

# **TDMQ for RabbitMQ**

## **SDK Documentation**

### **Product Documentation**



## Copyright Notice

©2013-2024 Tencent Cloud. All rights reserved.

Copyright in this document is exclusively owned by Tencent Cloud. You must not reproduce, modify, copy or distribute in any way, in whole or in part, the contents of this document without Tencent Cloud's the prior written consent.

## Trademark Notice



All trademarks associated with Tencent Cloud and its services are owned by Tencent Cloud Computing (Beijing) Company Limited and its affiliated companies. Trademarks of third parties referred to in this document are owned by their respective proprietors.

## Service Statement

This document is intended to provide users with general information about Tencent Cloud's products and services only and does not form part of Tencent Cloud's terms and conditions. Tencent Cloud's products or services are subject to change. Specific products and services and the standards applicable to them are exclusively provided for in Tencent Cloud's applicable terms and conditions.

# Contents

## SDK Documentation

Spring Boot Starter

Spring Cloud Stream

SDK for Java

SDK for Go

SDK for Python

SDK for PHP

SDK for C++

# SDK Documentation

## Spring Boot Starter

Last updated : 2024-01-03 11:45:32

### Overview

This document describes how to use Spring Boot Starter SDK to send and receive messages and helps you better understand the message sending and receiving processes.

### Prerequisites

You have created the required resources as instructed in [Resource Creation and Preparation](#).

[You have installed JDK 1.8 or later.](#)

[You have installed Maven 2.5 or later.](#)

[You have downloaded the demo](#)

### Directions

#### Step 1. Add dependencies

Add dependencies to the `pom.xml` file.



```
<!--rabbitmq-->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-amqp</artifactId>
</dependency>
```

## Step 2. Prepare configurations

1. Add RabbitMQ configuration information to the configuration file (with the YAML configuration as an example).



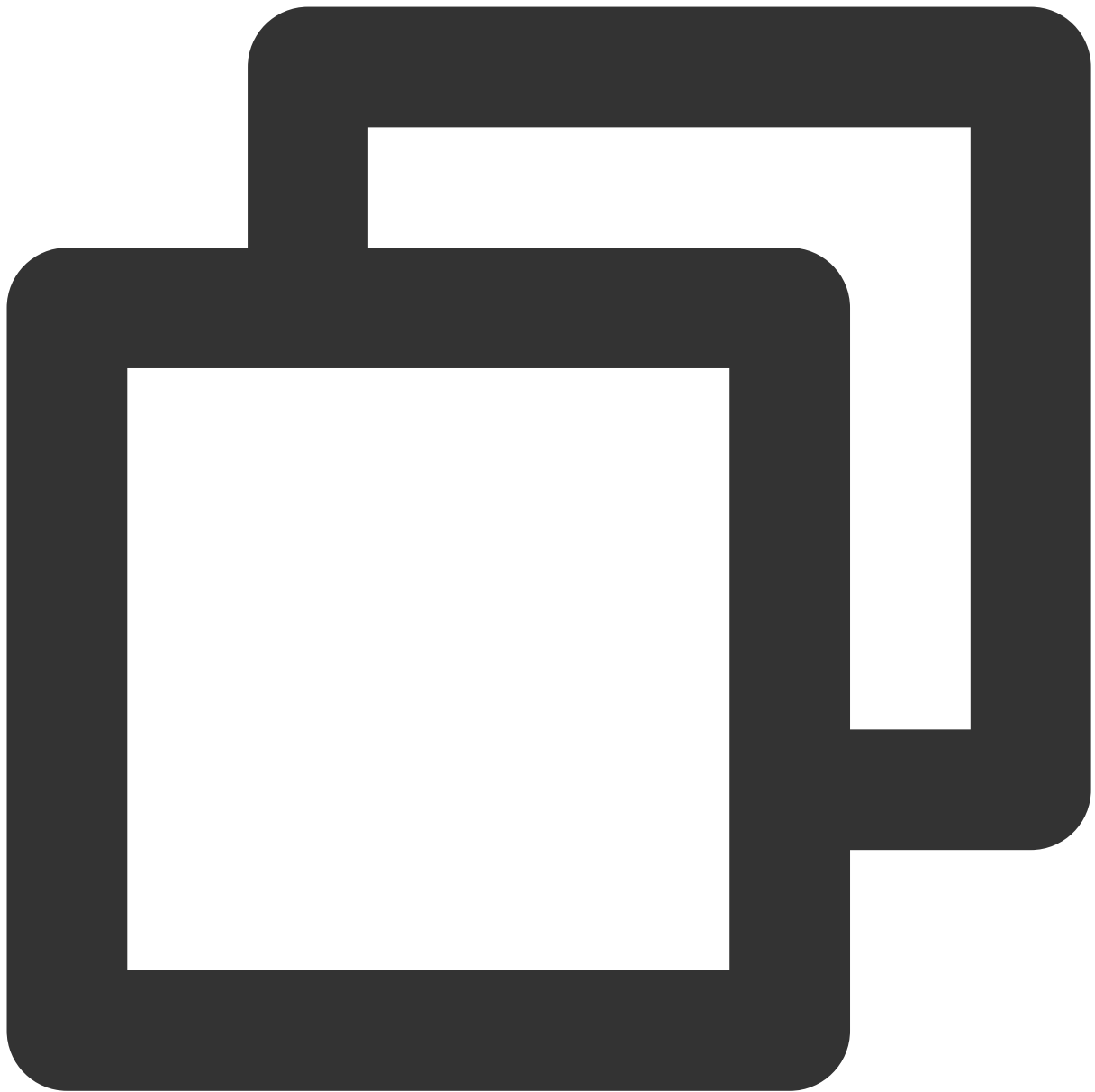
```
spring:
  rabbitmq:
    # The host address can be obtained in the console. You can also use the RabbitMQ host address
    host: amqp-xx.rabbitmq.x.tencenttdmq.com
    port: 5672
    # Name of the role to be used, which can be obtained on the **Role Management** page
    username: admin
    # Role token
    password: eyJrZXlJZ....
    # Full name of vhost, which can be obtained on the **Vhost** tab in the console
    virtual-host: amqp-xxx|Vhost
```

Parameter	Description										
host	<div>Cluster access address, which can be obtained from the <b>Client Access</b> section on the <b>Basic Info</b> page of the cluster.</div> <div><div><div><div>Client Access ⓘ</div><div>Add a routing policy</div></div><table><tr><th>Access Type</th><th>Access Policy</th><th>Public ...</th><th>Network</th><th>Operat...</th></tr><tr><td>VPC Network</td><td>-</td><td>-</td><td><div>vpc-fs6qq7yn ⓘ subnet-8ah6a7rs ⓘ amqp://10.0.2.4:5672 📄</div></td><td>Delete</td></tr></table></div></div>	Access Type	Access Policy	Public ...	Network	Operat...	VPC Network	-	-	<div>vpc-fs6qq7yn ⓘ subnet-8ah6a7rs ⓘ amqp://10.0.2.4:5672 📄</div>	Delete
Access Type	Access Policy	Public ...	Network	Operat...							
VPC Network	-	-	<div>vpc-fs6qq7yn ⓘ subnet-8ah6a7rs ⓘ amqp://10.0.2.4:5672 📄</div>	Delete							
port	Cluster access address, which can be obtained from the <b>Client Access</b> section on the <b>Basic Info</b> page of the cluster.										
username	Enter the name of the user created in the console.										
password	Enter the password of the user created in the console.										
virtual-host	Vhost name, which can be obtained from the vhost list in the console.										

2. Create a configuration file loading program (with the `Fanout exchange` as an example).

**Note**

For the configurations of exchanges in other types, see [Demo](#).



```
/**
 * Fanout exchange configuration
 */
@Configuration
public class FanoutRabbitConfig {

    /**
     * Exchange
     */
    @Bean
    public FanoutExchange fanoutExchange() {
```



```
        return new FanoutExchange("fanout-logs", true, false);
    }

    /**
     * Message queue
     */
    @Bean
    public Queue fanoutQueueA() {
        return new Queue("ps_queue", true);
    }

    @Bean
    public Queue fanoutQueueB() {
        // You can use this method to bind a dead letter queue
        Map<String, Object> requestParam = new HashMap<>();
        requestParam.put("x-dead-letter-exchange", "my-deadLetter-exchange");
        // Set the message validity period
        requestParam.put("x-message-ttl", 60000);
        return new Queue("ps_queue1", true, false, true, requestParam);
    }

    /**
     * Bind the message queue to the exchange
     */
    @Bean
    public Binding bindingFanoutA() {
        return BindingBuilder.bind(fanoutQueueA())
            .to(fanoutExchange());
    }

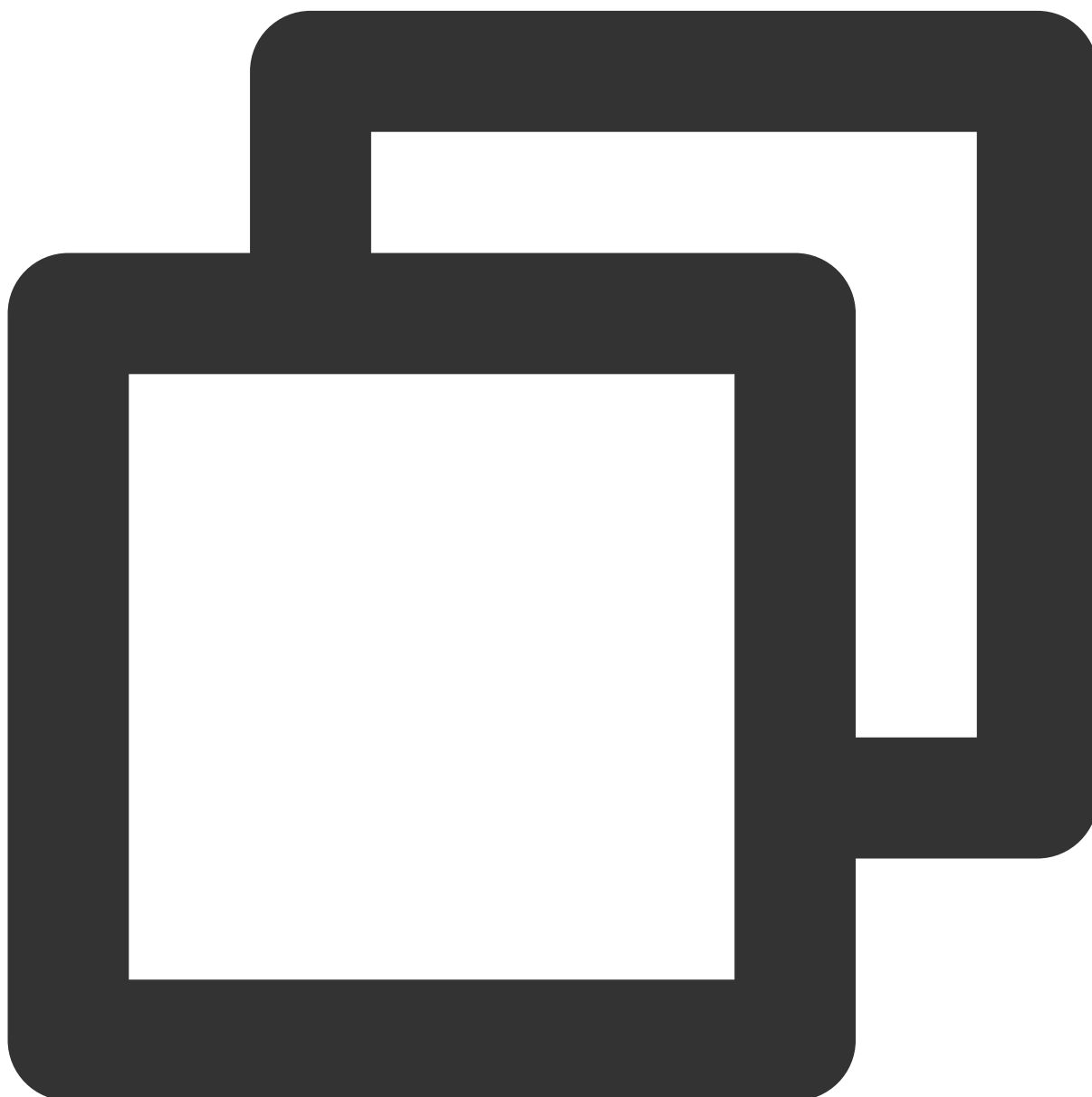
    @Bean
    public Binding bindingFanoutB() {
        return BindingBuilder.bind(fanoutQueueB())
            .to(fanoutExchange());
    }
}
```

Parameter	Description
fanout-logs	Bound exchange name, which can be obtained from the exchange list in the console.
ps_queue	Name of the first queue bound to the exchange, which can be obtained from the queue list in the console.
my-deadLetter-	Dead letter exchange name, which can be obtained from the exchange list in the

exchange	console.
ps_queue1	Name of the second queue bound to the exchange, which can be obtained from the queue list in the console.

### Step 3. Send messages

Create and compile the message sending program `DemoApplication.java` and use `RabbitTemplate` to send message.



