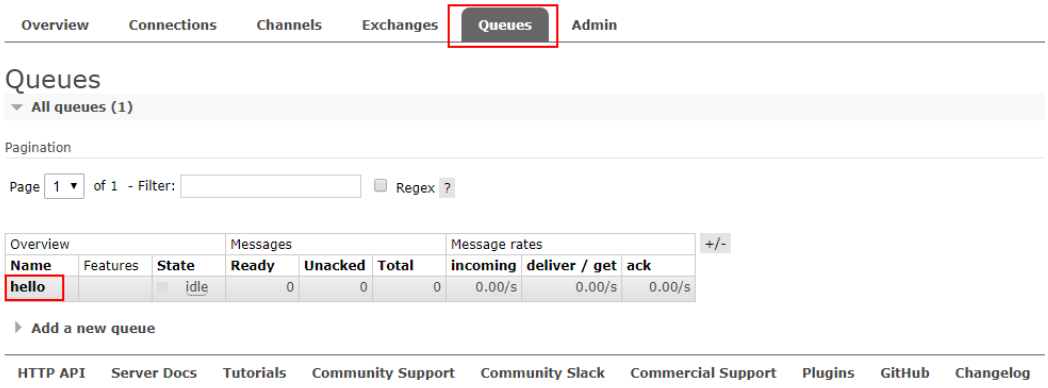
Before deleting a queue, ensure that all messages in the queue have been retrieved. Otherwise, unretrieved messages will be deleted together with the queue.

Method 1: Deleting a Single Queue on the Management UI

Step 1 Log in to the RabbitMQ management UI.

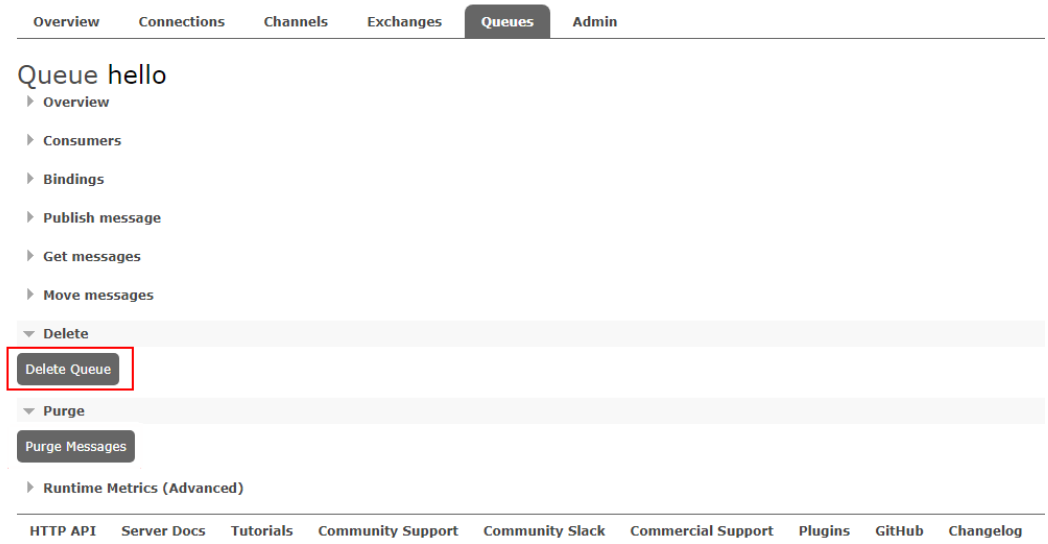
Step 2 On the **Queues** tab page, click the name of the desired queue.

Figure 5-5 Queue list



Step 3 Click **Delete Queue** to delete the queue.

Figure 5-6 Deleting a single queue



----End

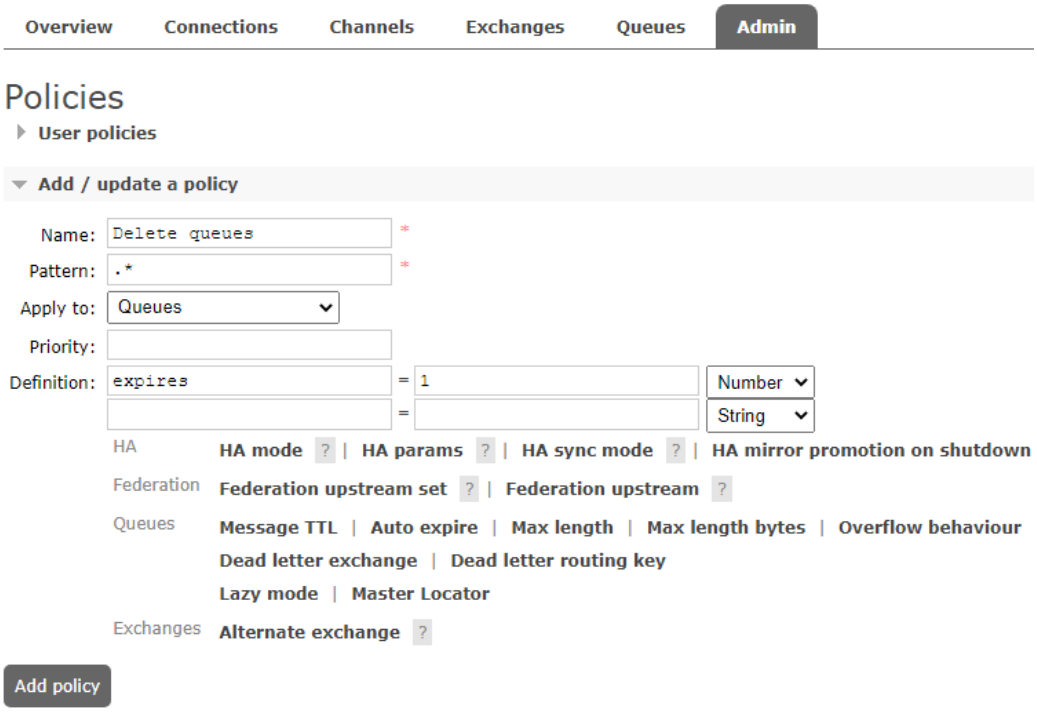
Method 2: Deleting Queues in Batches Using a Policy

Add a policy to delete multiple queues at a time. The policy has the same prefix as the queues to be deleted, and the queue TTL is 1 ms.

Step 1 **Log in to the RabbitMQ management UI**.

Step 2 On the **Admin** > **Policies** page, add a policy.

Figure 5-7 Adding a policy to delete queues in batches



- **Name**: Enter a policy name.
- **Pattern**: queue matching mode. Enter a queue name. Queues containing this queue name will be matched. If this parameter is set to **.***, all queues are matched. If this parameter is set to **.*queue-name**, all queues whose name contains **queue-name** are matched.
- **Apply to**: Select **Queues**.
- **Priority**: policy priority. A larger value indicates a higher priority. This parameter is optional.
- **Definition**: TTL, in milliseconds. Set **expires** to **1**, indicating that the queue expiration time is 1 ms.

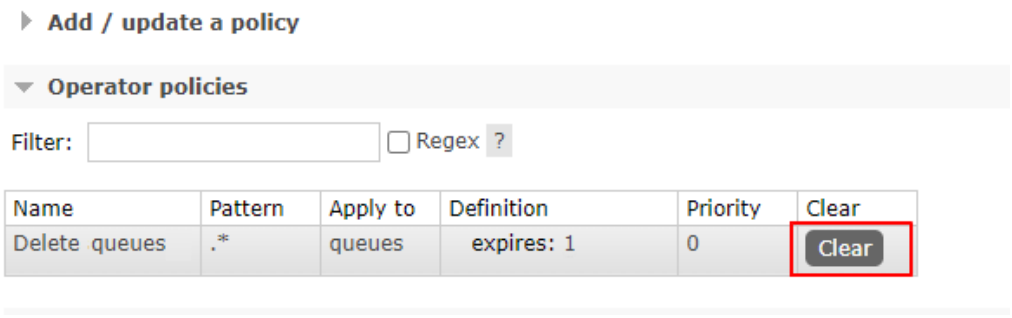
Step 3 Click **Add policy**.

On the **Queues** tab page, check whether the queues are successfully deleted.

Step 4 After the queues are deleted, choose **Admin** > **Policies**, locate the row that contains the policy added in **Step 2**, and click **Clear** to delete the policy.

If this policy is retained, it will also apply to queues created later, and queues may be deleted by mistake.

Figure 5-8 Deleting the policy



----End

Method 3: Deleting a Single Queue Using an API

If SSL is not enabled for a RabbitMQ instance, you can call an API to delete a queue.

Step 1 Connect to the instance in Linux. For details, see **Accessing a RabbitMQ Instance Without SSL Encryption**.

Step 2 Run the following command to delete a queue:

```
curl -i -XDELETE http://${USERNAME}:${PASSWORD}@${HOST}:${PORT}/api/queues/${VHOST_NAME}/${QUEUE_NAME}
```

Parameter description:

- **USERNAME**: username set when the instance was created.
- **PASSWORD**: password set when the instance was created. If you forget the password, reset it by referring to **Resetting the Instance Password**.
- **HOST**: management UI address queried on the instance details page.
- **PORT**: management UI port number queried on the instance details page.
- **VHOST_NAME**: virtual host name. The default value is **/**. Change it to **%2F** in the command.
- **QUEUE_NAME**: name of the queue to be deleted.

Example:

```
curl-i -XDELETE http://test:Zsxxxdx@192.168.0.241:15672/api/queues/%2F/hello
```

If the deletion is successful, the following information is displayed.

Figure 5-9 Queue deleted

You can also check whether the queue is successfully deleted on the **Queues** tab page of the management UI.

----End

Method 4: Deleting Queues in Batches Using an API

If SSL is not enabled for a RabbitMQ instance, you can compile a shell script to repeatedly call an API to delete queues in batches.

Step 1 Connect to the instance in Linux. For details, see **Accessing a RabbitMQ Instance Without SSL Encryption**.

Step 2 Create the **delete_queues.sh** script.

```
touch delete_queues.sh
```

Step 3 Edit the script.

```
vim delete_queues.sh
```

Copy the following content to the script. Change the values of **USERNAME**, **PASSWORD**, **HOST**, and **QUEUES_LIST** as required.

```
#!/usr/bin/env bash

USERNAME=root
PASSWORD=Zsxxxdx
HOST=192.168.0.241
PORT=15672
VHOST='%2F'


QUEUES_LIST="test1 test2 test3";
for QUEUE_NAME in $QUEUES_LIST :
do
   curl -i -XDELETE http://$USERNAME:$PASSWORD@$HOST:$PORT/api/queues/$VHOST/$QUEUE_NAME
done
```

Parameter description:

- **USERNAME**: username set when the instance was created.
- **PASSWORD**: password set when the instance was created. If you forget the password, reset it by referring to **Resetting the Instance Password**.
- **HOST**: management UI address queried on the instance details page.
- **PORT**: management UI port number queried on the instance details page.
- **VHOST**: virtual host name. The default value is **/**. Change it to **%2F** in the command.
- **QUEUES_LIST**: names of the queues to be deleted. Use spaces to separate queue names.

Step 4 Save the script.

Step 5 Configure the script permissions.

