Distributed Message Service for RabbitMQ

Developer Guide

Issue 01

Date 2023-11-15

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Contents

1 Overview

This guide describes how to collect the information required for connecting to a RabbitMQ instance, such as the instance connection address, port, username, and password. It also provides connection examples in Python, and Spring Boot.

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

Open Source SDKs

DMS for RabbitMQ supports all SDK open-source versions, **Table 1-1** lists common ones.

Table 1-1 Open source SDKs

Language	SDK
Java	**rabbitmq-java-client**
Spring Framework	**SpringAMQP**
.Net	**rabbitmq-dotnet-client**
Python	**pika**
PHP	**php-amqplib**
C	**rabbitmq-c**
Go	**amqp091-go**

NOTE

The latest release of the SDKs is recommended.

Client Network Environment

A client can access a RabbitMQ instance in any of the following modes:

1. Within a Virtual Private Network (VPC)

   If the client runs on an Elastic Cloud Server (ECS) and is in the same region and VPC as the RabbitMQ instance, the client can access the instance using an IP address within a subnet in the VPC.

2. Using a VPC peering connection

   If the client runs on an ECS and is in the same region but not the same VPC as the RabbitMQ instance, the client can access the instance using an IP address within a subnet in the VPC of the RabbitMQ instance after a VPC peering connection has been established.

   For details, see **VPC Peering Connection**.

3. Over public networks

   If the client is not in the same network environment or region as the RabbitMQ instance, the client can access the instance using its public network IP address.

NOTE

The three modes differ only in the connection address used by the client to access the instance. This document takes intra-VPC access as an example to describe how to set up the development environment.

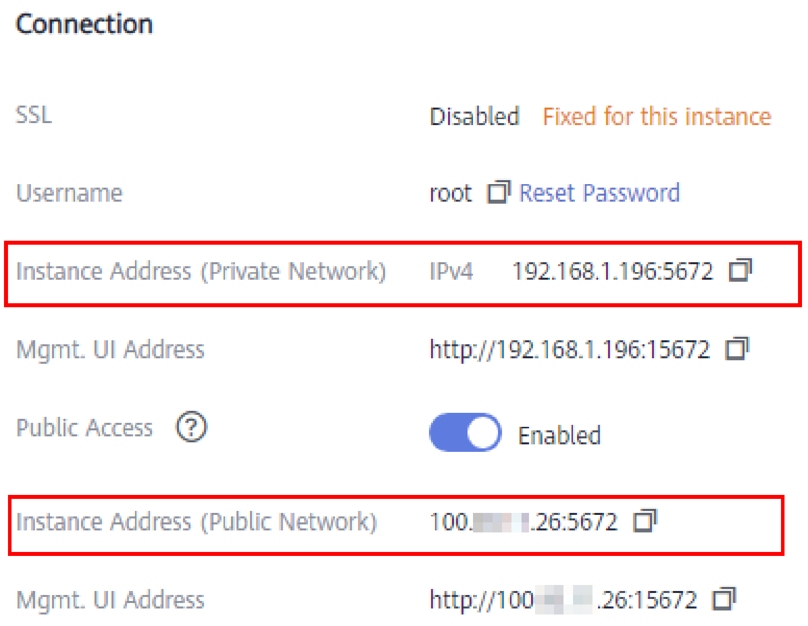
If the connection times out or fails, check the network connectivity. You can use Telnet to check connectivity of the connection address and port number of the instance.

2 Collecting Connection Information

- Instance connection address and port

  After an instance is created, obtain its connection address from the **Connection** area on the **Basic Information** page.

  Figure 2-1 Viewing the connection address and port of a RabbitMQ instance

  Connection

SSL	Disabled Fixed for this instance
Username	root Reset Password
Instance Address (Private Network)	IPv4 192.168.1.196:5672
Mgmt. UI Address	http://192.168.1.196:15672
Public Access	Enabled
Instance Address (Public Network)	100.[redacted].26:5672
Mgmt. UI Address	http://100[redacted].26:15672

- Username and password for accessing an instance

  After an instance is created, obtain the username for connecting to it from the **Connection** area on the **Basic Information** page. If you have forgotten the password, click **Reset Password**.

3 Configuring Clients in Python

This section describes how to access a RabbitMQ instance using a RabbitMQ client in Python on the Linux CentOS, including how to install the client, and produce and consume messages.

Before getting started, ensure that you have collected the information described in **Collecting Connection Information**.

Preparing the Environment

- Python

  Generally, Python is pre-installed in the system. Enter **python** in a CLI. If the following information is displayed, Python has already been installed.

  ```
  [root@ecs-test python]# python3
  Python 3.7.1 (default, Jul  5 2020, 14:37:24)
  [GCC 4.8.5 20150623 (Red Hat 4.8.5-39)] on linux
  Type "help", "copyright", "credits" or "license" for more information.
  >>>
  ```

  If Python is not installed, run the following command to install it:

  yum install python

- A RabbitMQ client in Python. In this document, pika is used as an example.

  Run the following command to install the recommended version of pika:

  pip install pika

  If pika cannot be installed using the **pip** command, run the following **pip3** command instead:

  pip3 install pika

Producing Messages

NOTE

Replace the information in bold with the actual values.