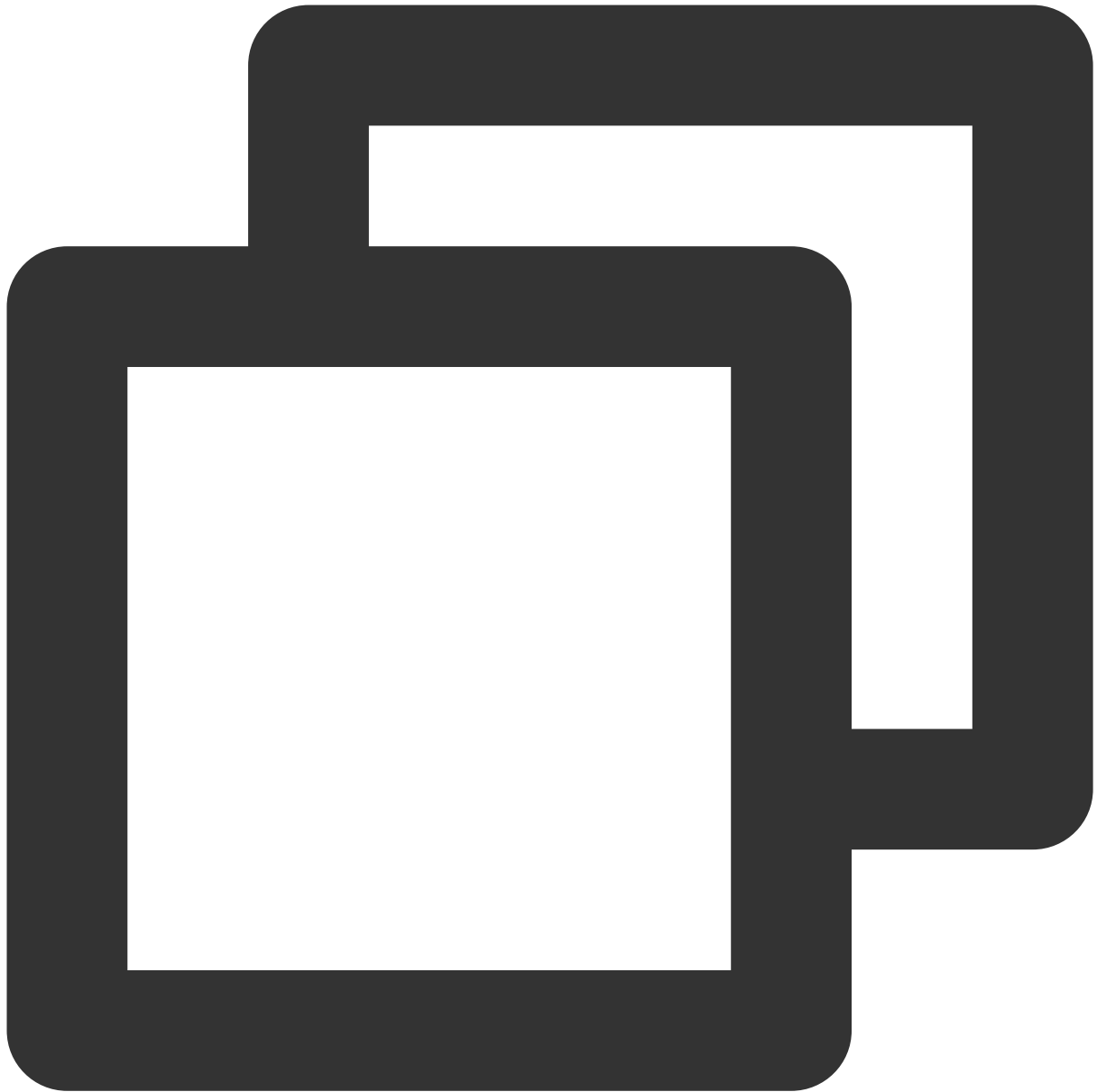
@Autowired

```
private RabbitTemplate rabbitTemplate;

public String send() {
    String msg = "This is a new message.";
    // Send the message
    // Parameter description: Parameter 1: Exchange name, which can be obtained from
    rabbitTemplate.convertAndSend("direct_logs", "", msg);
    return "success";
}
```

#### Step 4. Consume messages

Create and compile the message receiving program `FanoutReceiver.java` (with `Fanout` exchange as an example).



```
@Component
public class FanoutReceiver {
    // Register a listener to listen on the specified message queue
    @RabbitHandler
    @RabbitListener(queues = "ps_queue") // Name of the queue bound to the exchange,
    public void listenerPsQueue(String msg) {
        // Business processing...
        System.out.println("(ps_queue) receiver message. [" + msg + "]);
    }
}
```

## Step 5. View messages

If you want to confirm whether the messages have been successfully sent to TDMQ for RabbitMQ, you can view the status of connected consumers on the [Cluster](#)> Queue page in the console.

The screenshot displays the Tencent Cloud console interface for the 'aliveness-test' queue. The breadcrumb navigation at the top reads: Cluster / amqp-25mpxb9x / (AMQP default vhost) / **aliveness-test**. The 'Basic Info' section provides details about the queue: Queue Type is 'Regular queue', Online Consumer count is '0', Policy is 'View Applied Policies', Node is 'rabbit@rabbitmq-broker-0.rabbitmq-broker-internal.amqp-25mpxb9x.svc.cluster.local', Message TTL is '-ms', Auto Expire is '-', and Overflow Behavior is '-'. The Creation Time is '2023-07-27 15:36:41'. Below this, the 'Consumer List' section features a search bar with the placeholder 'Enter a keyword'. A table with columns 'Client IP' and 'Consumer Tag' is shown, but it is empty with the message 'No data yet'. At the bottom, it indicates 'Total items: 0' and a pagination control set to '20 / page' with navigation buttons for '1 / 1 page'.

### Note

For other samples, see [Spring AMQP official documentation](#).

# Spring Cloud Stream

Last updated : 2024-01-03 11:45:32

## Overview

This document describes how to use open-source SDK to send and receive messages by using the SDK for Spring Cloud Stream as an example and helps you better understand the message sending and receiving processes.

## Prerequisites

You have created the required resources as instructed in [Resource Creation and Preparation](#).

[You have installed JDK 1.8 or later.](#)

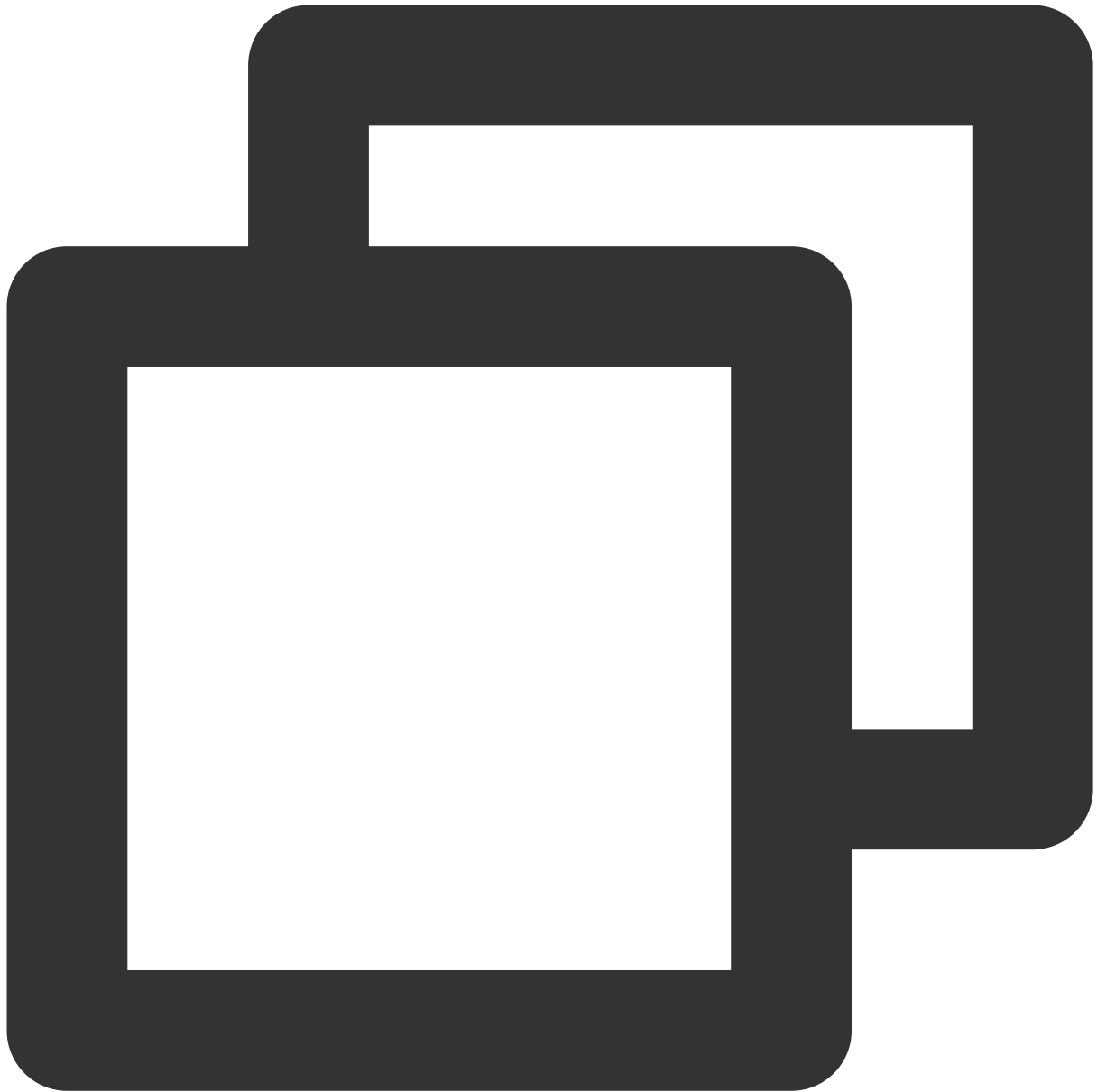
[You have installed Maven 2.5 or later.](#)

[You have downloaded the demo](#)

## Directions

### Step 1. Add dependencies

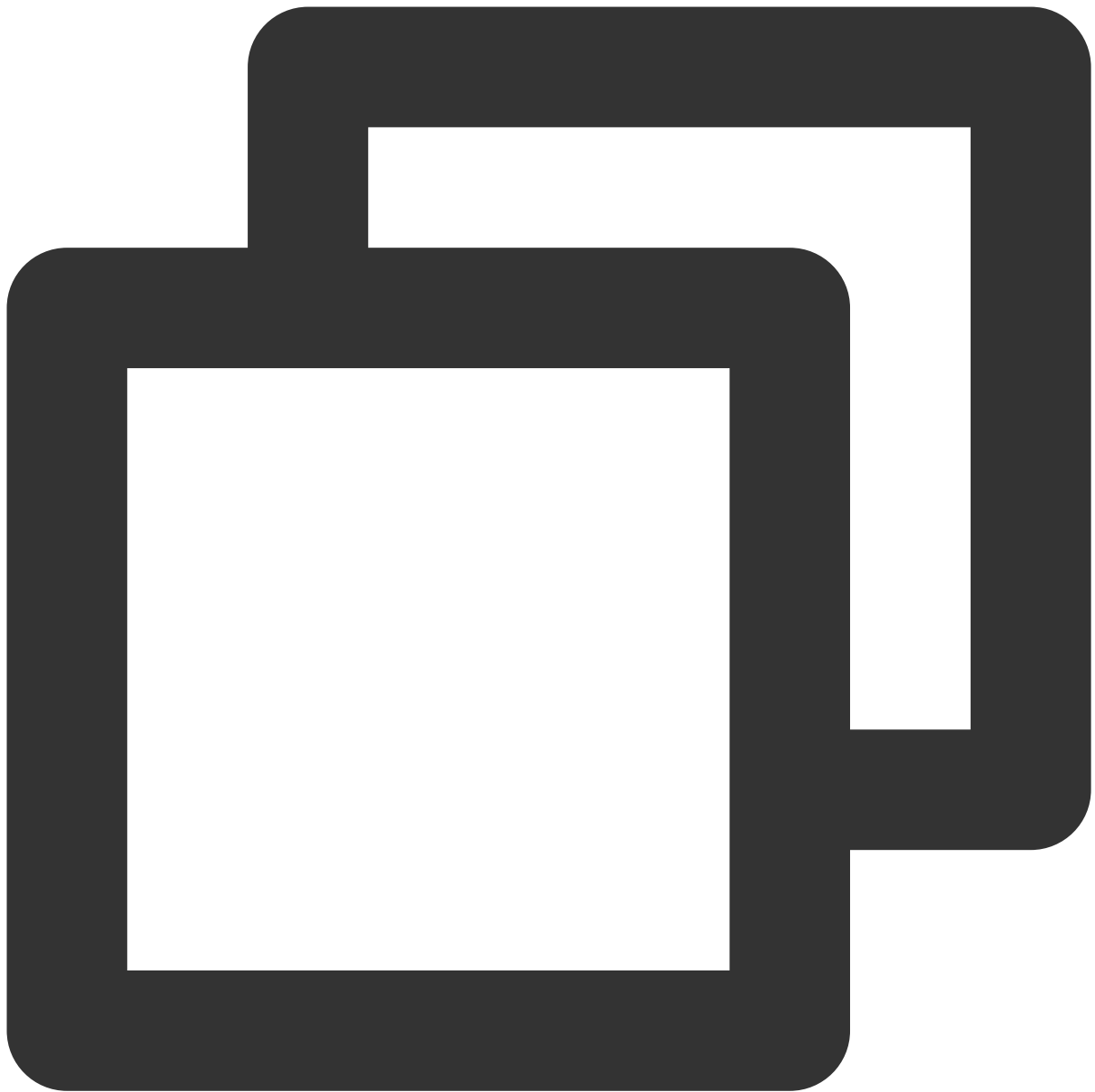
Add `Stream RabbitMQ` dependencies to `pom.xml` .



```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-stream-rabbit</artifactId>
</dependency>
```