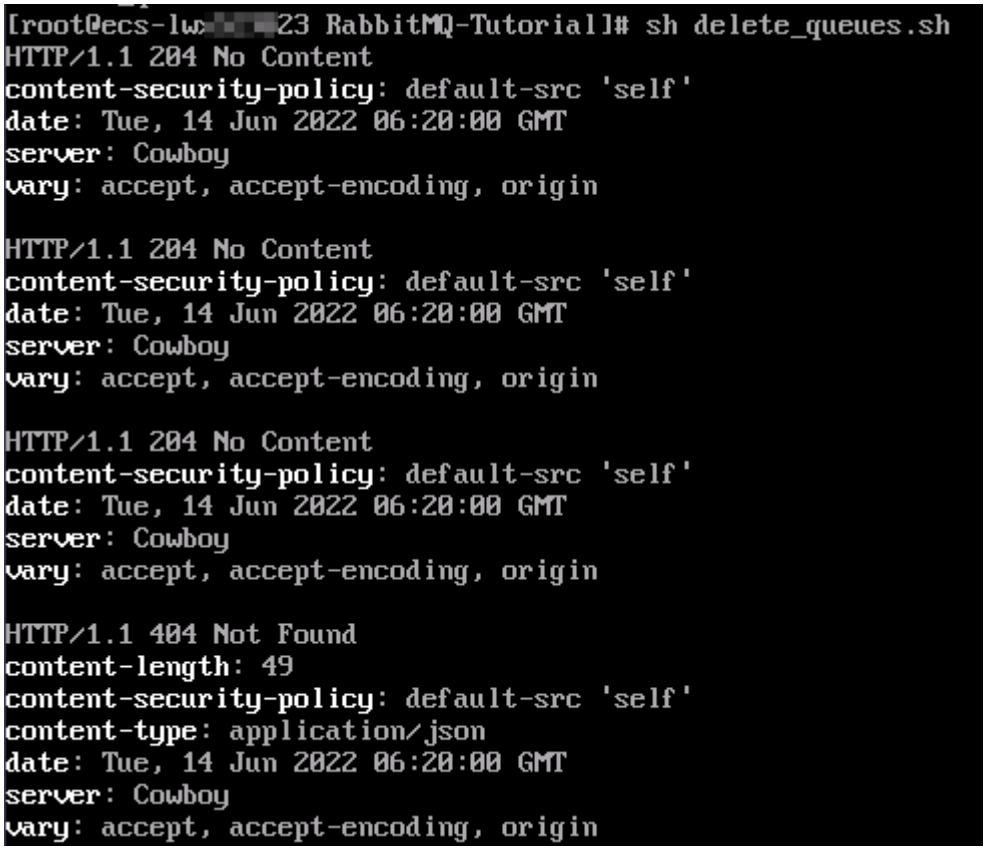
```
chmod 777 delete_queues.sh
```

Step 6 Run the script.

```
sh delete_queues.sh
```

If the deletion is successful, the following information is displayed.

Figure 5-10 Queues deleted



You can also check whether the queues are successfully deleted on the **Queues** tab page of the management UI.

----End

6 Plug-in Management

6.1 Enabling Plug-ins

After creating a RabbitMQ instance, you can enable the plug-ins listed in the following table. The plug-ins are disabled by default when the instance is created.

RabbitMQ plug-ins can be used for testing and service migration. Do not use them for production. For details, see **Notes and Constraints**.

NOTE

When plug-ins are enabled, the instance will not be restarted. However, enabling plug-ins rabbitmq_mqtt, rabbitmq_web_mqtt, rabbitmq_stomp, and rabbitmq_web_stomp will restart Keepalived and disconnect the instance. After the instance is disconnected, it may be automatically reconnected depending on the service logic.

Table 6-1 Plug-ins that can be enabled or disabled

Name	Function	Port
rabbitmq_amqp1_0	Support for AMQP 1.0	-
rabbitmq_delayed_message_exchange	Delayed messages There may be an error of about 1%. The actual delivery time may be earlier or later than the scheduled delivery time.	-
rabbitmq_federation	Federation	-
rabbitmq_sharding	Sharding	-
rabbitmq_shovel	Message moving	-
rabbitmq_tracing	Message tracing	-
rabbitmq_mqtt	Support for MQTT over TCP	1883
rabbitmq_web_mqtt	Support for MQTT over WebSocket	15675

Name	Function	Port
rabbitmq_stomp	Support for STOMP over TCP	61613
rabbitmq_web_stomp	Support for STOMP over WebSocket	15674
rabbitmq_consistent_hash_exchange	Consistent hash exchange	-

NOTE

The ports of the plug-ins cannot be changed.

Procedure

Step 1 Log in to the management console.

Step 2 In the upper left corner, click and select a region.

NOTE

Select the region where your RabbitMQ instance is.

Step 3 Click and choose **Application** > **Distributed Message Service for RabbitMQ** to open the console of DMS for RabbitMQ.

Step 4 Click the desired instance to view its details.

Step 5 On the **Plug-ins** tab page, click **Enable** next to the desired plug-in.

Confirm that you want to enable the plug-in and wait for it to be enabled successfully.

----End

6.2 Using the rabbitmq_tracing Plug-in

Scenario

The rabbitmq_tracing plug-in provides the message tracing function. It traces incoming and outgoing messages of RabbitMQ, captures the messages, and saves message logs to the corresponding trace file.

Prerequisites

You have purchased an instance.

Procedure

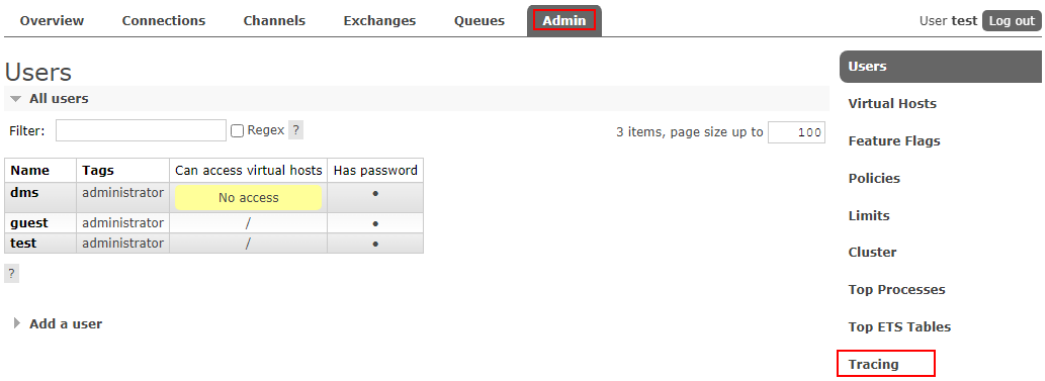
Step 1 Enable the rabbitmq_tracing plug-in by referring to **Enabling Plug-ins**.

Step 2 **Log in to the RabbitMQ management UI**.

Step 3 On the top navigation bar, choose **Admin**.

Step 4 In the navigation tree on the right, choose **Tracing**.

Figure 6-1 Admin page



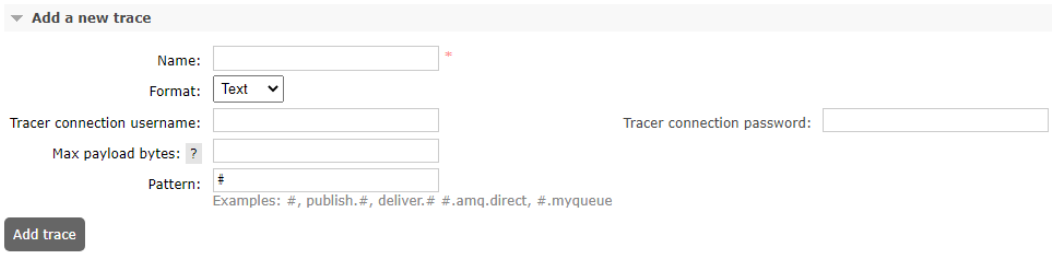
Step 5 In the **Add a new trace** area, set the following parameters and click **Add trace** to add a trace.

Table 6-2 Trace parameters

Parameter	Description
Name	Name of the trace.
Format	Format of the output message log. **Text** (easy to read) and **JSON** (easy to parse) are supported.
Tracer connection username	Name of the user that creates tracing.
Tracer connection password	Password of the user that creates a trace.
Max payload bytes	Maximum size of a message, in bytes. Assume that **Max payload bytes** is set to **10**. A message larger than 10 bytes will be truncated when it is transferred through RabbitMQ. For example, **trace test payload** will become **trace test** after truncation.

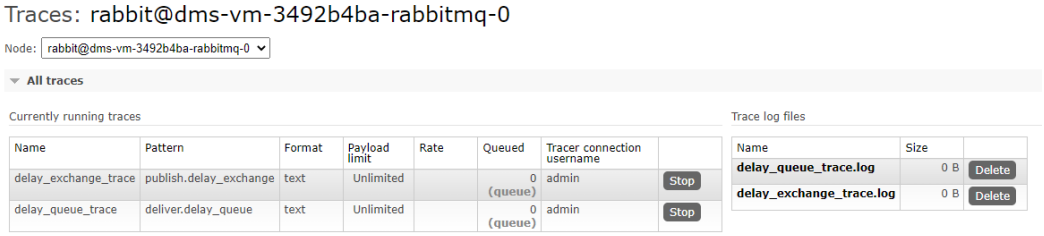
Parameter	Description
Pattern	Matching pattern. Options: • **#**: Trace all messages entering and leaving RabbitMQ. • **publish#**: Trace all messages entering RabbitMQ. • **deliver#**: Trace all messages leaving RabbitMQ. • **publish.***delay_exchange*: Trace messages entering a specified exchange. *delay_exchange* indicates an exchange name. Change it to the actual value. • **deliver.***delay_queue*: Trace messages entering a specified queue. *delay_queue* indicates a queue name. Change it to the actual value.

Figure 6-2 Adding a trace



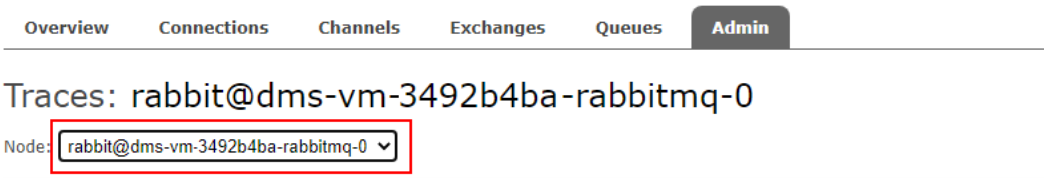
After the trace is created, it is displayed in the **All traces** area.

Figure 6-3 Trace list



Step 6 (Optional) For a cluster RabbitMQ instance, switch nodes by specifying **Node** and repeat **Step 5** to create traces for them.

Figure 6-4 Switching nodes



Step 7 After message logs are stored in the trace log file, click the trace log file name to view the log content.

Figure 6-5 Trace log files

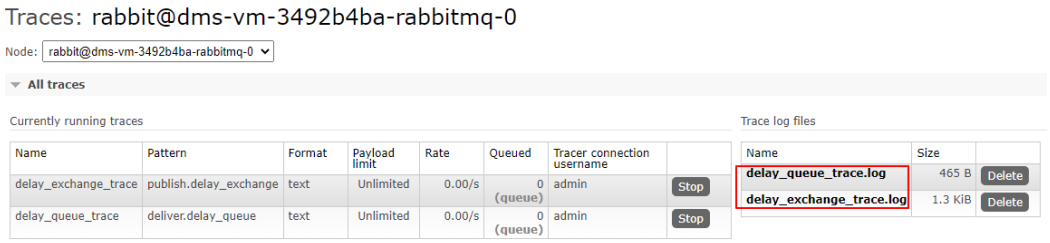


Figure 6-6 shows the content of **delay_exchange_trace.log**.

Figure 6-6 delay_exchange_trace.log

```
================================================================================
2022-07-20 3:22:32:837: Message published

Node:          rabbit@dms-vm-3492b4ba-rabbitmq-0
Connection:    <rabbit@dms-vm-3492b4ba-rabbitmq-0.1657790484.10274.7>
Virtual host:  /
User:          admin
Channel:       1
Exchange:      delay_exchange
Routing keys:  [<<>>]
Routed queues:  []
Properties:    [{<<"delivery_mode">>,signedint,2},{<<"headers">>,table,[]}]
Payload:
hello world
```

Figure 6-7 shows the content of **delay_queue_trace.log**.

Figure 6-7 delay_queue_trace.log

```
================================================================================
2022-07-20 3:23:22:468: Message received

Node:          rabbit@dms-vm-3492b4ba-rabbitmq-0
Connection:    <rabbit@dms-vm-3492b4ba-rabbitmq-0.1657790484.10565.7>
Virtual host:  /
User:          admin
Channel:       1
Exchange:
Routing keys:  [<<"delay_queue">>]
Queue:         delay_queue
Properties:    [{<<"delivery_mode">>,signedint,1},{<<"headers">>,table,[]}]
Payload:
hello world
```

----End

7 Managing Virtual Hosts

7.1 Creating a Virtual Host

Scenario

Each virtual host serves as an independent RabbitMQ server. Virtual hosts provide logical separation of exchanges, queues, and bindings. Different applications run on different virtual hosts without interfering with each other. An instance can have multiple virtual hosts, and a virtual host can have multiple exchanges and queues. To connect a producer or consumer to a RabbitMQ instance, you must specify a virtual host. For details, see **Virtual Hosts** on the official RabbitMQ website.