- SSL authentication

  ```
  import pika
  import ssl

  #Connection information
  conf = {
      'host': 'ip',
  ```

```
    'port': 5671,
    'queue_name': 'queue-test',
    'username': 'root',
    'password': 'password'
}

context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2)
credentials = pika.PlainCredentials(conf['username'], conf['password'])
parameters = pika.ConnectionParameters(conf['host'],
                                       conf['port'],
                                       '/',
                                       credentials,
                                       ssl_options=pika.SSLOptions(context))

connection = pika.BlockingConnection(parameters)
channel = connection.channel()
channel.queue_declare(conf['queue_name'])
data = bytes('Hello World!', encoding='utf-8')
channel.basic_publish(exchange='',
                      routing_key=conf['queue_name'],
                      body=data)

print(" [x] Sent 'Hello World!'")

connection.close()
```

- Non-SSL authentication

```
import pika

#Connection information
conf = {
    'host': 'ip',
    'port': 5672,
    'queue_name': 'queue-test',
    'username': 'root',
    'password': 'password'
}

credentials = pika.PlainCredentials(conf['username'], conf['password'])
parameters = pika.ConnectionParameters(conf['host'],
                                       conf['port'],
                                       '/',
                                       credentials)

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

channel.queue_declare(conf['queue_name'])

data = bytes("Hello World!", encoding="utf-8")

channel.basic_publish(exchange='',
                      routing_key=conf['queue_name'],
                      body=data)

print(" [x] Sent 'Hello World!'")

connection.close()
```

Consuming Messages

NOTE

Replace the information in bold with the actual values.

- SSL authentication

```
import pika
import ssl
```

```
#Connection information
conf = {
    'host': 'ip',
    'port': 5671,
    'queue_name': 'queue-test',
    'username': 'root',
    'password': 'password'
}

context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2)
credentials = pika.PlainCredentials(conf['username'], conf['password'])
parameters = pika.ConnectionParameters(conf['host'],
                                       conf['port'],
                                       '/',
                                       credentials,
                                       ssl_options=pika.SSLOptions(context))

connection = pika.BlockingConnection(parameters)
channel = connection.channel()
channel.queue_declare(conf['queue_name'])


def callback(ch, method, properties, body):
    print(" [x] Received %r" % body.decode('utf-8'))


channel.basic_consume(queue=conf['queue_name'], on_message_callback=callback, auto_ack=True)

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

- Non-SSL authentication

```
import pika

#Connection information
conf = {
    'host': 'ip',
    'port': 5672,
    'queue_name': 'queue-test',
    'username': 'root',
    'password': 'password'
}

credentials = pika.PlainCredentials(conf['username'], conf['password'])
parameters = pika.ConnectionParameters(conf['host'],
                                       conf['port'],
                                       '/',
                                       credentials)

connection = pika.BlockingConnection(parameters)
channel = connection.channel()
channel.queue_declare(conf['queue_name'])


def callback(ch, method, properties, body):
    print(" [x] Received %r" % body.decode('utf-8'))


channel.basic_consume(queue=conf['queue_name'], on_message_callback=callback, auto_ack=True)

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

4 Connecting to a RabbitMQ Instance with Spring Boot

This topic describes how to connect to a RabbitMQ instance with Spring Boot to produce and consume messages.

Before getting started, ensure that you have collected the information described in **Collecting Connection Information**.

Add the spring-boot-starter-amqp Dependency to pom.xml

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-amqp</artifactId>
  <version>2.3.1.RELEASE</version>
</dependency>
```

(Optional) Writing Configuration in application.properties

If SSL is enabled for the RabbitMQ instance, write the following configuration in **application.properties**.

```
#Enable SSL
spring.rabbitmq.ssl.enabled=true
#SSL algorithm
spring.rabbitmq.ssl.algorithm=TLSv1.2
#Whether to verify the host
spring.rabbitmq.ssl.verify-hostname=false
#Whether to validate the server certificate
spring.rabbitmq.ssl.validate-server-certificate=false
```

Producing Messages

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class ProducerController {
    private static final Logger LOG = LoggerFactory.getLogger(ProducerController.class);

    @Autowired
    private RabbitTemplate;

    @GetMapping(value = "/send")
    public boolean send(String msg, Long delayTime) {
        rabbitTemplate.convertAndSend("ex-sour", "abc", msg,
            message -> {
                /**
                 * Set delay time
                 */
                message.getMessageProperties().setHeader("x-delay", delayTime);
                return message;
            });
        LOG.info("send delayed message: {}, delay: {}ms", msg, delayTime);
        return true;
    }
}
```

Consuming Messages

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.amqp.rabbit.annotation.RabbitListener;
import org.springframework.stereotype.Component;
import java.text.SimpleDateFormat;
import java.util.Date;


@Component
public class ReceiveMsgService {
    Logger LOG =  LoggerFactory.getLogger(ReceiveMsgService.class);

    @RabbitListener(queues = "test")
    public void receive(String message) {
        SimpleDateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        LOG.info("receive message: {}", message + " time: " + simpleDateFormat.format(new Date()));
    }
}
```