## Step 2. Prepare configurations

1. Configure the configuration file (with the configuration of a `direct` exchange as an example).



```
spring:
  application:
    name: application-name
  cloud:
  stream:
    rabbit:
      bindings:
        # Output channel name
      output:
        # Producer configuration information
      producer:
```

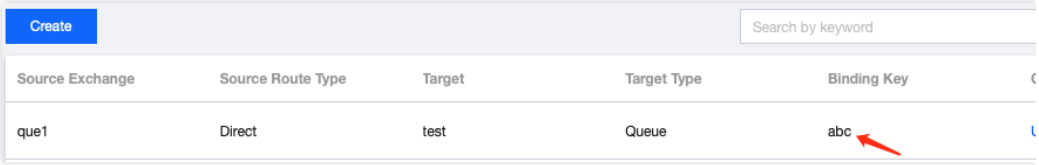
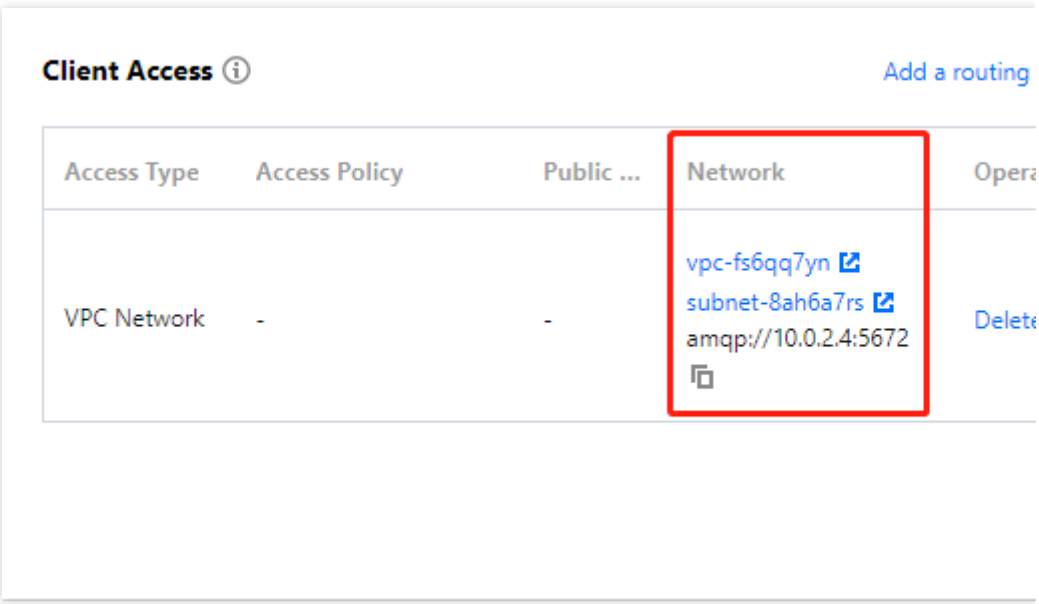


```

    # Type of the exchange used by the producer. If the exchange name already exists
    exchangeType: direct
    # It is used to specify a routing key expression
    routing-key-expression: headers["routeTo"] # This field indicates the routing key
    queueNameGroupOnly: true
# Input channel name
input:
  # Consumer configuration information
  consumer:
    # Type of the exchange used by the consumer. If the exchange name already exists
    exchangeType: direct
    # Routing keys bound to the consumer message queue
    bindingRoutingKey: info,waring,error
    # The configuration will process the above routing keys
    bindingRoutingKeyDelimiter: "," # This configuration item indicates the delimiter
    # Message acknowledgment mode. For more information, see `Acknowledge`
    acknowledge-mode: manual
    queueNameGroupOnly: true
bindings:
  # Output channel name
  output: # Channel name
    destination: direct_logs # Name of the exchange to be used
    content-type: application/json
    default-binder: dev-rabbit
  # Input channel name
  input: # Channel name
    destination: direct_logs # Name of the exchange to be used
    content-type: application/json
    default-binder: dev-rabbit
    group: route_queue1 # Name of the message queue to be used
binders:
  dev-rabbit:
    type: rabbit
    environment:
      spring:
        rabbitmq:
          host: amqp-xxx.rabbitmq.xxx.tencenttdmq.com #Cluster access address
          port: 5672
          username: admin # Role name
          password: password # Role token
          virtual-host: vhostname # Vhost name

```

Parameter	Description
bindingRoutingKey	Routing keys bound to the consumer message queue. Message routing rule can be obtained in the binding list in the console.

	
direct_log	Exchange name, which can be obtained from the exchange list in the console.
route_queue1	Queue name, which can be obtained from the queue list in the console.
host	<p>Cluster access address, which can be obtained from the <b>Client Access</b> section on the <b>Basi</b></p> 
port	Cluster access address port, which can be obtained in the <b>Operation</b> column on the <b>Cluste</b>
username	Enter the name of the user created in the console.
password	Enter the password of the user created in the console.
virtual-host	Vhost name, which can be obtained from the vhost list in the console.

2. Create a configuration file loading program.

OutputMessageBinding.java



```
public interface OutputMessageBinding {  
    /**  
     * Name of the channel to be used (output channel name)  
     */  
    String OUTPUT = "output";  
  
    @Output(OUTPUT)  
    MessageChannel output();  
}
```

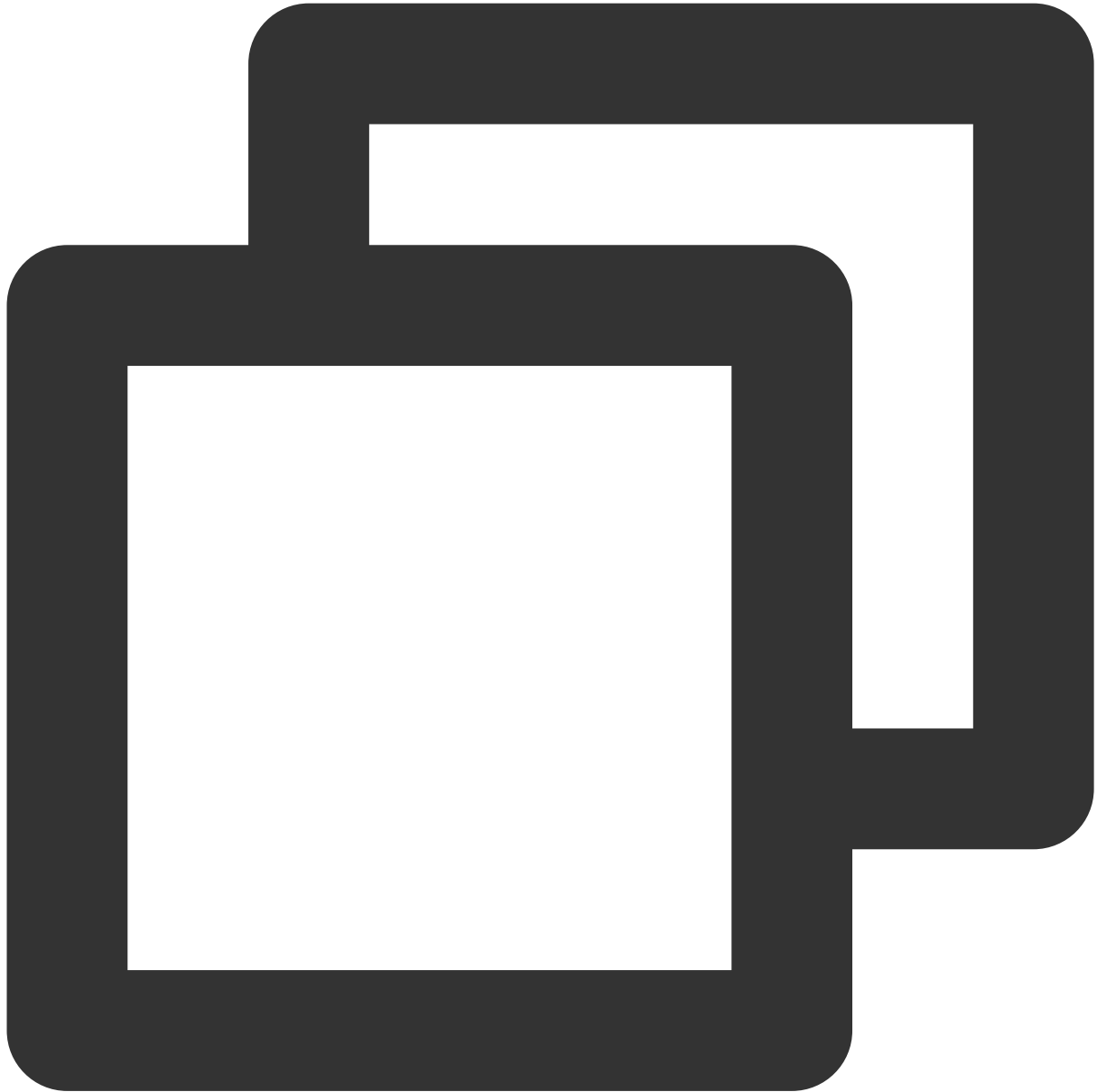
InputMessageBinding.java



```
public interface InputMessageBinding {  
  
    /**  
     * Name of the channel to be used  
     */  
    String INPUT = "input";  
  
    @Input(INPUT)  
    SubscribableChannel input();  
}
```

### Step 3. Send messages

Create and compile the message sending program `IMessageSendProvider.java` .



```
// Import the configuration class
@EnableBinding(OutputMessageBinding.class)
public class MessageSendProvider {

    @Autowired
    private OutputMessageBinding outputMessageBinding;

    public String sendToDirect() {
```

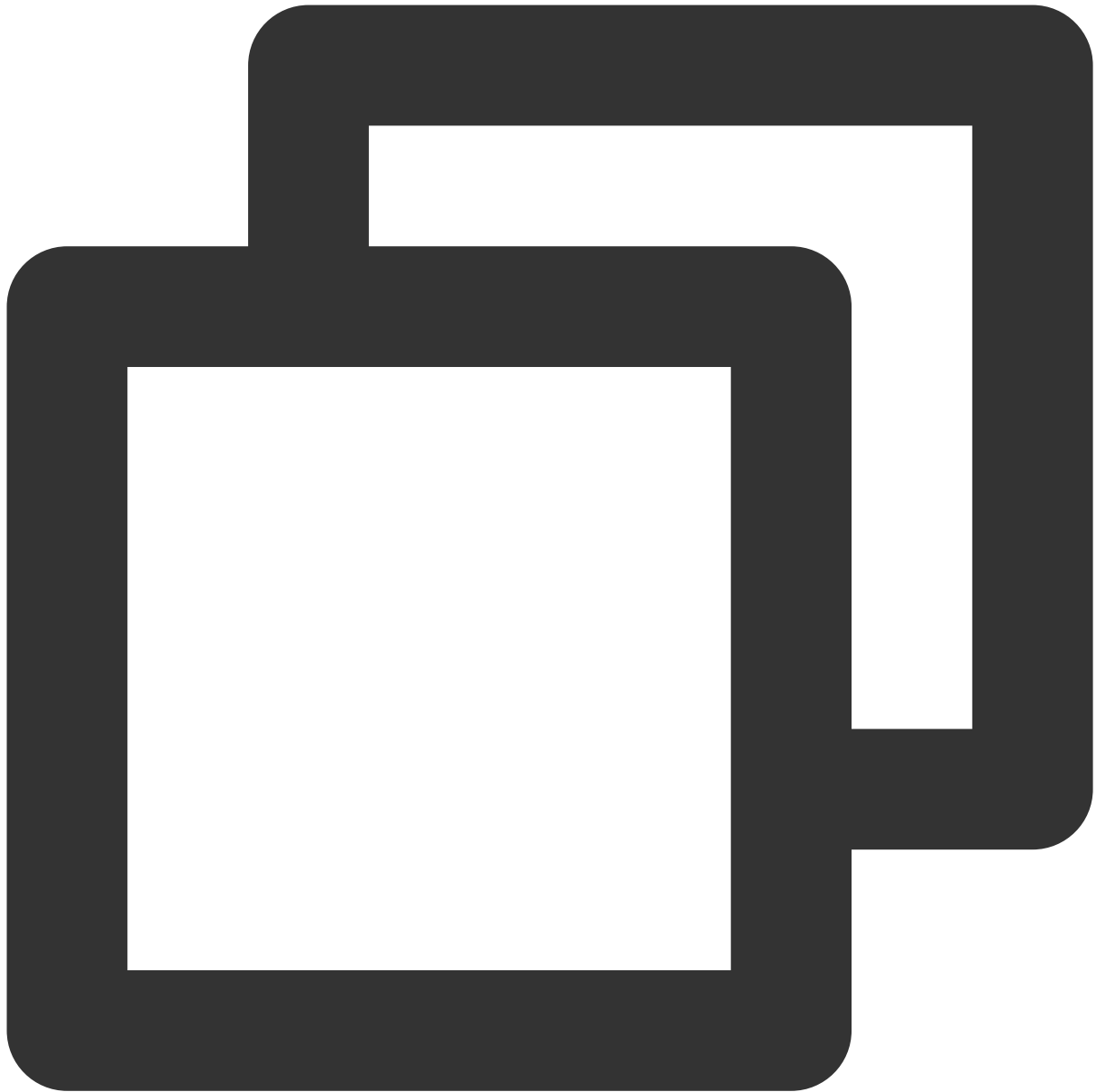
```
        outputMessageBinding.output().send(MessageBuilder.withPayload("[info] This is info message"));
        outputMessageBinding.output().send(MessageBuilder.withPayload("[warning] This is warning message"));
        outputMessageBinding.output().send(MessageBuilder.withPayload("[error] This is error message"));
        return "success";
    }

    public String sendToFanout() {
        for (int i = 0; i < 3; i++) {
            outputMessageBinding.output().send(MessageBuilder.withPayload("This is fanout message " + i));
        }
        return "success";
    }
}
```

Inject `MessageSendProvider` to the message sending class to send messages.