Methods of creating a virtual host:

- **Method 1: By using the console**
- **Method 2: By using the RabbitMQ management UI**
- **Method 3: By calling an API**

Method 1: Creating a Virtual Host on the Console

Step 1 Log in to the management console.

Step 2 In the upper left corner, click  and select a region.

NOTE

Select the region where your RabbitMQ instance is.

Step 3 Click  and choose **Application** > **Distributed Message Service for RabbitMQ** to open the console of DMS for RabbitMQ.

Step 4 Click an instance to go to the details page.

Step 5 In the navigation pane, choose **Virtual Hosts**.

Step 6 Click **Create Virtual Host**.

Step 7 Enter a virtual host name and click **OK**.

After the creation is successful, the new virtual host is displayed in the virtual host list.

Tracing indicates whether message tracing is enabled. If it is enabled, you can trace the message forwarding path.

NOTE

- The virtual host name cannot be modified once specified.
- After an instance is created, a virtual host named **/** is automatically created.

----End

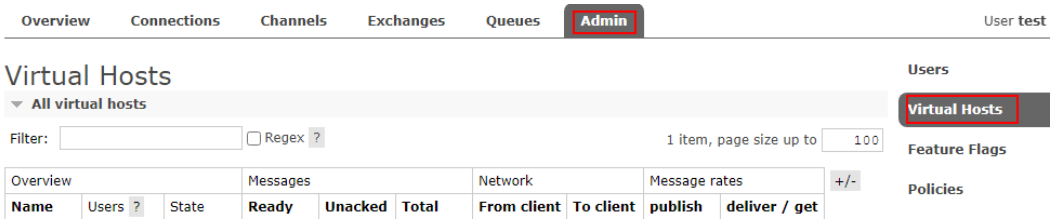
Method 2: Creating a Virtual Host on the RabbitMQ Management UI

Step 1 Log in to the **RabbitMQ management UI**.

Step 2 On the top navigation bar, choose **Admin**.

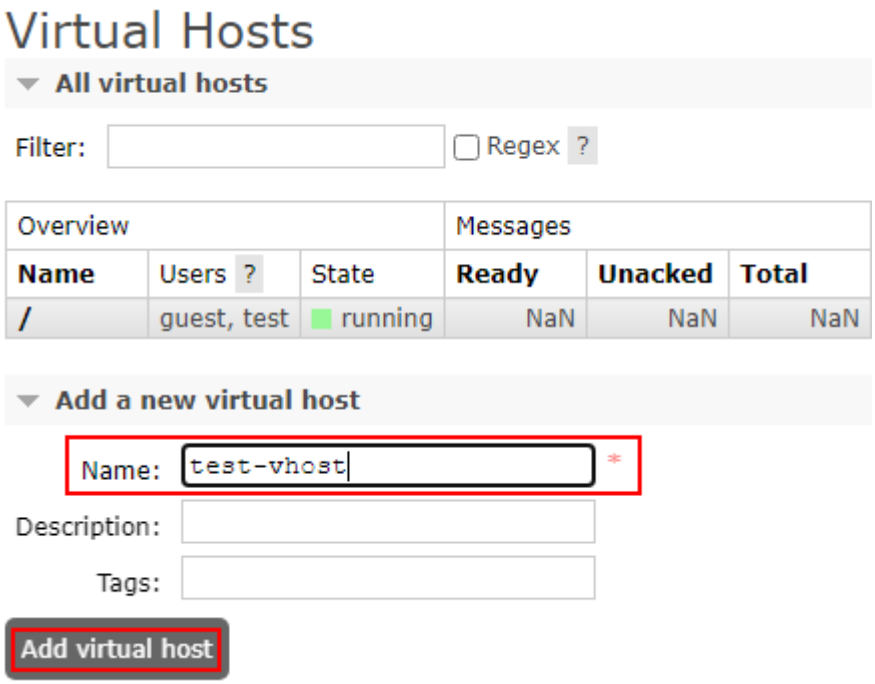
Step 3 In the navigation tree on the right, choose **Virtual Hosts**.

Figure 7-1 Virtual Hosts



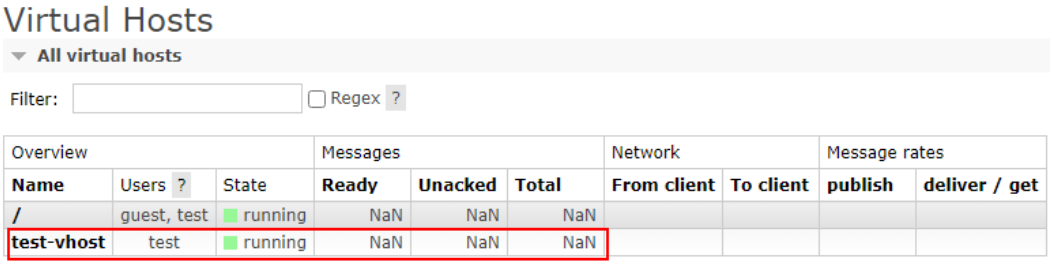
Step 4 In the **Add a new virtual host** area, enter the virtual host name and click **Add virtual host**.

Figure 7-2 Creating a virtual host (management UI)



After the creation is successful, the new virtual host is displayed in the **All virtual hosts** area.

Figure 7-3 Virtual host list (management UI)



----End

Method 3: Creating a Virtual Host by Calling an API

Step 1 In Linux, **connect to the RabbitMQ instance**.

Step 2 Run the following command to create a virtual host:

```
curl -i -X PUT http://${USERNAME}:${PASSWORD}@${HOST}:${PORT}/api/vhosts/${VHOST_NAME}
```

Parameter description:

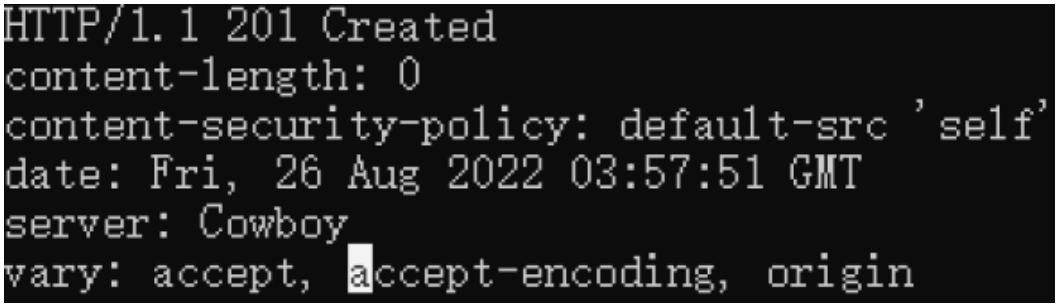
- **USERNAME**: username set when the RabbitMQ instance was created. View the username in the **Connection** area on the instance details page.
- **PASSWORD**: password set when the RabbitMQ instance was created. If you forget the password, **reset it**.
- **HOST**: IP address of the management UI. View the management UI address in the **Connection** area on the instance details page.
- **PORT**: port of the management UI. View the management UI port in the **Connection** area on the instance details page.
- **VHOST_NAME**: name of the virtual host to be created.

Example:

```
curl -i -X PUT http://root:txxxt@192.168.1.3:15672/api/vhosts/vhost-demo
```

If the creation is successful, the following information is displayed.

Figure 7-4 Virtual host created successfully



----End

7.2 Deleting a Virtual Host

Scenario

Methods of deleting a virtual host:

- **Method 1: By using the console**
- **Method 2: By using the RabbitMQ management UI**
- **Method 3: By calling an API**

Method 1: Deleting a Virtual Host on the Console

Step 1 Log in to the management console.

Step 2 In the upper left corner, click and select a region.

NOTE

Select the region where your RabbitMQ instance is.

Step 3 Click and choose **Application** > **Distributed Message Service for RabbitMQ** to open the console of DMS for RabbitMQ.

Step 4 Click an instance to go to the details page.

Step 5 In the navigation pane, choose **Virtual Hosts**.

Step 6 Delete virtual hosts using either of the following methods:

- Select one or more virtual hosts and click **Delete Virtual Host** in the upper left corner.
- In the row containing the virtual host you want to delete, click **Delete**.

Step 7 In the confirmation dialog box that is displayed, click **Yes**.

----End

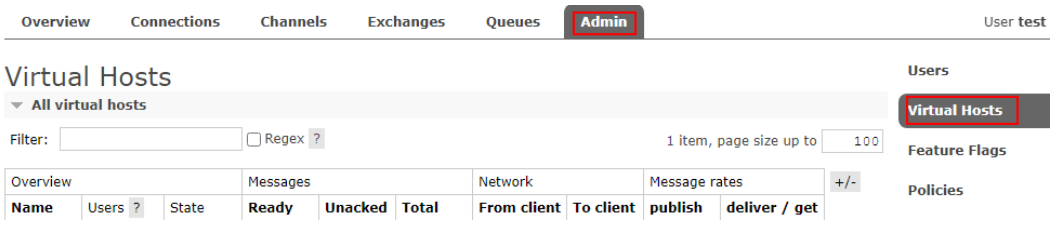
Method 2: Deleting a Virtual Host on the RabbitMQ Management UI

Step 1 Log in to the **RabbitMQ management UI**.

Step 2 On the top navigation bar, choose **Admin**.

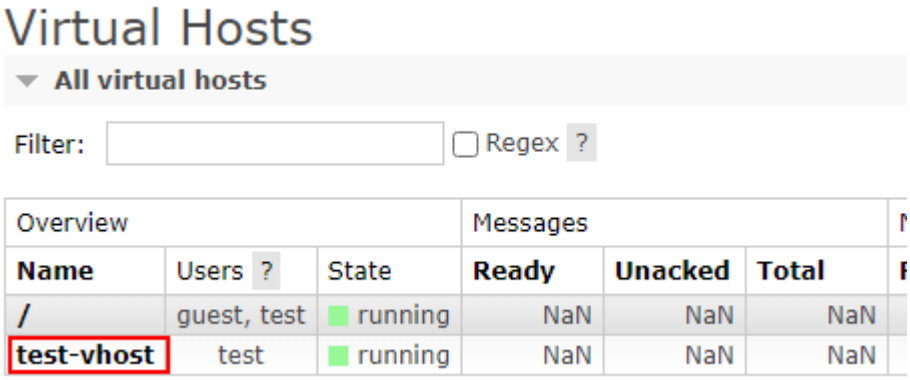
Step 3 In the navigation tree on the right, choose **Virtual Hosts**.

Figure 7-5 Virtual Hosts page



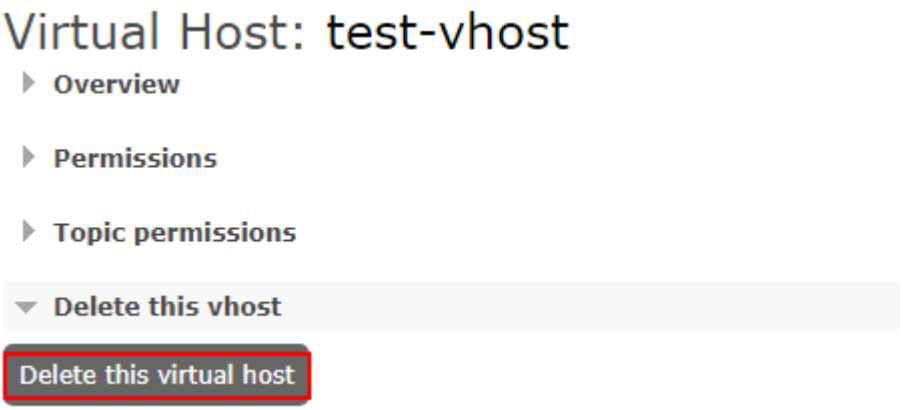
Step 4 Click the name of the virtual host to be deleted.

Figure 7-6 Virtual host to be deleted



Step 5 In the **Delete this vhost** area, click **Delete this virtual host**. A confirmation dialog box is displayed.

Figure 7-7 Deleting a virtual host



Step 6 Click **OK**.

----End

Method 3: Deleting a Virtual Host by Calling an API

Step 1 In Linux, **connect to the RabbitMQ instance**.

