Distributed Message Service for RabbitMQ

Developer Guide

Issue 01

Date 2021-10-12

HUAWEI

HUAWEI TECHNOLOGIES CO., LTD.

Trademarks and Permissions


and other Huawei trademarks are trademarks 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 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 Overview

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

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**.

Network Environment of the Client

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.

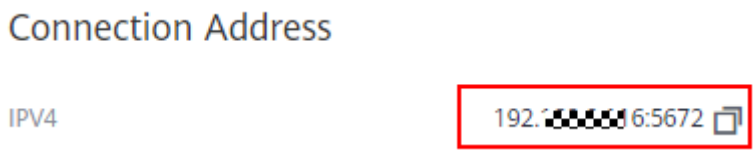
If the connection times out or fails, check the network connectivity. You can use Telnet to check the connection address and port number of the instance.

Obtaining Instance Connection Information

- Instance connection address and port

  After an instance is created, obtain its connection address from the **Connection Address** area on the **Basic Information** page. If public access is enabled, obtain the address from the **Public Access** area on the **Basic Information** page.

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

- Username and password for accessing an instance

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

2 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 **Obtaining Instance 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

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()
```