## Step 4. Consume messages

Create and compile the message consuming program `MessageConsumer.java`. **You can configure multiple channels to listen on different message queues.**



```
@Service
@EnableBinding(InputMessageBinding.class)
public class MessageConsumer {

    @StreamListener(InputMessageBinding.INPUT)
    public void test(Message<String> message) throws IOException {
        Channel channel = (com.rabbitmq.client.Channel) message.getHeaders().get(AmqpHeaders.CHANNEL);
        Long deliveryTag = (Long) message.getHeaders().get(AmqpHeaders.DELIVERY_TAG);
        channel.basicAck(deliveryTag, false);
        String payload = message.getPayload();
        System.out.println(payload);
    }
}
```

```
}  
}
```

## Step 5. View messages

If you want to confirm whether the messages have been successfully sent to TDMQ for RabbitMQ, you can view the status of connected consumers on the [Cluster](#) > Queue page in the console.

[←](#) Cluster / amqp-25mpxb9x / (AMQP default vhost) / **aliveness-test**

### Basic Info

Queue Type	Regular queue	Online Consumer	0	Policy
Node	<a href="#">rabbit@rabbitmq-broker-0.rabbitmq-broker-internal.amqp-25mpxb9x.svc.cluster.local</a>	Message TTL	-ms	Auto Expire
		Overflow Behavior	-	<a href="#">More Advanced Options &gt;</a>
Creation Time	2023-07-27 15:36:41			

### Consumer List

Enter a keyword

Client IP	Consumer Tag
No data yet	

Total items: 020 / page

### Note

Above is a sample based on the pub/sub pattern of RabbitMQ, which can be configured as needed. For more information, see [Demo](#) or [Spring cloud stream official documentation](#).



# SDK for Java

Last updated : 2024-01-03 11:45:32

## Overview

This document describes how to use open-source SDK to send and receive messages using the SDK for Java as an example and helps you better understand the message sending and receiving processes.

## Prerequisites

You have created the required resources.

You have installed JDK 1.8 or later.

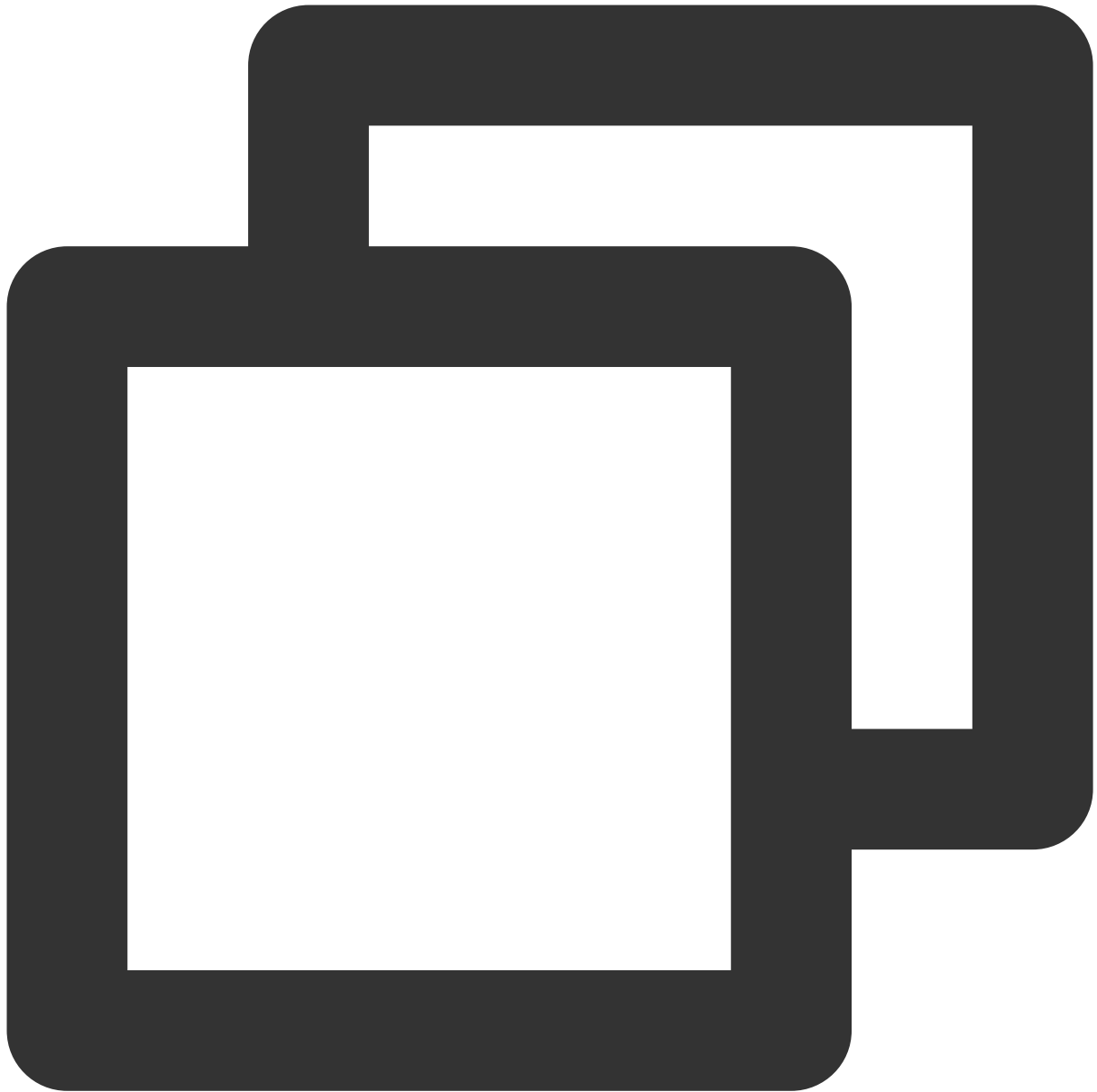
You have installed Maven 2.5 or later.

You have downloaded the demo

## Directions

### Step 1. Install the Java dependency library

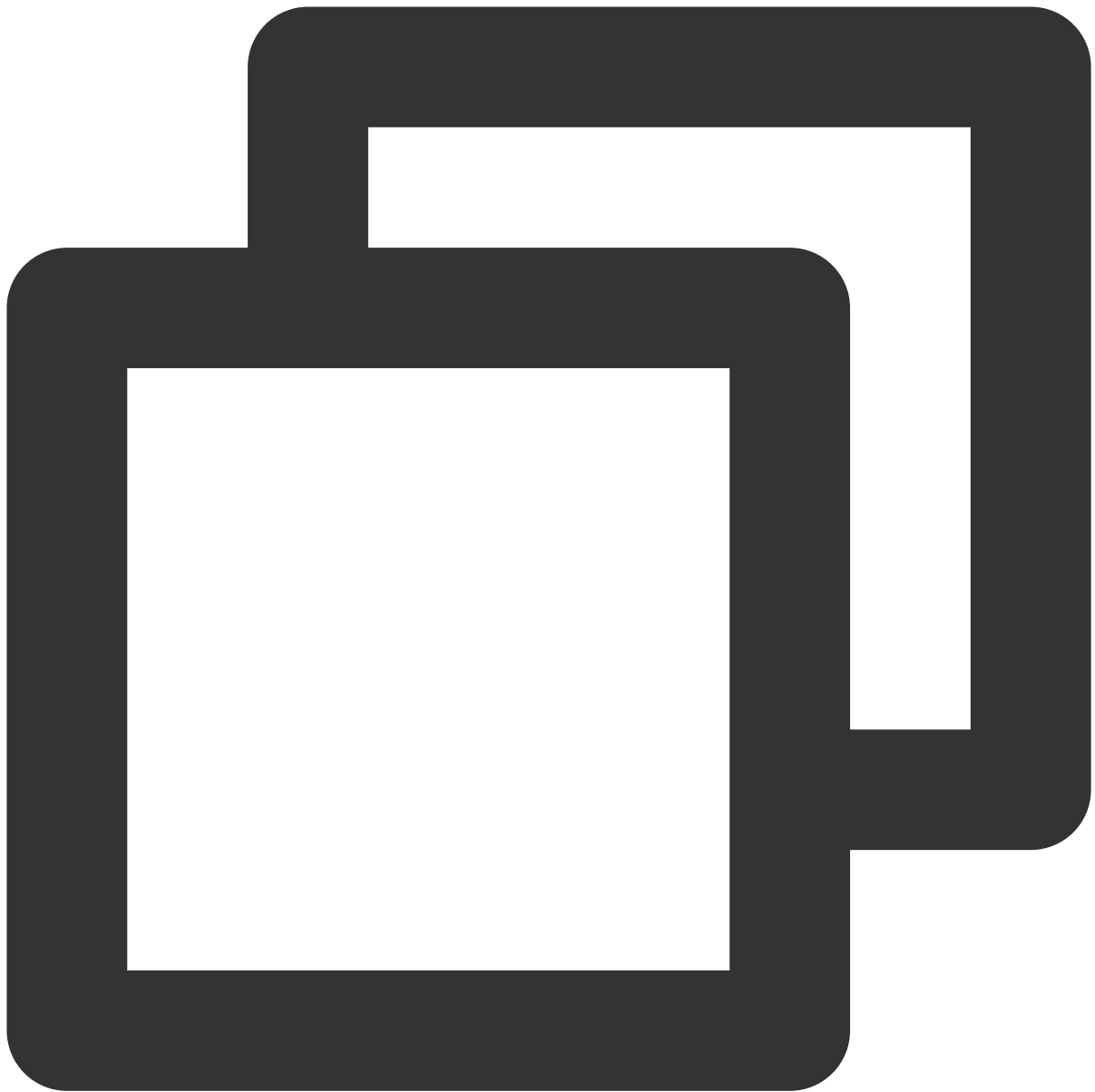
Add the following dependencies to the `pom.xml` file:



```
<!-- in your <dependencies> block -->
<dependency>
  <groupId>com.rabbitmq</groupId>
  <artifactId>amqp-client</artifactId>
  <version>5.13.0</version>
</dependency>
```

## Step 2. Produce messages

Create, compile, and run `MessageProducer.java` .



```
import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.ConnectionFactory;
import com.tencent.tdmq.demo.cloud.Constant;

/**
 * Message producer
 */
public class MessageProducer {

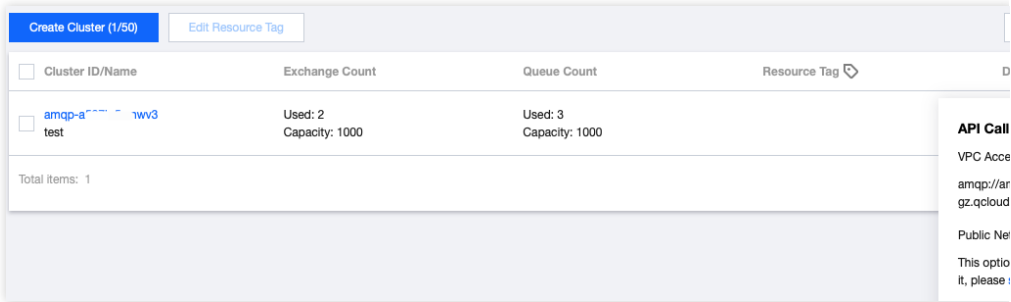
    /**
```