Step 2 Run the following command to delete a virtual host:

```
curl -i -X DELETE http://${USERNAME}:${PASSWORD}@${HOST}:${PORT}/api/vhosts/${VHOST_NAME}
```

Parameter description:

- **USERNAME**: username set when the RabbitMQ instance was created. View the username in the **Connection** area on the instance details page.
- **PASSWORD**: password set when the RabbitMQ instance was created. If you forget the password, **reset it**.
- **HOST**: IP address of the management UI. View the management UI address in the **Connection** area on the instance details page.

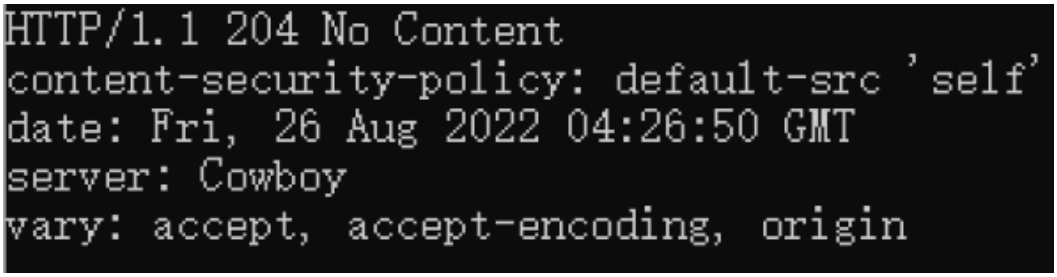
- **PORT**: port of the management UI. View the management UI port in the **Connection** area on the instance details page.
- **VHOST_NAME**: name of the virtual host to be deleted.

Example:

```
curl -i -X DELETE http://root:txxxt@192.168.1.3:15672/api/vhosts/vhost-demo
```

If the deletion is successful, the following information is displayed.

Figure 7-8 Virtual host deleted successfully



----End

8 Advanced Features

8.1 Lazy Queues

Scenario

By default, messages produced by RabbitMQ producers are stored in the memory. When the memory needs to be released, the messages will be paged out to the disk. Paging takes a long time, during which queues cannot process messages.

If production is too fast (for example during batch processing) or consumers cannot consume messages for a long time due to various reasons (for example when consumers are offline or broke down), a large number of messages will be stacked. Memory usage becomes high and paging is frequently triggered, which may affect message sending and receiving of other queues. In this case, you are advised to use lazy queues.

Lazy queues store as many messages to the disk as possible. Messages are loaded to the memory only when they are being consumed. This reduces memory consumption, but increases I/O and affects single-queue throughput. An important goal of lazy queues is to support long queues, that is, queues with many messages stored or stacked.

Lazy queues are recommended in the following scenarios:

- Messages may be stacked in queues.
- There is no high requirement on the queue performance (throughput), for example, less than 10,000 TPS.
- Stable production and consumption are desired, without being affected by memory changes.

Lazy queues are not suitable in the following scenarios:

- High RabbitMQ performance is expected.
- Queues are always short (with no messages stacked).
- The queue length limit is configured in a policy.

For more information about lazy queues, see **Lazy Queues**.

Configuring a Lazy Queue

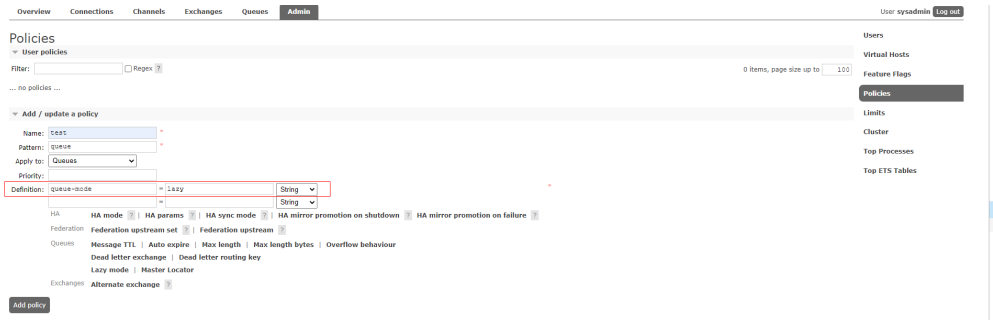
A queue has two modes: **default** and **lazy**. The default mode is **default**. To configure a queue to be **lazy**, you can use the **channel.queueDeclare** argument or a policy. If both these methods are used, the configuration set by the policy takes precedence.

- **The following example shows how to set a lazy queue by using channel.queueDeclare on a Java client.**

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

- The following figure shows how to use a policy to set a lazy queue on the **RabbitMQ management UI**.

Figure 8-1 Using a policy to set a lazy queue



8.2 Message Persistence

Scenario

By default, messages produced by RabbitMQ producers are stored in the memory. When a node breaks down or restarts, how can we ensure that messages are not lost? In RabbitMQ, you can configure persistence for exchanges, queues, and messages. **Persistence means writing messages in the memory to the disk to prevent them from being lost due to exceptions. However, if message persistence is enabled, RabbitMQ performance deteriorates because read and write operations are much slower in disks than in memory.** Different from the lazy queue mechanism, a persisted message is stored in both the disk and memory. It is deleted from the memory only when the memory is insufficient.

NOTICE

- Non-persistent queues and exchanges are lost after a restart.
- Non-persistent messages are lost after a restart. (Messages that are sent to persistent queues or exchanges will not automatically become persistent.)
- A message will be lost if the server restarts before the message persistence is complete.

Configuring Exchange Persistence

When creating an exchange on the **RabbitMQ management UI**, set **durable** to **true**, as shown in **Figure 8-2**. **Figure 8-3** shows a successful configuration.

Figure 8-2 Configuring exchange persistence

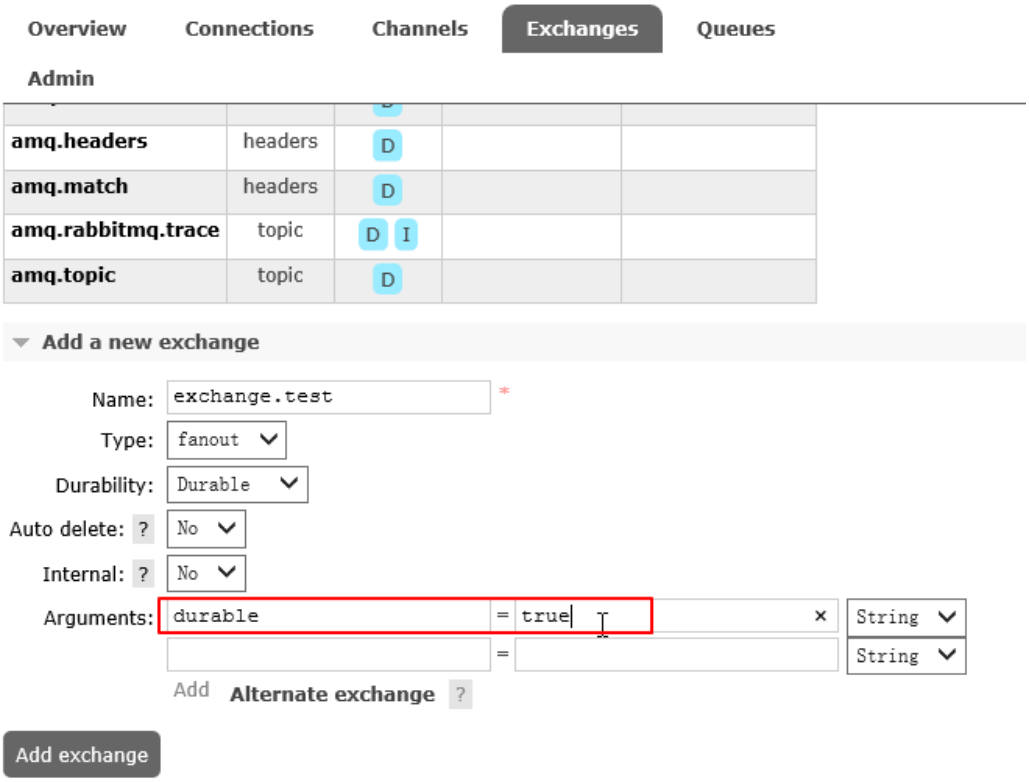
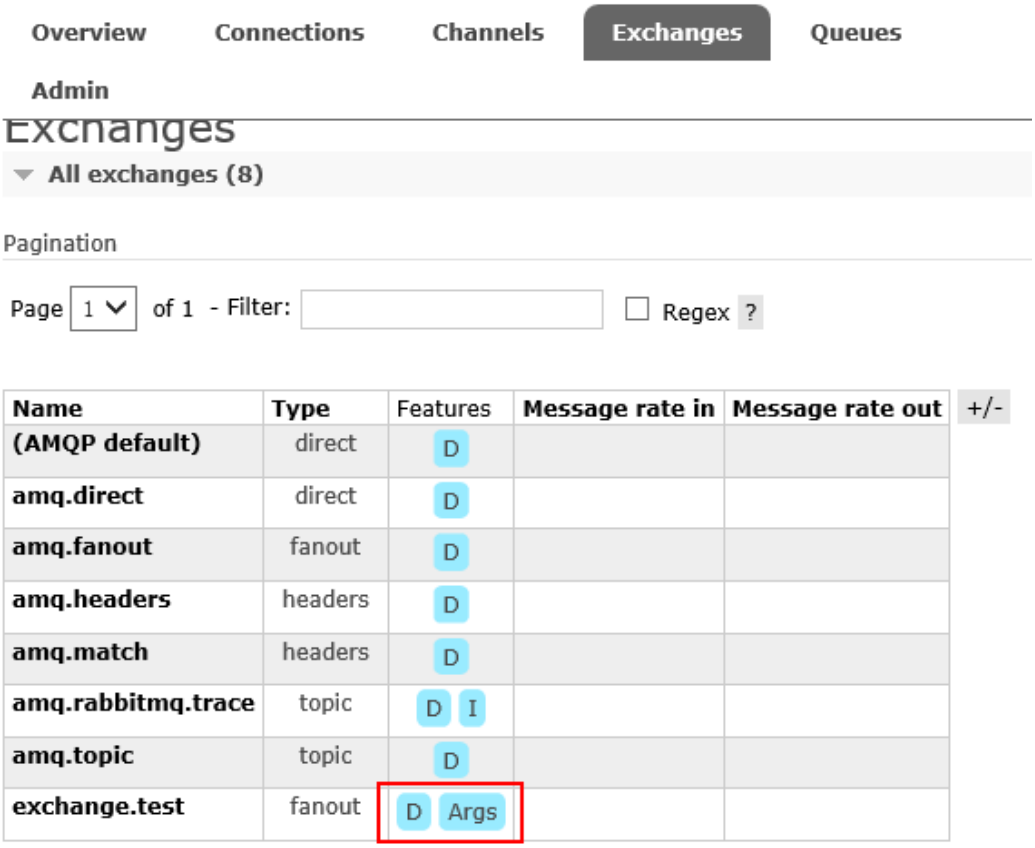


Figure 8-3 Exchange persistence configured



Configuring Queue Persistence

When creating a queue on the **RabbitMQ management UI**, set **durable** to **true**, as shown in **Figure 8-4**. **Figure 8-5** shows a successful configuration.

Figure 8-4 Configuring queue persistence

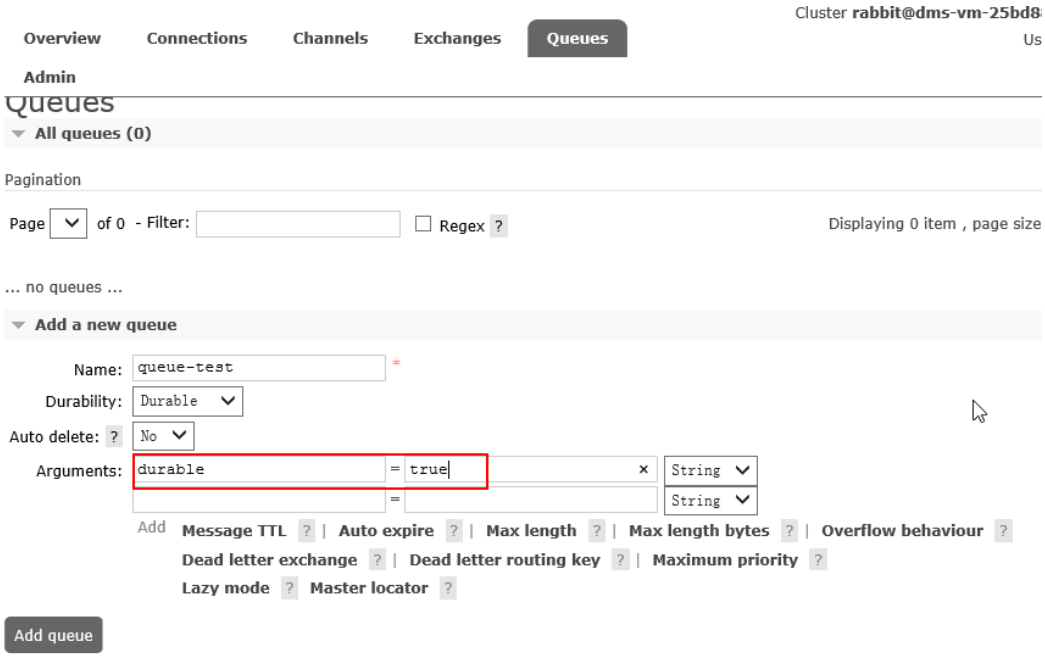
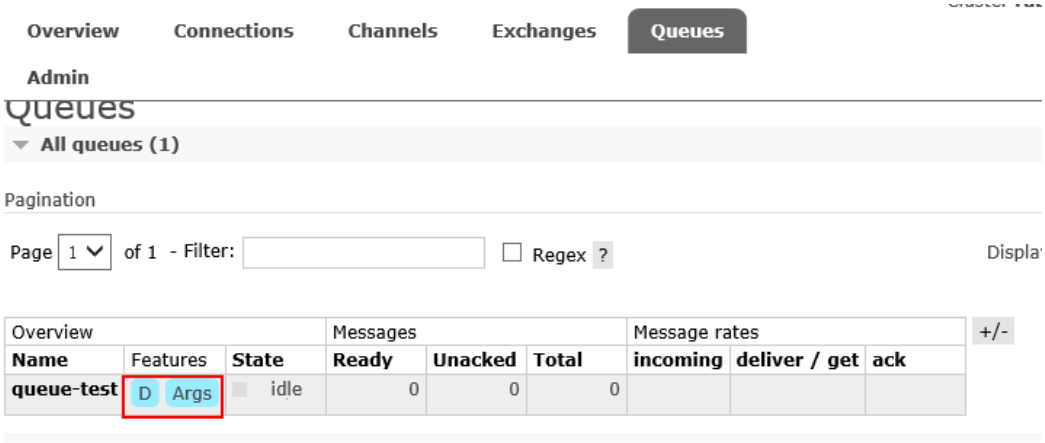


Figure 8-5 Queue persistence configured



Configuring Message Persistence

After configuring queue persistence, set **MessageProperties** to **PERSISTENT_TEXT_PLAIN** to send persistent messages to the queue.

The following example shows how to configure message persistence on a Java client.

```
import com.rabbitmq.client.MessageProperties;
channel.basicPublish("", "my_queue",MessageProperties.PERSISTENT_TEXT_PLAIN, message.getBytes());
```


8.3 Dead Lettering and TTL

Dead lettering and time to live (TTL) are two RabbitMQ features that must be used with caution because they may adversely affect system performance.

Dead Lettering

Dead lettering is a message mechanism in RabbitMQ. When a message is consumed, it becomes a dead letter message if any of the following happens:

- **requeue** is set to **false**, and the consumer uses **basic.reject** or **basic.nack** to negatively acknowledge (NACK) the message.
- The message has stayed in the queue for longer than the configured TTL.
- The number of messages in the queue exceeds the maximum queue length.

Such a message will be stored in a dead letter queue, if any, for special treatment. If there is no dead letter queue, the message will be discarded.

For more information about dead lettering, see **Dead Letter Exchanges**.

Configuring dead letter exchanges and routing information using queue parameters

To configure a dead letter exchange for a queue, specify the **x-dead-letter-exchange** and **x-dead-letter-routing-key** parameters when declaring the queue. The queue sends the dead letter message to the dead letter exchange based on **x-dead-letter-exchange** and sets the dead letter routing key for the dead letter message based on **x-dead-letter-routing-key**.

The following example shows how to configure a dead letter exchange and routing information on a Java client.

```
channel.exchangeDeclare("some.exchange.name", "direct");

Map<String, Object> args = new HashMap<String, Object>();
args.put("x-dead-letter-exchange", "some.exchange.name");
args.put("x-dead-letter-routing-key", "some-routing-key");
channel.queueDeclare("myqueue", false, false, false, args);
```

TTL

TTL indicates the expiration time. You can configure TTL for messages and queues. Message TTL can be configured using the following methods:

- Configure a TTL in queue properties: All messages in the queue have the same expiration time.
- Configure a TTL for each message: Each message has a dedicated TTL.

If both methods are used, the smaller TTL value is used.

If a message that has stayed in a queue for longer than the TTL, the message will be discarded. If a dead letter exchange has been configured for the queue, the message will be sent to the dead letter exchange, and then routed to the dead letter queue.

For more information about TTL, see **TTL**.

Configuring queue TTL

The **x-expires** parameter in the **channel.queueDeclare** argument is used to control after how long of being unused a queue is automatically deleted. "Unused" indicates that there is no consumer in the queue, the queue is not re-declared, and the **Basic.Get** command is not invoked during this period. The value of **x-expires** must be a non-zero integer, in milliseconds.

The following example shows how to configure a queue TTL on a Java client.

```
Map<String, Object> args = new HashMap<String, Object>();
args.put("x-expires", 1800000);
channel.queueDeclare("myqueue", false, false, false, args);
```

Configuring message TTL

Configure a TTL in queue properties: Add the **x-message-ttl** parameter to the **channel.queueDeclare** argument. The value of this parameter must be a non-zero integer, in milliseconds.

The following example shows how to configure a message TTL by setting queue properties on a Java client.

```
Map<String,Object> arg = new HashMap<String, Object>();
arg.put("x-message-ttl",6000);
channel.queueDeclare("normalQueue",true,false,false,arg);
```

Configure a TTL for each message: Add the **expiration** parameter to the **channel.basicPublish** argument. The value of this parameter must be a non-zero integer, in milliseconds.

The following example shows how to set a per-message TTL on a Java client.

```
byte[] messageBodyBytes = "Hello, world!".getBytes();
AMQP.BasicProperties properties = new AMQP.BasicProperties.Builder()
                                  .expiration("60000")
                                  .build();
channel.basicPublish("my-exchange", "routing-key", properties, messageBodyBytes);
```

8.4 Message Acknowledgment

Scenario

How does a producer confirm that a message has been properly published to the server? How does the server confirm that a message has been successfully consumed? The answer is RabbitMQ's acknowledgement mechanism.

Acknowledgments by publishers ("publisher confirms") and consumers are critical to ensure data reliability. If a connection fails, messages being transmitted may be lost and need to be transmitted again. The acknowledgment mechanism enables the server and client to know when to retransmit messages. A client may acknowledge a message upon receipt of the message, or after it has completely processed the message. **Producer confirms affect performance and should be disabled if high throughput is required. However, disabling producer confirms leads to lower reliability.**

For details about the message acknowledgment mechanism, see **Consumer Acknowledgment and Publisher Confirms**.

Producer Confirms

The server confirms that it has received the message sent from the producer.

The following example shows how to configure publisher confirms on a Java client.

```
try {
   channel.confirmSelect(); // Enable publisher confirms on the channel.
   // Send messages normally.
   channel . basicPublish( "exchange " , " routingKey" , null , "publisher confirm test " .getBytes());
   if (!channel.waitForConfirms()) {
      System.out.println( "send message failed " ) ;
      // do something else....
   }
} catch (InterruptedException e) {
      e.printStackTrace() ;
}
```

After the **channel .waitForConfirms** method is called, the system waits for a confirmation from the server. Such synchronous waiting affects performance, but is necessary if the publisher requires at-least-once delivery.

Consumer Acknowledgment

The server determines whether to delete a message from a queue based on whether the message is successfully received by the consumer.

Consumer acknowledgments are important to ensure data reliability. Consumer applications should take enough time to process important messages before acknowledging the messages. In this way, we do not have to worry about message losses caused by consumer process exceptions (such as program breakdown and restart) during message processing.

Consumer acknowledgment can be enabled by using the **basicConsume** method. In most cases, consumer acknowledgments are enabled on channels.

The following example shows how to configure consumer acknowledgments on a Java client (using **Channel#basicAck** and **basic.ack** for positive acknowledgment):

```
// 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);
        }
    });
```

Unacknowledged messages are cached in the memory. If there are too many unacknowledged messages, the memory usage becomes high. In this case, you

can limit the number of messages prefetched by the consumer. For details, see **Prefetch**.

8.5 Prefetch

Scenario

Prefetch limits the number of unacknowledged messages. Once a consumer has more unacknowledged messages than the prefetch limit, the server stops sending messages to the consumer, unless at least one message is acknowledged. Prefetch is essentially a flow control measure on consumers.

Consider the following factors when setting prefetch:

- If the limit is too low, the performance may be affected, because RabbitMQ keeps waiting for the permission to send messages.
- If the limit is too high, a large number of messages may be transmitted to a consumer, leaving other consumers idle. In addition, you also need to consider consumer configurations. When processing messages, consumers save all messages in the memory. A high prefetch limit may affect consumer performance and may even crash the consumer.

For details about prefetch, see **Consumer Prefetch**.

What Is a Proper Prefetch Limit?

- If you have only one or a few consumers processing messages, it is recommended that you prefetch multiple messages at a time to keep the client busy. If your processing time and network status are stable, you can obtain an estimated prefetch value by dividing the total round-trip time by the processing time of each message on the client.
- If you have a large number of consumers and the processing time is short, a low prefetch limit is recommended. However, if the limit is too low, consumers will be idle after they have processed a batch of messages but the next batch has not yet arrived. If the limit is too high, a single consumer may be busy while other consumers are idle.
- If you have a large number of consumers and the processing time is long, set the prefetch value to **1** so that messages can be evenly distributed among all consumers.

NOTICE

If automatic message acknowledgment has been configured on the client, the prefetch value is invalid, and acknowledged messages are deleted from queues.

Setting the Prefetch Value

The following example shows how to set the prefetch value to **10** for a single consumer on a Java client.


```
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);
```

On a Java client, the default value of **global** is **false**. Therefore, the preceding example can be simply written as **channel.basicQos(10)**.

The values of **global** are described as follows.

Table 8-1 Description of global values

Value of global	Description
false	Applied separately to each new consumer on the channel.
true	Shared among all consumers on the channel.

8.6 Heartbeat Detection

RabbitMQ heartbeats help the application layer detect interrupted connections and unresponsive peers in a timely manner. Heartbeats also prevent some network devices from disconnecting TCP connections where there is no activity for a certain period of time.

Heartbeat Timeout

The heartbeat timeout defines after how long the peer TCP connection is considered closed by the server or client.

After a timeout is set on the RabbitMQ server and client separately, the server and client negotiate the timeout value. The value must be configured for the client so that it can request heartbeats. The Java, .NET, and Erlang clients maintained by RabbitMQ use the following negotiation logic:

- If the heartbeat timeout set on neither the server nor the client is **0**, the smaller value is used.
- If the heartbeat timeout is set to **0** on either the server or the client, the non-zero value is used.
- If the heartbeat timeout set on both the server and the client is **0**, heartbeats are disabled.

For more information about heartbeat detection, see **Detecting Dead TCP Connections with Heartbeats and TCP Keepalives**.

Heartbeat Frames

The interval for sending heartbeat frames (also called a heartbeat interval) is half of the heartbeat timeout. After a client misses two heartbeats, it is considered

unreachable. Different clients show this differently, but the TCP connection will be closed. If the client detects that the server cannot be accessed due to heartbeats, the client needs to reconnect to the server.

Any traffic, including protocol operations, message publishing, message acknowledgment, and heartbeat frames, is considered as a valid heartbeat. If there is any other traffic on the connection, the client can choose to send heartbeat frames or not. If there is no other traffic on the connection, the client must send heartbeat frames.