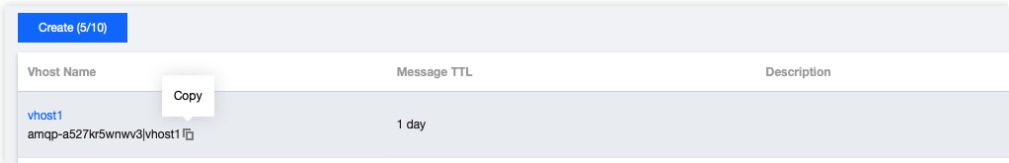
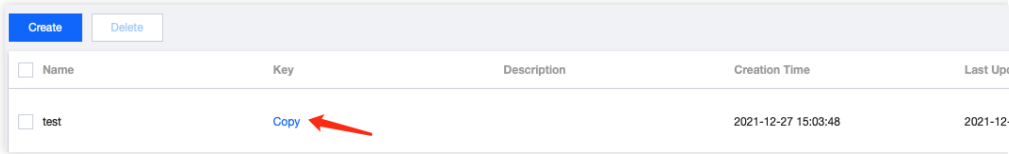
```

    * Exchange name
    */
private static final String EXCHANGE_NAME = "exchange_name";

public static void main(String[] args) throws Exception {
    // Connection factory
    ConnectionFactory factory = new ConnectionFactory();
    // Set the service address (replace with the access point address copied in
    factory.setUri("amqp://***");
    // Set vhost (replace with the vhost name copied in the console)
    factory.setVirtualHost(VHOST_NAME);
    // Set the username (use the role name in the permission configuration of t
    factory.setUsername(USERNAME);
    // Set the password (use the role key)
    factory.setPassword("eyJh****");
    // Get the connection address and establish the channel
    try (Connection connection = factory.newConnection(); Channel channel = con
        // Bind the message exchange (`EXCHANGE_NAME` must exist in the TDMQ fo
        channel.exchangeDeclare(EXCHANGE_NAME, "fanout");
        for (int i = 0; i < 10; i++) {
            String message = "this is rabbitmq message " + i;
            // Publish a message to the exchange, which will automatically deli
            channel.basicPublish(EXCHANGE_NAME, "", null, message.getBytes());
            System.out.println(" [producer] Sent '" + message + "'");
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
}

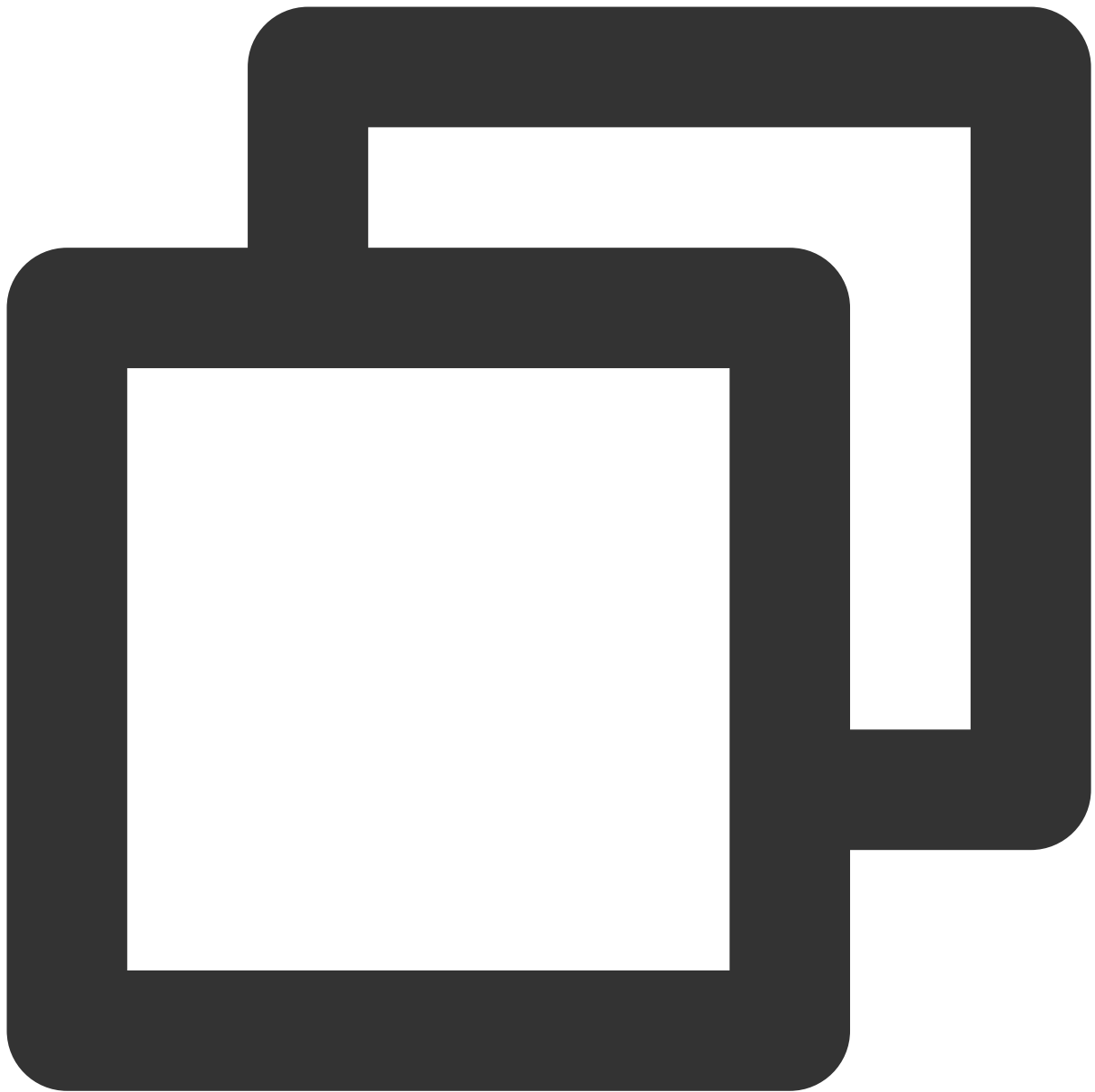
```

Parameter	Description
EXCHANGE_NAME	Exchange name, which can be obtained from the exchange list in the console.
factory.setUri	<p>Cluster access address, which can be obtained from <b>Access Address</b> in the <b>Operation</b></p> 

factory.setVirtualHost	<p>Vhost name in the format of "<b>cluster ID +   + vhost name</b>", which can be copied on the \</p> 
factory.setUsername	<p>Role name, which can be copied on the <a href="#">Role Management</a> page.</p>
factory.setPassword	<p>Role key, which can be copied in the <b>Key</b> column on the <a href="#">Role Management</a> page.</p> 

### Step 3. Consume messages

Create, compile, and run `MessageConsumer.java`.



```
import com.rabbitmq.client.AMQP;
import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.ConnectionFactory;
import com.rabbitmq.client.DefaultConsumer;
import com.rabbitmq.client.Envelope;
import com.tencent.tdmq.demo.cloud.Constant;

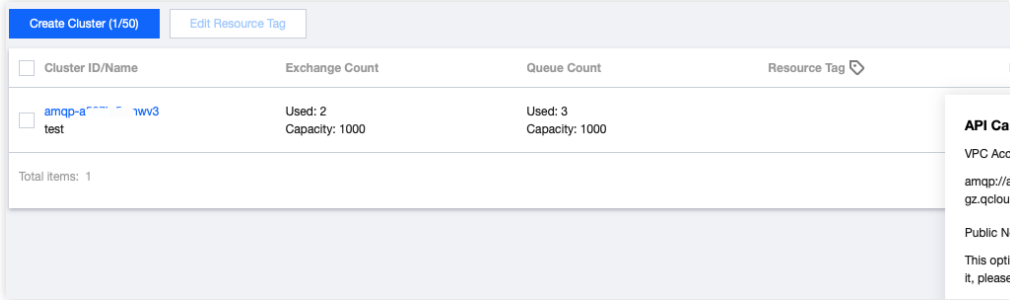
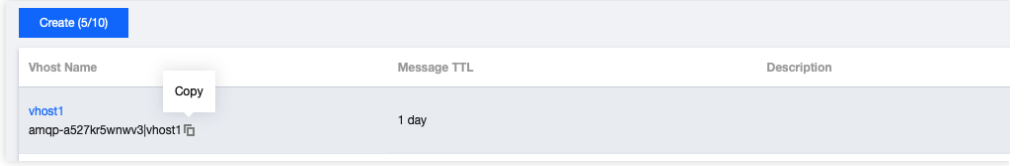
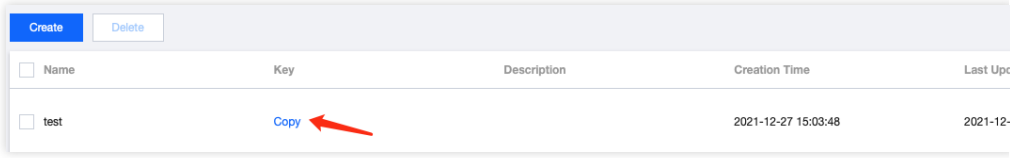
import java.io.IOException;
import java.nio.charset.StandardCharsets;
```

```
/**
 * Message consumer
 */
public class MessageConsumer1 {

    /**
     * Queue name
     */
    public static final String QUEUE_NAME = "queue_name";

    /**
     * Exchange name
     */
    private static final String EXCHANGE_NAME = "exchange_name";

    public static void main(String[] args) throws Exception {
        // Connection factory
        ConnectionFactory factory = new ConnectionFactory();
        // Set the service address (replace with the access point address copied in
        factory.setUri("amqp://***");
        // Set vhost (replace with the vhost name copied in the console)
        factory.setVirtualHost(VHOST_NAME);
        // Set the username (use the role name in the permission configuration of t
        factory.setUsername(USERNAME);
        // Set the password (use the role key)
        factory.setPassword("eyJh****");
        // Get the connection address
        Connection connection = factory.newConnection();
        // Establish a channel
        Channel channel = connection.createChannel();
        // Bind the message exchange
        channel.exchangeDeclare(EXCHANGE_NAME, "fanout");
        // Declare the queue message
        channel.queueDeclare(QUEUE_NAME, true, false, false, null);
        // Bind the message exchange (`EXCHANGE_NAME` must exist in the TDMQ for Ra
        channel.queueBind(QUEUE_NAME, EXCHANGE_NAME, "");
        System.out.println(" [Consumer1] Waiting for messages.");
        // Subscribe to the message
        channel.basicConsume(QUEUE_NAME, false, "ConsumerTag", new DefaultConsumer(
            @Override
            public void handleDelivery(String consumerTag, Envelope envelope,
                                     AMQP.BasicProperties properties, byte[] body
                                     throws IOException {
                // Received message for business logic processing
                System.out.println("Received: " + new String(body, StandardCharsets
                channel.basicAck(envelope.getDeliveryTag(), false);
            }
        )
    }
}
```

<pre>         }     } } </pre>	
Parameter	Description
QUEUE_NAME	Queue name, which can be obtained from the queue list in the console.
EXCHANGE_NAME	Exchange name, which can be obtained from the exchange list in the console.
factory.setUri	<p>Cluster access address, which can be obtained from <b>Access Address</b> in the <b>Operation</b></p> 
factory.setVirtualHost	<p>Vhost name in the format of "<b>cluster ID +   + vhost name</b>", which can be copied on the <b>Virtual Host</b></p> 
factory.setUsername	Role name, which can be copied on the <a href="#">Role Management</a> page.
factory.setPassword	<p>Role key, which can be copied in the <b>Key</b> column on the <a href="#">Role Management</a> page.</p> 

## Step 4. Query messages

To check whether messages are sent to TDMQ for RabbitMQ successfully, view the connected consumer status on the [Cluster](#) > [Queue](#) page in the console.



Basic Info

Vhost

Exchange

Queue

Routing

Current Vhost

vhost1

TTL (retention period for unconsumed messages)

1 day

Max TPS

8000

Create (3/1000)

Queue Name	Monitoring	Consumer Info	Description
test		Online Consumer 0 TPS 0 Total Heap 0	

Basic Info

Message Heap	0	Dead Letter Exchange	-
Creation Time	2022-03-21 16:54:03	Dead Letter RoutingKey	-
Automatic Deletion	Close	Online Consumer	0

Consumer List

Client Address	Consumer Tag	Creation Time
No data yet		

Total items: 0

20 / pag

**Note:**

Above is a sample based on the pub/sub pattern of RabbitMQ. For more samples, see [Demo](#) or [RabbitMQ Tutorials](#).

# SDK for Go

Last updated : 2024-01-03 11:45:32

## Overview

This document describes how to use open-source SDK to send and receive messages by using the SDK for Go as an example and helps you better understand the message sending and receiving processes.