Configuring Heartbeat Timeout on the Client

If the heartbeat timeout is set to a small value, false alarms may be reported in the case of temporary network congestion or temporary server traffic control. In most cases, it is recommended that the timeout be set to 5 to 20 seconds.

- Java client

  Before creating a connection, configure the **ConnectionFactory#setRequestedHeartbeat** parameter. Example:

  ```
  ConnectionFactory cf = new ConnectionFactory();
  // The heartbeat timeout is 15 seconds.
  cf.setRequestedHeartbeat(15);
  ```

- .NET client

  ```
  var cf = new ConnectionFactory();
  // The heartbeat timeout is 15 seconds.
  cf.RequestedHeartbeat = TimeSpan.FromSeconds(15);
  ```

8.7 Single Active Consumer

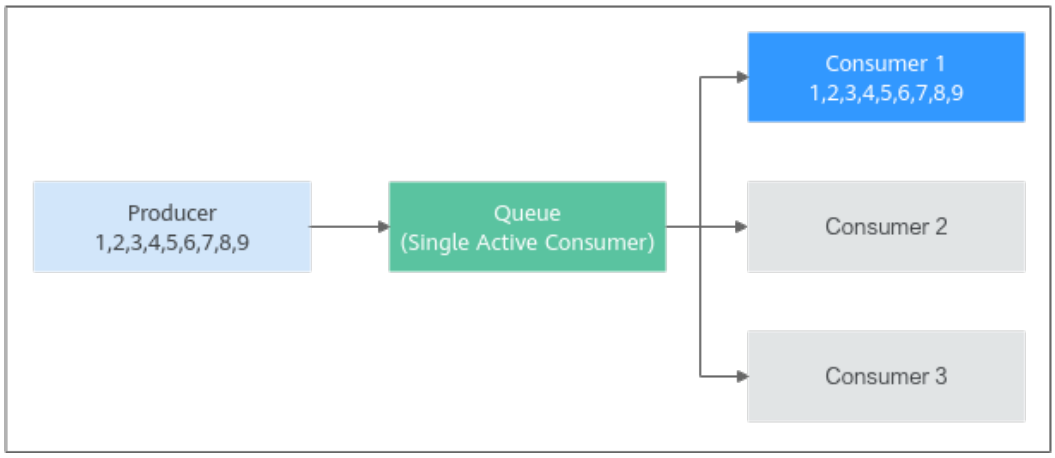
Scenario

A queue can have multiple registered consumers, but single active consumer allows only one consumer to consume messages from the queue. Another consumer can consume messages only when the active one is abnormal. Single active consumer can be used when the message consumption sequence must be ensured and high reliability is required.

NOTICE

Single active consumer is available only in RabbitMQ 3.8.35.

Figure 8-6 Single active consumer



In **Figure 8-6**, Producer produces nine messages. Due to the setting of single active consumer, only Consumer 1 can consume messages.

For more information about single active consumer, see **Single Active Consumer**.

Configuration

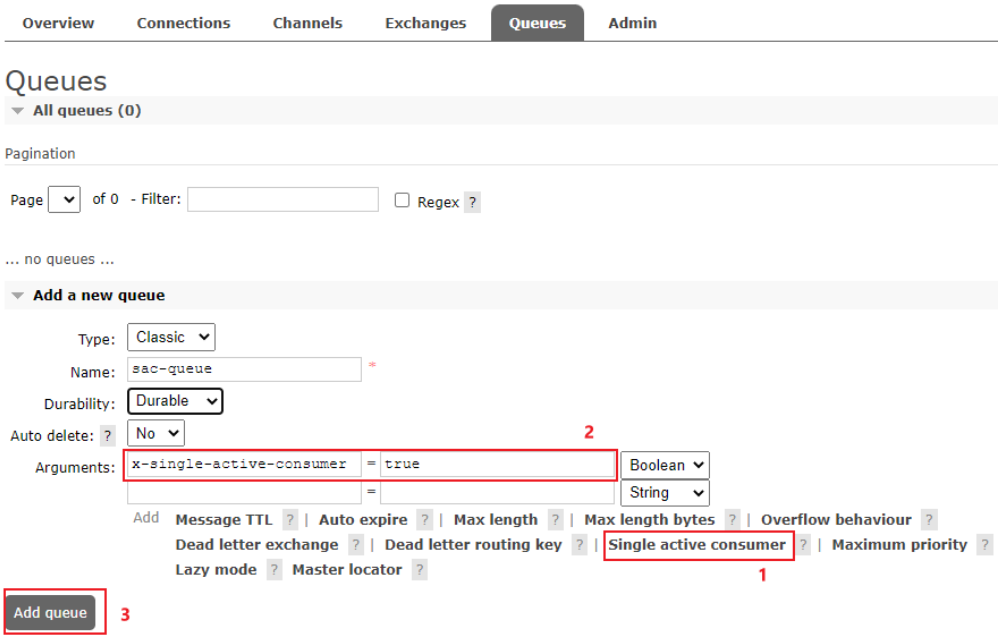
When declaring a queue, you can configure a single active consumer by setting the **x-single-active-consumer** parameter to **true**.

- **The following example shows how to configure single active consumer on a Java client.**

```
Channel ch = ...;
Map<String, Object> arguments = newHashMap<String, Object>();
arguments.put("x-single-active-consumer", true);
ch.queueDeclare("my-queue", false, false, false, arguments);
```

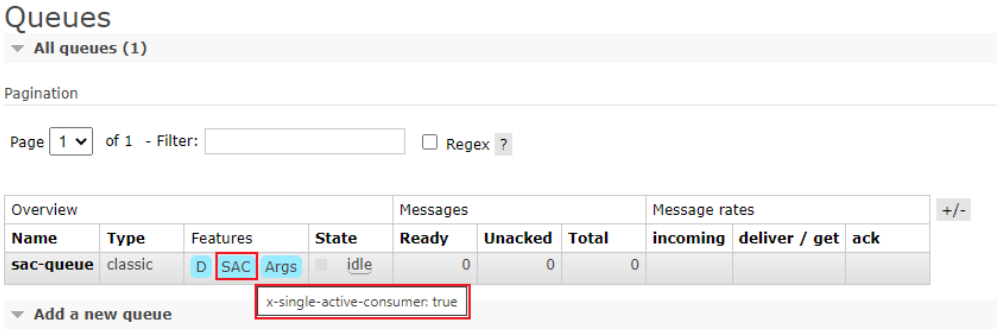
- **The following example shows how to configure single active consumer on the RabbitMQ management UI.**

Figure 8-7 Configuring single active consumer



After the setting is complete, check whether the queue features contain single active consumer on the **Queues** page. As shown in **Figure 8-8**, **SAC** indicates single active consumer.

Figure 8-8 Viewing queue features



8.8 Quorum Queues

Scenario

Quorum queues provide the queue replication capability to ensure high data availability and security. Quorum queues can be used to replicate queue data between RabbitMQ nodes, ensuring that queues can still run when a node breaks down.

Quorum queues can be used in scenarios where queues exist for a long time and there are high requirements on queue fault tolerance and data security and low requirements on latency and queue features. Quorum queues are not recommended if a large number of messages may be stacked, because write amplification significantly increases disk usage.

Messages in quorum queues are preferentially stored in the memory. You are advised to limit the queue length and memory usage of quorum queues. When the number of stacked messages exceeds the threshold, the messages are written to the disk to avoid high memory watermark.

For more information about quorum queues, see **Quorum Queues**.

NOTICE

Quorum queues are available only in RabbitMQ 3.8.35.

Comparing Quorum Queues and Mirrored Queues

Quorum queues are introduced in RabbitMQ 3.8 to address the technical limitations of mirrored queues. Quorum queues have similar functions as mirrored queues and provide high-availability queues for RabbitMQ.

Mirrored queues are slow at replicating messages.

- A mirrored queue has a queue leader and many mirrors. When a producer sends a message to the queue leader, the leader replicates the message to the mirrors, and sends back confirmation to the producer only after all mirrors have saved the message.
- If a node in a RabbitMQ cluster goes offline due to a fault, all data in the mirrors stored on the node will be lost after the fault is rectified and the node goes online again. In this case, O&M personnel need to determine whether to replica data from the queue leader to the mirrors. If they choose not to replicate data, messages may be lost. If they choose to replicate data, the queue is blocked during replication and no operation can be performed on the queue. When a large number of messages are stacked, the queue will be unavailable for several minutes, hours, or even longer.

Quorum queues can solve these problems.

- Quorum queues are based on a variant of the Raft consensus algorithm. They deliver a higher message throughput. A quorum queue has a primary replica (queue leader) and many followers. When a producer sends a message to the leader, the leader replicates the message to the followers, and sends back confirmation to the producer only after half of the followers have saved the message. This means that a small number of slow followers do not affect the performance of the entire queue. Similarly, the election of the leader requires the consent of more than half of the followers, which prevents the queue from having two leaders in the event of network partitioning. Therefore, quorum queues attach more importance to consistency than availability.
- After a node in a RabbitMQ cluster goes online after recovering from a fault, the data stored on the node will not be lost. The queue leader directly replicates messages from the position where the followers were interrupted. The replication process is non-blocking, and the entire queue is not affected by the addition of new followers.

Compared with mirrored queues, quorum queues have fewer features, as shown in **Table 8-2**. Quorum queues consume more memory and disk space.

Table 8-2 Features

Feature	Mirrored Queues	Quorum Queues
Non-durable queues	Supported	Not supported
Exclusive queues	Supported	Not supported
Message persistence	Per message	Always
Queue rebalancing	Automatic	Manual
Message TTL	Supported	Not supported
Queue TTL	Supported	Supported
Queue length limit	Supported	Supported (except **x-overflow: reject-publish-dlx**)

Feature	Mirrored Queues	Quorum Queues
Lazy queues	Supported	Supported after the queue length is limited
Message priority	Supported	Not supported
Consumption priority	Supported	Supported
Dead letter exchanges	Supported	Supported
Dynamic policy	Supported	Supported
Poison message (let consumers consume infinitely) handling	Not supported	Supported
Global QoS prefetch	Supported	Not supported

Configuration

When declaring a queue, set the **x-queue-type** queue argument to **quorum**. This argument can be set only during queue declaration and cannot be set by using a policy.

The default replication factor of a quorum queue is five.

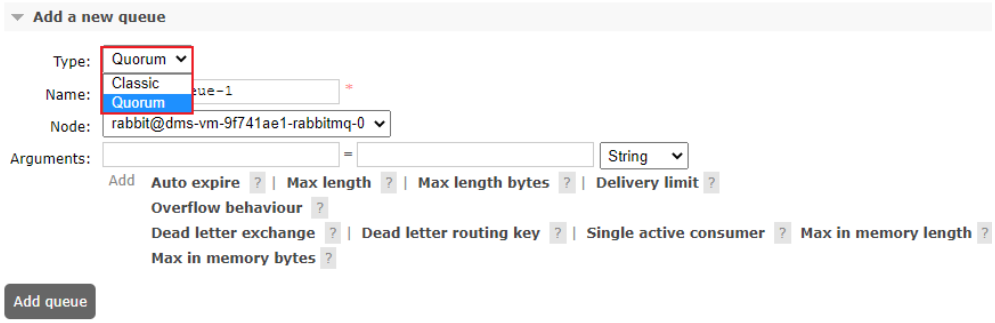
- **The following example shows how to configure a quorum queue on a Java client.**

```
ConnectionFactory factory = newConnectionFactory();
factory.setRequestedHeartbeat(30);
factory.setHost(HOST);
factory.setPort(PORT);
factory.setUsername(USERNAME);
factory.setPassword(PASSWORD);

finalConnection connection = factory.newConnection();
finalChannel channel = connection.createChannel();
// Create the queue parameter map.
Map<String, Object> arguments = newHashMap<>();
arguments.put("x-queue-type", "quorum");
// Declare the quorum queue.
channel.queueDeclare("test-quorum-queue", true, false, false, arguments);
```