## Prerequisites

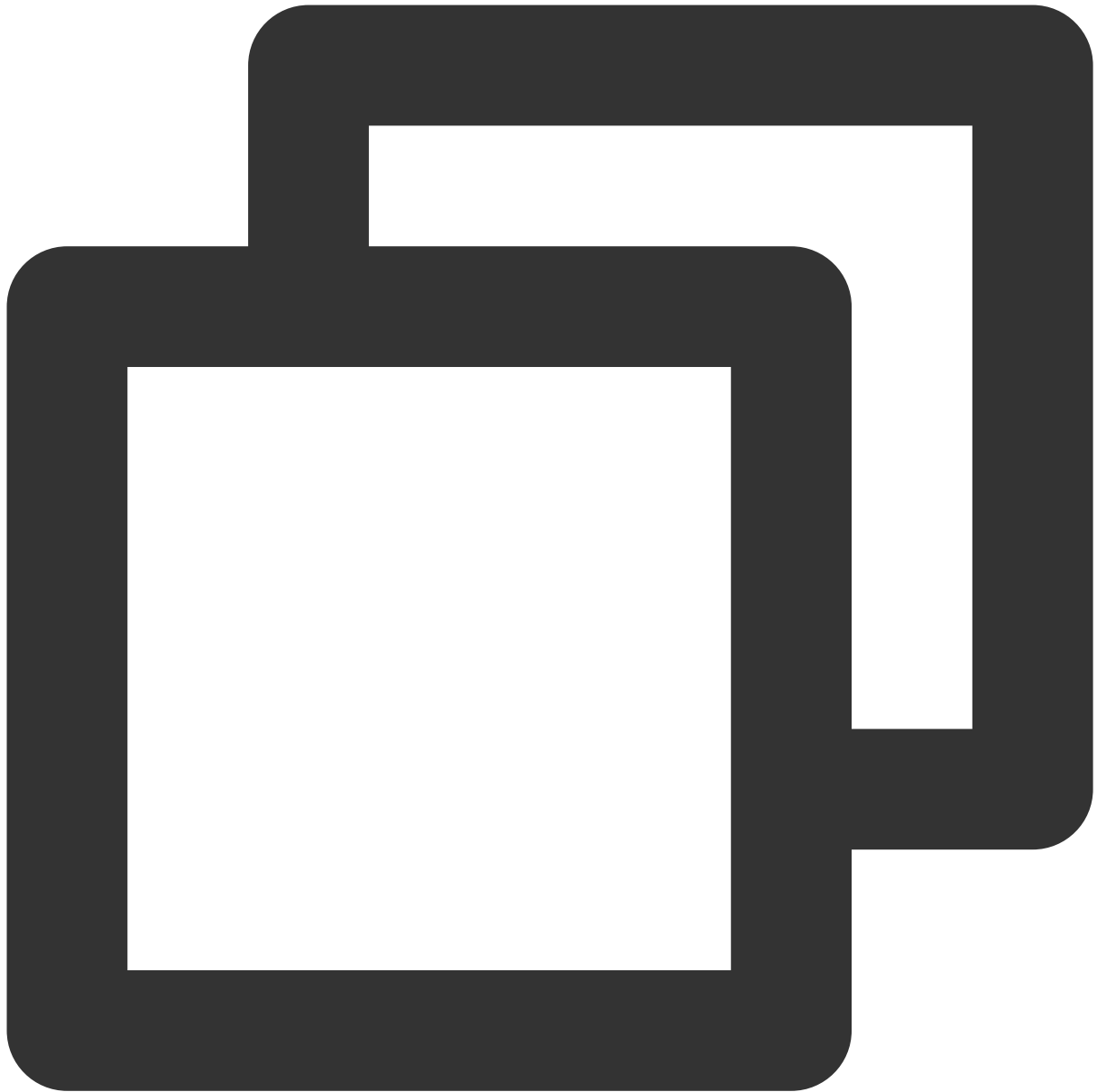
[You have created the required resources.](#)

[You have installed Go](#)

[You have downloaded the demo](#)

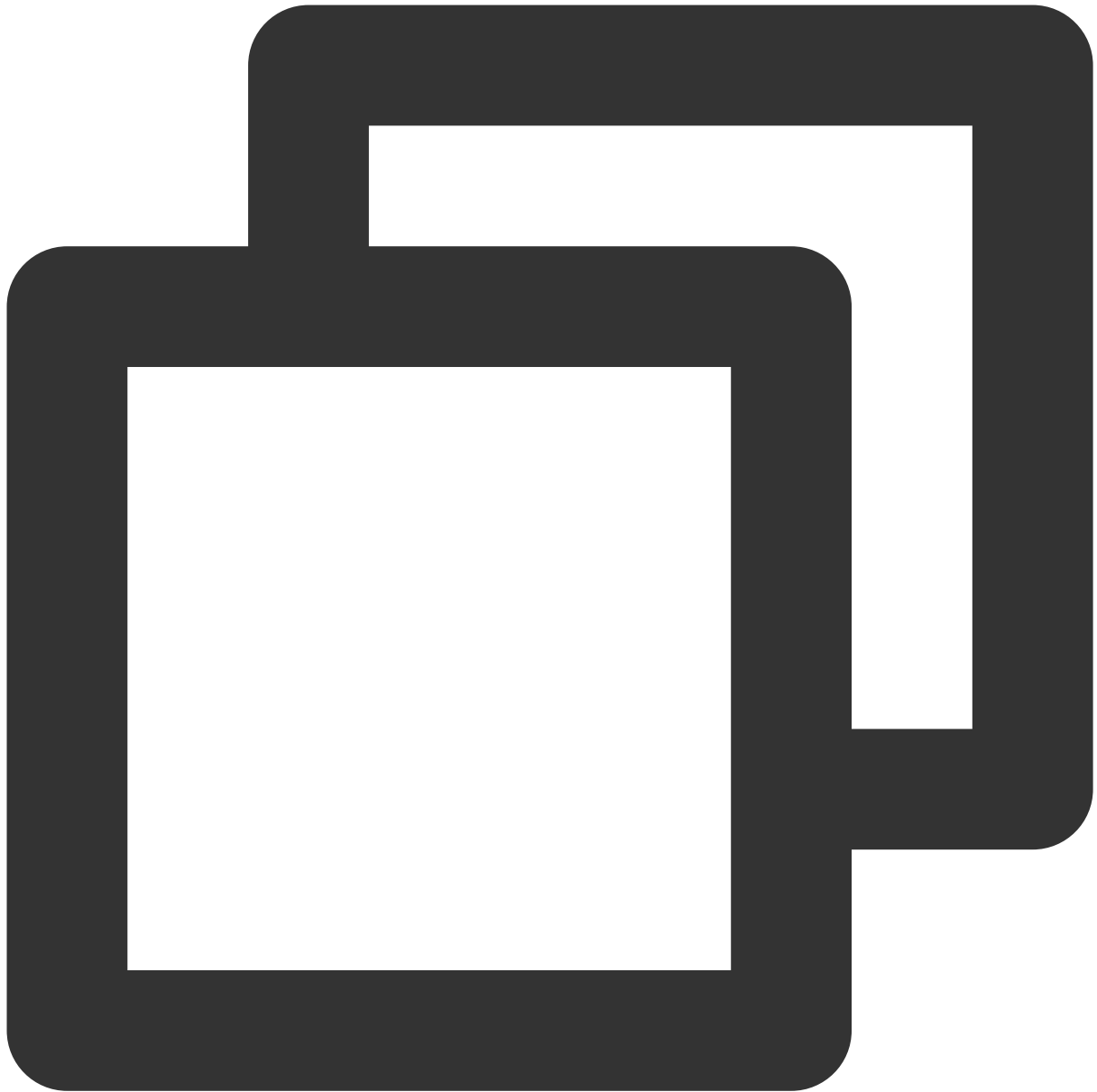
## Directions

1. Run the following command to install the required package in the client environment:



```
go get "github.com/rabbitmq/amqp091-go"
```

2. After the installation is completed, import the package to your Go project file.

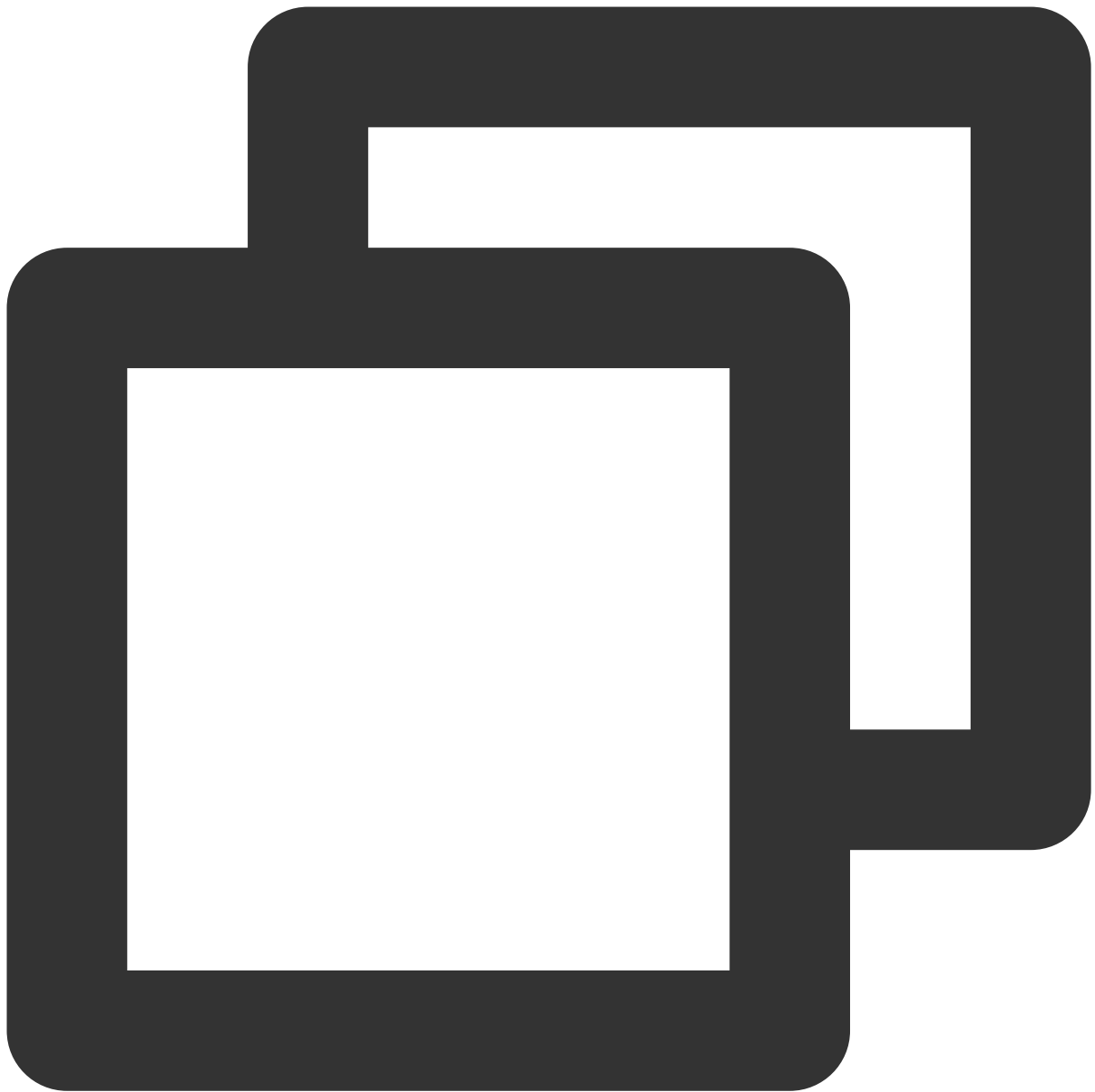


```
import (amqp "github.com/rabbitmq/amqp091-go")
```

After the import, you can use the client in your project.

## Samples

1. Establish the connection and communication channel.



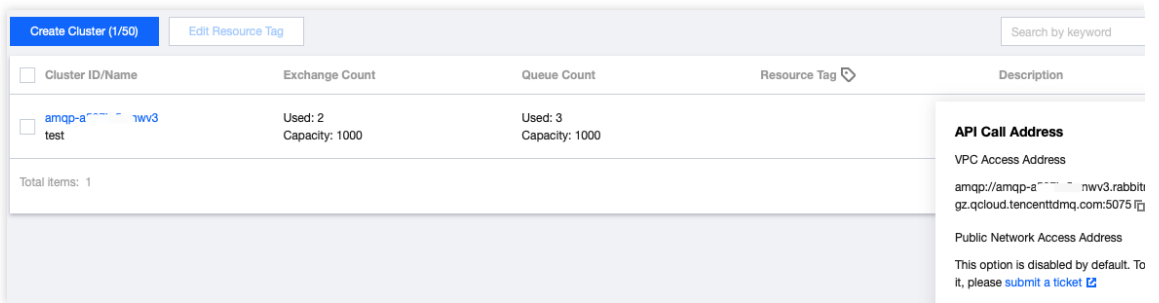
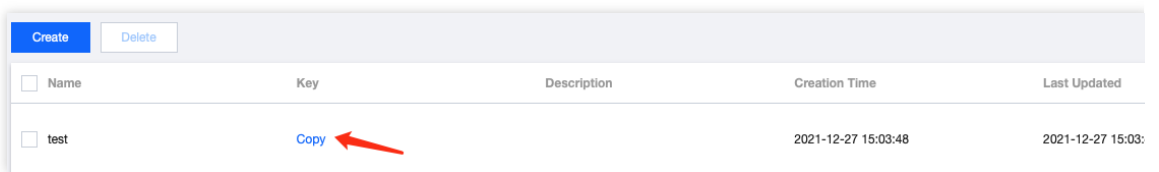
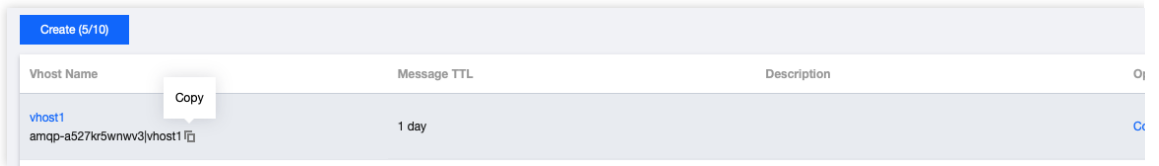
```
// Required parameters
const (
    host      = "amqp-xx.rabbitmq.x.tencenttdmq.com" // Service access address
    username  = "roleName" // Role name in the console
    password  = "eyJrZX..." // Role key
    vhost     = "amqp-xx|Vhost" // Full name of the vhost to be used
)
// Create a connection
conn, err := amqp.Dial("amqp://" + username + ":" + password + "@" + host + ":5672/")
failOnError(err, "Failed to connect to RabbitMQ")
defer func(conn *amqp.Connection) {
```

```

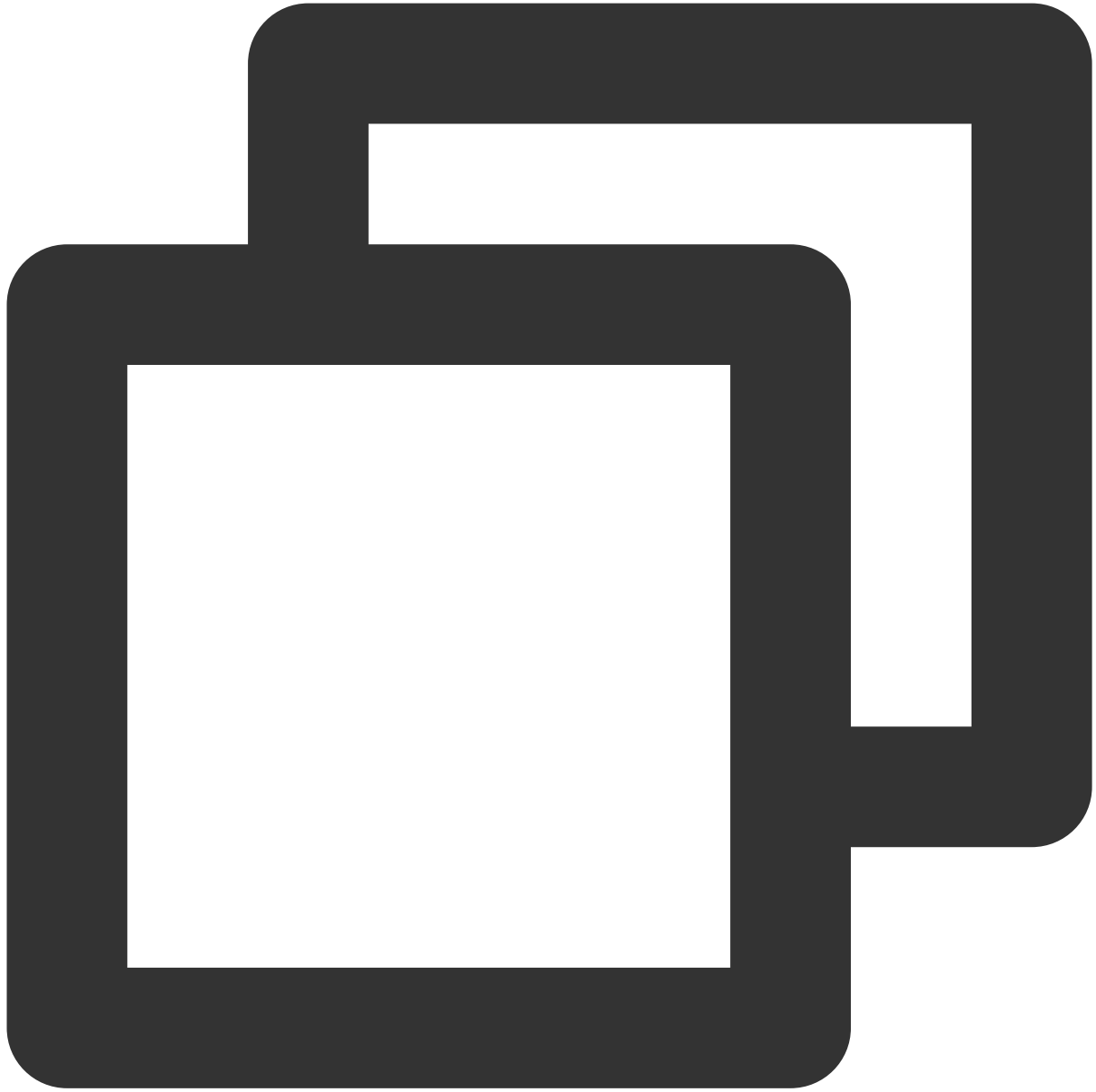
err := conn.Close()
if err != nil {
}
}(conn)

// Establish a channel
ch, err := conn.Channel()
failOnError(err, "Failed to open a channel")
defer func(ch *amqp.Channel) {
    err := ch.Close()
    if err != nil {
    }
}(ch)

```

Parameter	Description
host	<p>Cluster access address, which can be obtained from <b>Access Address</b> in the <b>Operation</b> column on 1</p> 
username	Role name, which can be copied on the <a href="#">Role Management</a> page.
password	<p>Role key, which can be copied in the <b>Key</b> column on the <a href="#">Role Management</a> page.</p> 
vhost	<p>Vhost name in the format of "<b>cluster ID +   + vhost name</b>", which can be copied on the <b>Vhost</b> page</p> 

## 2. Declare the exchange.



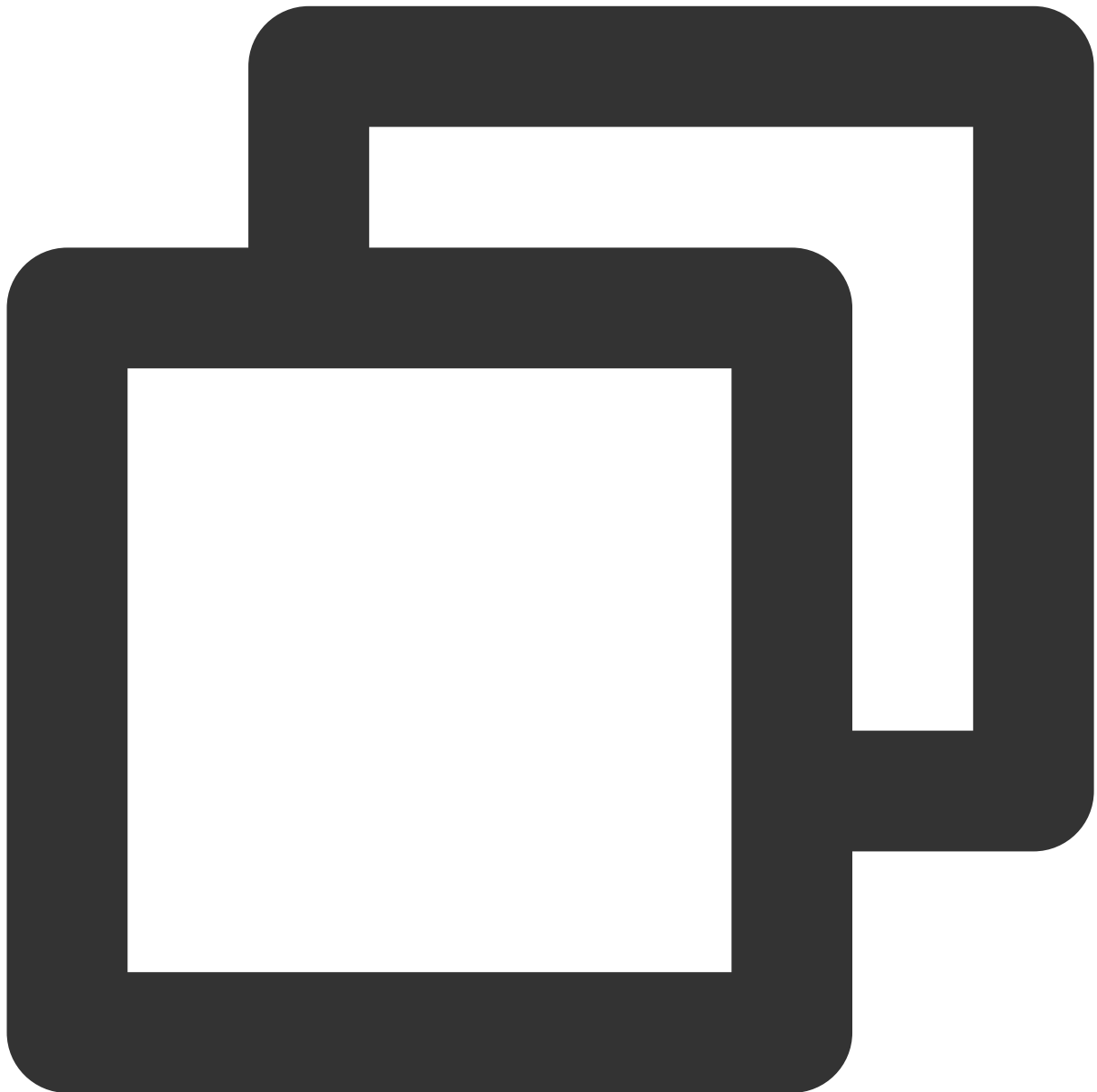
```
// Declare the exchange (the name and type must be the same as those of the existing exchange)
err = ch.ExchangeDeclare(
    "logs-exchange", // Exchange name
    "fanout", // Exchange type
    true, // durable
    false, // auto-deleted
    false, // internal
    false, // no-wait
    nil, // arguments
)
```

```
)  
failOnError(err, "Failed to declare a exchange")
```

### 3. Publish messages.

Messages can be directly sent to the exchange or the specified queue ( `hello world` and `work` ).

Publish messages to the exchange:



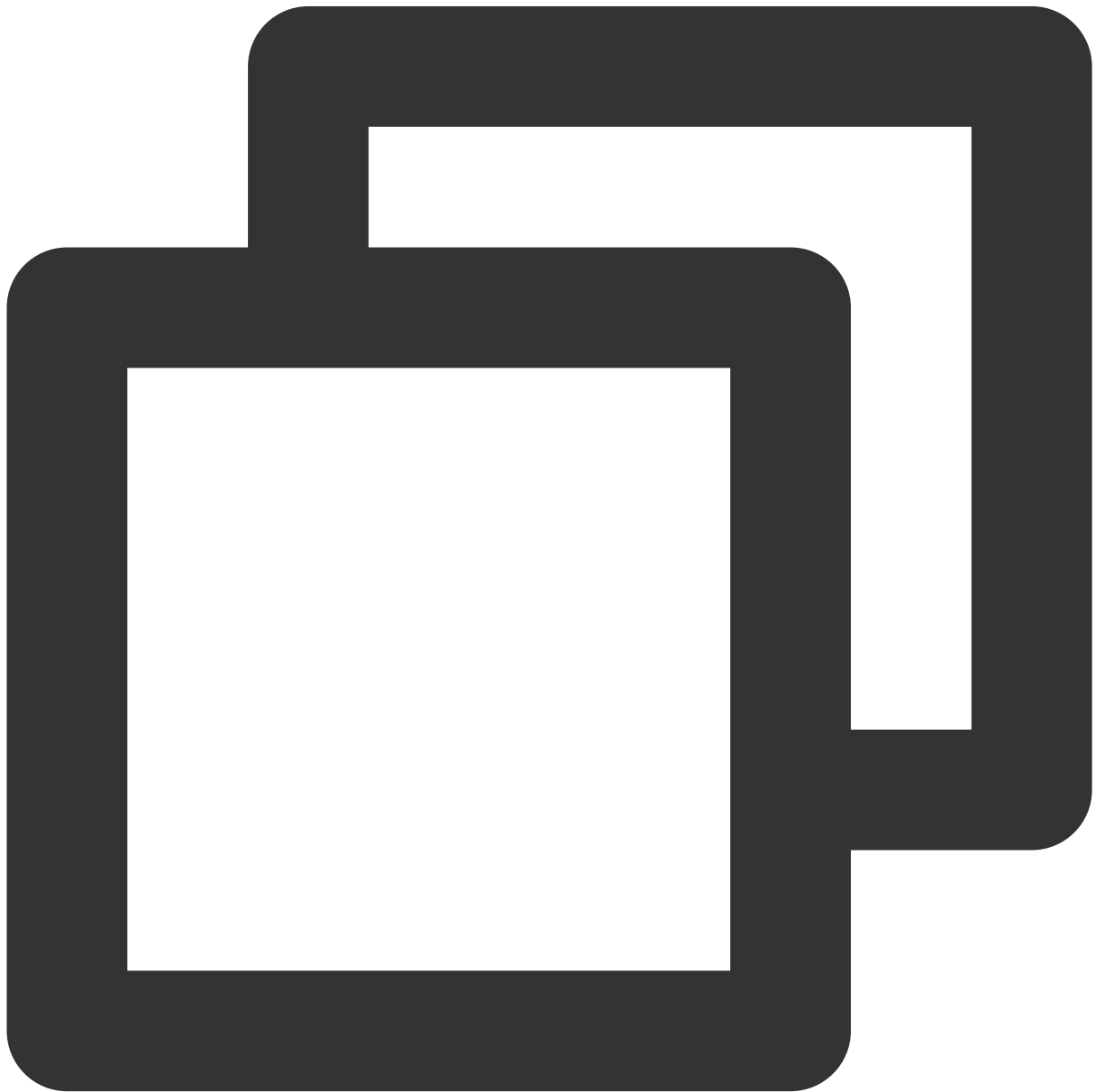
```
// Message content  
body := "this is new message."  
// Publish messages to the exchange  
err = ch.Publish(  