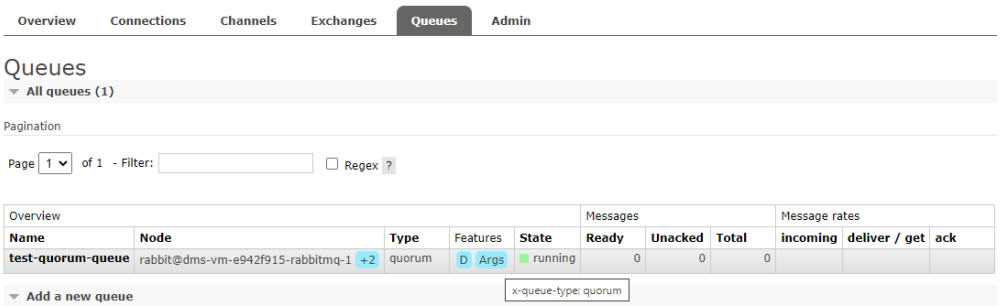
- **The following example shows how to configure a quorum queue on the RabbitMQ management UI.**

Figure 8-9 Configuring a quorum queue



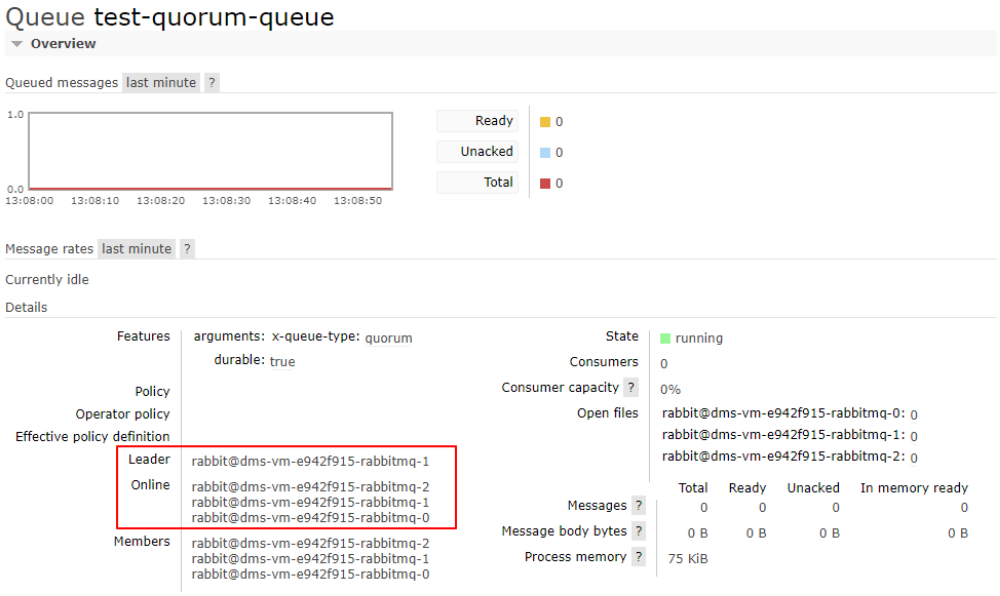
After the configuration is complete, check whether the queue type is **quorum** on the **Queues** page, as shown in **Figure 8-10**. In the **Node** column, **+2** indicates that the queue has two replicas. Blue indicates that the two replicas have been synchronized. Red indicates that some messages have not been replicated.

Figure 8-10 Checking the queue type



On the **Queues** page, click the name of the desired queue. Check the node where the leader of this quorum queue resides and the node where active followers reside.

Figure 8-11 Queue details page



Configuring the Quorum Queue Length

You can configure a policy or queue attributes to limit the length of a quorum queue and the length that can be stored in the memory.

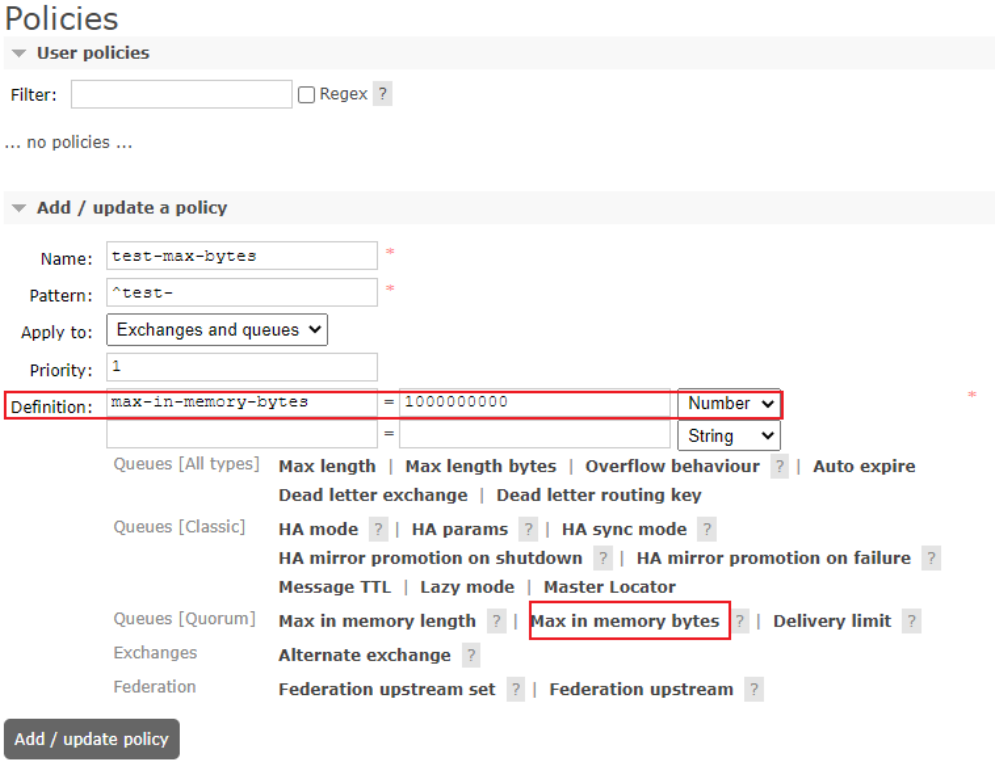
- **x-max-length**: maximum number of messages in the quorum queue. If this limit is exceeded, excess messages will be discarded or sent to the dead letter exchange.
- **x-max-length-bytes**: maximum size (in bytes) of messages in the quorum queue. If this limit is exceeded, excess messages will be discarded or sent to the dead letter exchange.

- **x-max-in-memory-length**: maximum number of messages of the quorum queue that can be stored in memory.
- **x-max-in-memory-bytes**: maximum size (in bytes) of messages of the quorum queue that can be stored in memory.

The following describes how to limit the length of a quorum queue stored in the memory by configuring a policy or the queue attribute.

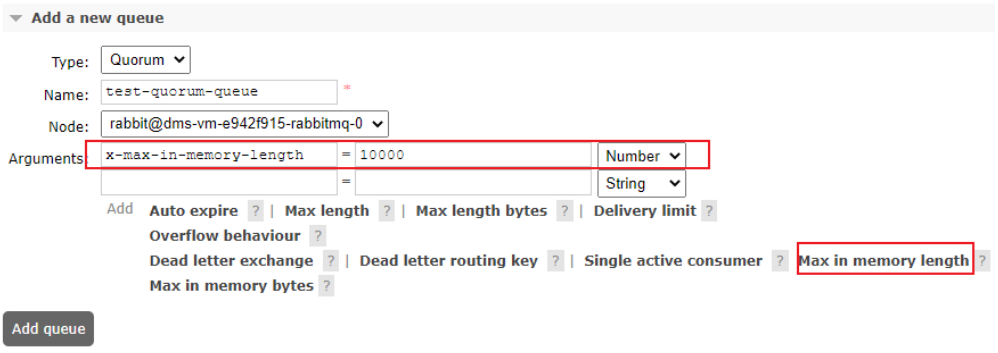
- By using a policy (recommended)

  Figure 8-12 Using a policy to set x-max-in-memory-bytes

- By configuring the queue argument

  Figure 8-13 Setting x-max-in-memory-length in the queue argument

9 Quotas

What Is Quota?

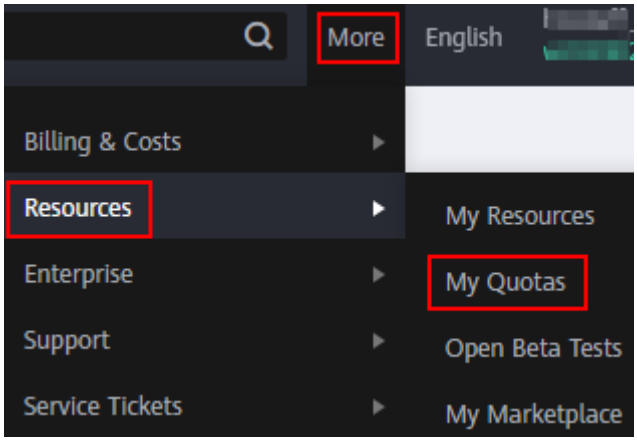
A quota is a limit on the quantity or capacity of a certain type of service resources that you can use, for example, the maximum number of RabbitMQ instances that you can create.

If the current resource quota cannot meet your service requirements, you can apply for a higher quota.

How Do I View My Quota?

1. Log in to the management console.
2. In the upper right corner of the page, choose **Resources** > **My Quotas**.
   The **Quotas** page is displayed.

   Figure 9-1 My Quotas



3. On the **Quotas** page, view the used and total quotas of resources.
   If a quota cannot meet your needs, apply for a higher quota by performing the following operations.

How Do I Increase My Quota?

1. Log in to the management console.

2. In the upper right corner of the page, choose **Resources** > **My Quotas**.

   The **Service Quota** page is displayed.

3. Click **Increase Quota**.
4. On the **Create Service Ticket** page, set the parameters.

   In the **Problem Description** area, enter the required quota and the reason for the quota adjustment.

5. Read the agreements and confirm that you agree to them, and then click **Submit**.

10 Monitoring

10.1 RabbitMQ Metrics

Introduction

This section describes metrics reported by DMS for RabbitMQ to Cloud Eye as well as their namespaces and dimensions. You can use the Cloud Eye console to query the metrics and alarms of RabbitMQ instances. You can also view the metrics on the **Monitoring** page of the RabbitMQ console.

Namespace

SYS.DMS

Instance Metrics

Table 10-1 Instance metrics

Metric ID	Metric Name	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
connections	Connections	Number of connections in the RabbitMQ instance Unit: count	≥ 0	RabbitMQ instance	1 minute
channels	Channels	Number of channels in the RabbitMQ instance Unit: count	0–2047	RabbitMQ instance	1 minute

Metric ID	Metric Name	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
queues	Queues	Number of queues in the RabbitMQ instance Unit: count	0–1200	RabbitMQ instance	1 minute
consumers	Consumers	Number of consumers in the RabbitMQ instance Unit: count	0–1200	RabbitMQ instance	1 minute
messages_ready	Available Messages	Number of messages that can be retrieved in the RabbitMQ instance Unit: count	0–10,000,000	RabbitMQ instance	1 minute
messages_unacknowledged	Unacked Messages	Number of messages that have been retrieved but not acknowledged in the RabbitMQ instance Unit: count	0–10,000,000	RabbitMQ instance	1 minute
publish	Publish	Rate at which messages are created in the RabbitMQ instance Unit: count/s	0–25,000	RabbitMQ instance	1 minute
deliver	Deliver (manual ack)	Rate at which messages are retrieved (and manually acknowledged) in a RabbitMQ instance Unit: count/s	0–25,000	RabbitMQ instance	1 minute
deliver_no_ack	Deliver (auto ack)	Rate at which messages are retrieved (and automatically acknowledged) in a RabbitMQ instance. Unit: count/s	0–50,000	RabbitMQ instance	1 minute

Node Metrics

Table 10-2 Node metrics

Metric ID	Metric Name	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
fd_used	File Handles	Number of file handles used by RabbitMQ in the node Unit: count	0–65,535	RabbitMQ instance node	1 minute
socket_used	Socket Connections	Number of socket connections used by RabbitMQ in the node Unit: count	0–50,000	RabbitMQ instance node	1 minute
proc_used	Erlang Processes	Number of Erlang processes used by RabbitMQ in the node Unit: count	0–1,048,576	RabbitMQ instance node	1 minute
mem_used	Memory Usage	Memory usage of RabbitMQ in the node Unit: byte	0–32,000,000,000	RabbitMQ instance node	1 minute
disk_free	Available Memory	Available memory of RabbitMQ in the node Unit: byte	0–500,000,000,000	RabbitMQ instance node	1 minute
rabbitmq_alive	Node Alive	Whether the RabbitMQ node is alive	1: alive 0: not alive	RabbitMQ instance node	1 minute
rabbitmq_disk_usage	Disk Capacity Usage	Disk usage of the RabbitMQ VM Unit: %	0–100%	RabbitMQ instance node	1 minute
rabbitmq_cpu_usage	CPU Usage	CPU usage of the RabbitMQ VM Unit: %	0–100%	RabbitMQ instance node	1 minute
rabbitmq_cpu_core_load	Average Load per CPU Core	Average load of each CPU core of the RabbitMQ VM	> 0	RabbitMQ instance node	1 minute

Metric ID	Metric Name	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
rabbitmq_memory_usage	Memory Usage	Memory usage of the RabbitMQ VM Unit: %	0–100%	RabbitMQ instance node	1 minute
rabbitmq_disk_read_await	Average Disk Read Time	Average time for each disk I/O read in the monitoring period Unit: ms	> 0	RabbitMQ instance node	1 minute
rabbitmq_disk_write_await	Average Disk Write Time	Average time for each disk I/O write in the monitoring period Unit: ms	> 0	RabbitMQ instance node	1 minute
rabbitmq_node_bytes_in_rate	Inbound Traffic	Inbound traffic per second Unit: byte/s	> 0	RabbitMQ instance node	1 minute
rabbitmq_node_bytes_out_rate	Outbound Traffic	Outbound traffic per second Unit: byte/s	> 0	RabbitMQ instance node	1 minute
rabbitmq_node_queues	Queues	Number of queues in the node Unit: Count	> 0	RabbitMQ instance node	1 minute
rabbitmq_memory_high_watermark	Memory High Watermark	Whether the node has reached the memory high watermark, blocking all producers in the cluster	1: yes 0: no	RabbitMQ instance node	1 minute
rabbitmq_disk_insufficient	Disk High Watermark	Whether the node has reached the disk high watermark, blocking all producers in the cluster	1: yes 0: no	RabbitMQ instance node	1 minute

Queue Metrics

Table 10-3 Queue metrics

Metric ID	Metric Name	Description	Value Range	Monitored Object	Monitoring Period (Raw Data)
queue_messages_unacknowledged	Unacked Messages	Number of messages that have been retrieved but not acknowledged in the RabbitMQ queue Unit: count	0–10,000,000	RabbitMQ instance queue	1 minute
queue_messages_ready	Available Messages	Number of messages that can be retrieved in the RabbitMQ queue Unit: count	0–10,000,000	RabbitMQ instance queue	1 minute

Dimensions

Key	Value
rabbitmq_instance_id	RabbitMQ instance
rabbitmq_node	RabbitMQ instance node
rabbitmq_queue	RabbitMQ instance queue

10.2 Setting RabbitMQ Alarm Rules

This section describes the alarm rules of some metrics and how to configure the rules. In actual scenarios, you are advised to configure alarm rules for metrics by referring to the following alarm policies.

Table 10-4 Alarm rules for RabbitMQ instances

Metric	Alarm Policy	Description	Solution
Memory High Watermark	Alarm threshold: Raw data ≥ 1 Number of consecutive periods: 1 Alarm severity: Critical	A threshold of 1 indicates that the memory high watermark is reached, blocking message publishing.	• Accelerate message retrieval. • Use publisher confirms and monitor the publishing rate and duration on the publishing end. When the duration increases significantly, apply flow control.
Disk High Watermark	Alarm threshold: Raw data ≥ 1 Number of consecutive periods: 1 Alarm severity: Critical	A threshold of 1 indicates that the disk high watermark is reached, blocking message publishing.	• Reduce the number of messages accumulated in lazy queues. • Reduce the number of messages accumulated in durable queues. • Delete queues.
Memory Usage	Alarm threshold: Raw data > Expected usage (30% is recommended) Number of consecutive periods: 3–5 Alarm severity: Major	To prevent high memory watermarks from blocking publishing, configure an alarm for this metric on each node.	• Accelerate message retrieval. • Use publisher confirms and monitor the publishing rate and duration on the publishing end. When the duration increases significantly, apply flow control.
CPU Usage	Alarm threshold: Raw data > Expected usage (70% is recommended) Number of consecutive periods: 3–5 Alarm severity: Major	A high CPU usage may slow down publishing rate. Configure an alarm for this metric on each node.	• Reduce the number of mirrored queues. • For a cluster instance, add nodes and rebalance queues between all nodes.

Metric	Alarm Policy	Description	Solution
Available Messages	Alarm threshold: Raw data > Expected number of available messages Number of consecutive periods: 1 Alarm severity: Major	If the number of available messages is too large, messages are accumulated.	See **the solution to preventing message accumulation**.
Unacked Messages	Alarm threshold: Raw data > Expected number of unacknowledged messages Number of consecutive periods: 1 Alarm severity: Major	If the number of unacknowledged messages is too large, messages may be accumulated.	• Check whether the consumer is abnormal. • Check whether the consumer logic is time-consuming.
Connections	Alarm threshold: Raw data > Expected number of connections Number of consecutive periods: 1 Alarm severity: Major	A sharp increase in the number of connections may be a warning of a traffic increase.	The services may be abnormal. Check whether other alarms exist.
Channels	Alarm threshold: Raw data > Expected number of channels Number of consecutive periods: 1 Alarm severity: Major	A sharp increase in the number of channels may be a warning of a traffic increase.	The services may be abnormal. Check whether other alarms exist.

Metric	Alarm Policy	Description	Solution
Erlang Processes	Alarm threshold: Raw data > Expected number of processes Number of consecutive periods: 1 Alarm severity: Major	A sharp increase in the number of processes may be a warning of a traffic increase.	The services may be abnormal. Check whether other alarms exist.

NOTE

- Set the alarm threshold based on the service expectations. For example, if the expected usage is 35%, set the alarm threshold to 35%.
- The number of consecutive periods and alarm severity can be adjusted based on the service logic.

Procedure

Step 1 Log in to the management console.

Step 2 In the upper left corner, click and select a region.

NOTE

Select the region where your RabbitMQ instance is.

Step 3 Click and choose **Application** > **Distributed Message Service for RabbitMQ** to open the console of DMS for RabbitMQ.

Step 4 View the instance metrics using either of the following methods:

- Click next to a RabbitMQ instance name. On the Cloud Eye console, view the metrics of the instance, nodes, and queues. Metric data is reported to Cloud Eye every minute.
- Click the desired RabbitMQ instance to view its details. In the navigation pane, choose **Monitoring** view. On the displayed page, view the metrics of the instance, nodes, and queues. Metric data is reported to Cloud Eye every minute.

Step 5 Hover the mouse pointer over a metric and click to create an alarm rule for the metric.

Step 6 Specify the alarm rule details.

For more information about creating alarm rules, see **Creating an Alarm Rule**.

1. Enter the alarm name and description.

2. Specify the alarm policy and alarm severity.

   For example, an alarm can be triggered and notifications can be sent once every day if the raw value of connections exceeds the preset value for three consecutive periods and no actions are taken to handle the exception.

3. Set **Alarm Notification** configurations. If you enable **Alarm Notification**, set the validity period, notification object, and trigger condition.
4. Click **Create**.

----End

10.3 Viewing Metrics

Scenario

Cloud Eye monitors DMS for RabbitMQ metrics in real time. You can view these metrics on the console.

Prerequisites

At least one RabbitMQ instance has been created. The instance has at least one available message.

Procedure

Step 1 Log in to the management console.

Step 2 In the upper left corner, click and select a region.

NOTE

Select the region where your RabbitMQ instance is.

Step 3 Click and choose **Application** > **Distributed Message Service for RabbitMQ** to open the console of DMS for RabbitMQ.

Step 4 View the instance metrics using either of the following methods:

- Click next to a RabbitMQ instance name. On the Cloud Eye console, view the metrics of the instance, nodes, and queues. Metric data is reported to Cloud Eye every minute.
- Click the desired RabbitMQ instance to view its details. In the navigation pane, choose **Monitoring** view. On the displayed page, view the metrics of the instance, nodes, and queues. Metric data is reported to Cloud Eye every minute.

On the **By Queue** tab page, the name of a queue is displayed if the queue is on the default virtual host. If a queue is not on the default virtual host, the queue name is displayed in the format "*Name of the virtual host where the queue is*_*Queue name*". For example, if the **test01** queue is on **Vhost-13142708**, the queue name displayed on the monitoring page is **Vhost-13142708_test01**.

----End

11 Auditing

11.1 Operations Logged by CTS

With Cloud Trace Service (CTS), you can record operations associated with DMS for RabbitMQ for later query, audit, and backtrack operations.

Table 11-1 DMS for RabbitMQ operations that can be recorded by CTS

Operation	Resource Type	Trace Name
Successfully deleting a background task	rabbitmq	deleteDMSBackendJobSuccess
Failing to delete a background task	rabbitmq	deleteDMSBackendJobFailure
Successfully creating an order for creating an instance	rabbitmq	createDMSInstanceOrderSuccess
Failing to create an order for creating an instance	rabbitmq	createDMSInstanceOrderFailure
Successfully creating an order for modifying an instance	rabbitmq	modifyDMSInstanceOrderSuccess
Failing to create an order for modifying an instance	rabbitmq	modifyDMSInstanceOrderFailure
Successfully scaling up an instance	rabbitmq	extendDMSInstanceSuccess
Failing to scale up an instance	rabbitmq	extendDMSInstanceFailure

Operation	Resource Type	Trace Name
Successfully resetting instance password	rabbitmq	resetDMSInstancePasswordSuccess
Failing to reset instance password	rabbitmq	resetDMSInstancePasswordFailure
Successfully deleting an instance that failed to be created	rabbitmq	deleteDMSCreateFailureInstancesSuccess
Failing to delete an instance that failed to be created	rabbitmq	deleteDMSCreateFailureInstancesFailure
Successfully restarting an instance	rabbitmq	restartDMSInstanceSuccess
Failing to restart an instance	rabbitmq	restartDMSInstanceFailure
Successfully deleting multiple instances at a time	rabbitmq	batchDeleteDMSInstanceSuccess
Failing to delete multiple instances at a time	rabbitmq	batchDeleteDMSInstanceFailure
Successfully restarting multiple instances at a time	rabbitmq	batchRestartDMSInstanceSuccess
Failing to restart multiple instances at a time	rabbitmq	batchRestartDMSInstanceFailure
Successfully modifying instance information	rabbitmq	modifyDMSInstanceInfoSuccess
Failing to modify instance information	rabbitmq	modifyDMSInstanceInfoFailure
Successfully deleting multiple instance tasks at a time	rabbitmq	batchDeleteDMSInstanceTask
Successfully unfreezing an instance	rabbitmq	unfreezeDMSInstanceTaskSuccess
Failing to unfreeze an instance	rabbitmq	unfreezeDMSInstanceTaskFailure
Successfully freezing an instance	rabbitmq	freezeDMSInstanceTaskSuccess

Operation	Resource Type	Trace Name
Failing to freeze an instance	rabbitmq	freezeDMSInstanceTaskFailure
Successfully deleting an instance task	rabbitmq	deleteDMSInstanceTaskSuccess
Failing to delete an instance task	rabbitmq	deleteDMSInstanceTaskFailure
Successfully creating an instance task	rabbitmq	createDMSInstanceTaskSuccess
Failing to create an instance task	rabbitmq	createDMSInstanceTaskFailure
Successfully submitting a request for scaling up an instance	rabbitmq	extendDMSInstanceTaskSuccess
Failing to submit a request for scaling up an instance	rabbitmq	extendDMSInstanceTaskFailure
Successfully submitting a request for restarting an instance	rabbitmq	restartDMSInstanceTaskSuccess
Failing to submit a request for restarting an instance	rabbitmq	restartDMSInstanceTaskFailure
Successfully submitting a request for restarting multiple instances at a time	rabbitmq	batchRestartDMSInstanceTask-Success
Failing to submit a request for restarting multiple instances at a time	rabbitmq	batchRestartDMSInstanceTask-Failure
Successfully submitting a request for modifying instance information	rabbitmq	modifyDMSInstanceInfoTaskSuc-cess
Failing to submit a request for modifying instance information	rabbitmq	modifyDMSInstanceInfoTaskFai-lure

11.2 Viewing Audit Logs

See **Querying Real-Time Traces**.