```



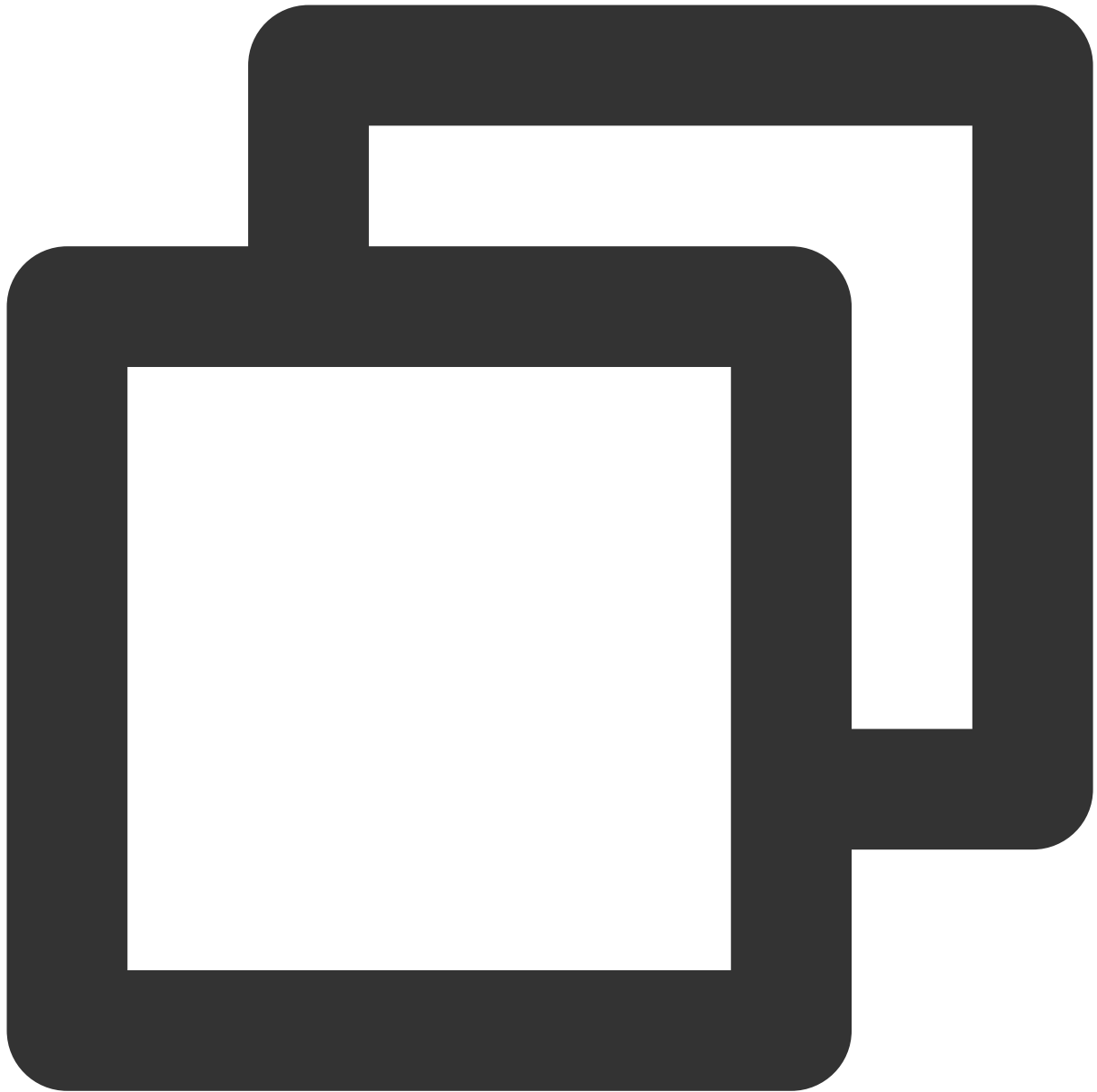
```
"logs-exchange",          // exchange
"", // Routing key (set whether the routing key is required based on the used e
false,                    // mandatory
false,                    // immediate
amqp.Publishing{
    ContentType: "text/plain",
    Body:        []byte(body),
})
failOnError(err, "Failed to publish a message")
```

Publish messages to the specified queue:



```
// Publish messages to the specified message queue
err = ch.Publish(
    "",          // exchange
    queue.Name, // routing key
    false,      // mandatory
    false,      // immediate
    amqp.Publishing{
        ContentType: "text/plain",
        Body:        []byte(body),
    })
failOnError(err, "Failed to publish a message")
```

#### 4. Subscribe to messages.



```
// Create a consumer and consume messages in the specified message queue
msgs, err := ch.Consume(
    "message-queue", // message-queue
    "",              // consumer
    false,           // Set to manual acknowledgment as needed
    false,           // exclusive
    false,           // no-local
    false,           // no-wait
    nil,             // args
)
failOnError(err, "Failed to register a consumer")
```

```
// Get messages in the message queue
forever := make(chan bool)
go func() {
    for d := range msgs {
        log.Printf("Received a message: %s", d.Body)
        t := time.Duration(1)
        time.Sleep(t * time.Second)
        // Manually return the acknowledgment
        d.Ack(false)
    }
}()
log.Printf(" [Consumer] Waiting for messages.")
<-forever
```

5. The consumer uses the routing key.



```
// You need to specify the exchange and routing key in the message queue
err = ch.QueueBind(
    q.Name, // queue name
    "routing_key", // routing key
    "topic_demo", // exchange
    false,
    nil,
)
failOnError(err, "Failed to bind a queue")
```

**Note:**

---

For detailed samples, see [Demo](#) or [RabbitMQ Tutorials](#).

# SDK for Python

Last updated : 2024-01-03 11:45:32

## Overview

This document describes how to use open-source SDK to send and receive messages using the SDK for Python as an example and helps you better understand the message sending and receiving processes.

## Prerequisites

[You have created the required resources.](#)

[You have installed Python](#)

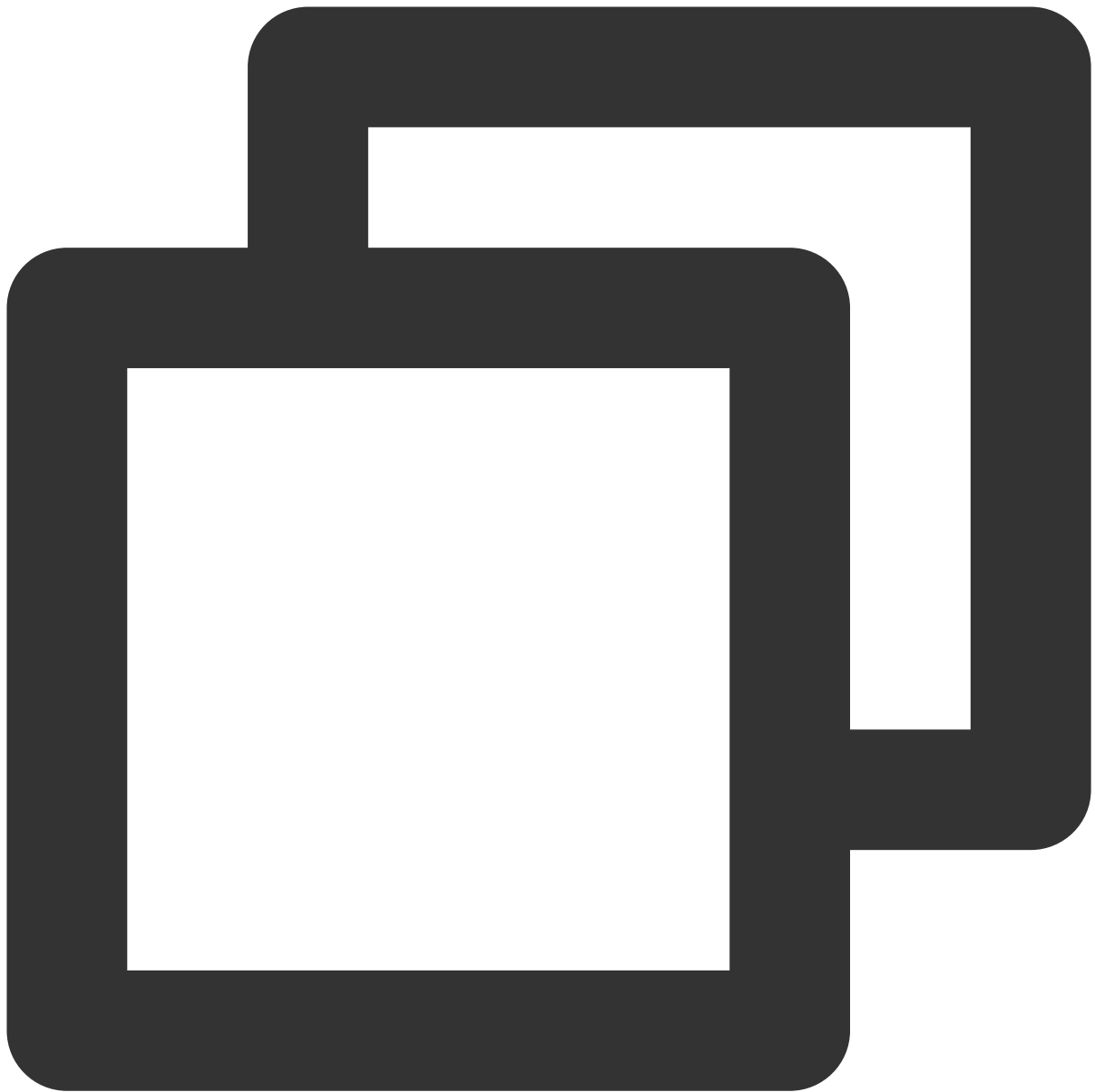
[You have installed pip](#)

[You have downloaded the demo](#)

## Directions

### Step 1. Add dependencies

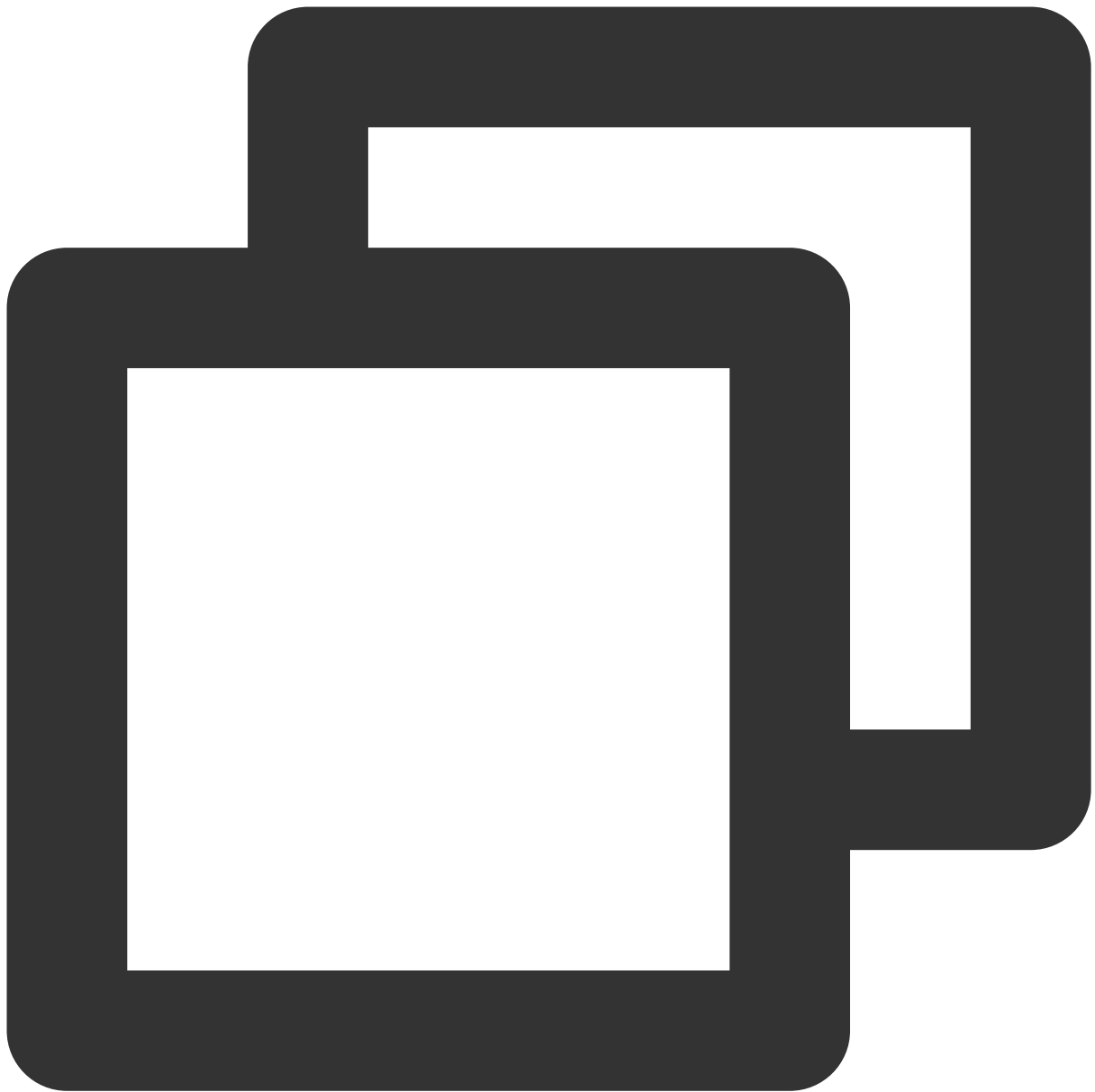
1. RabbitMQ officially recommends Pika. First, you need to install Pika in the client environment.



```
python -m pip install pika --upgrade
```

2. Import Pika when creating the client.

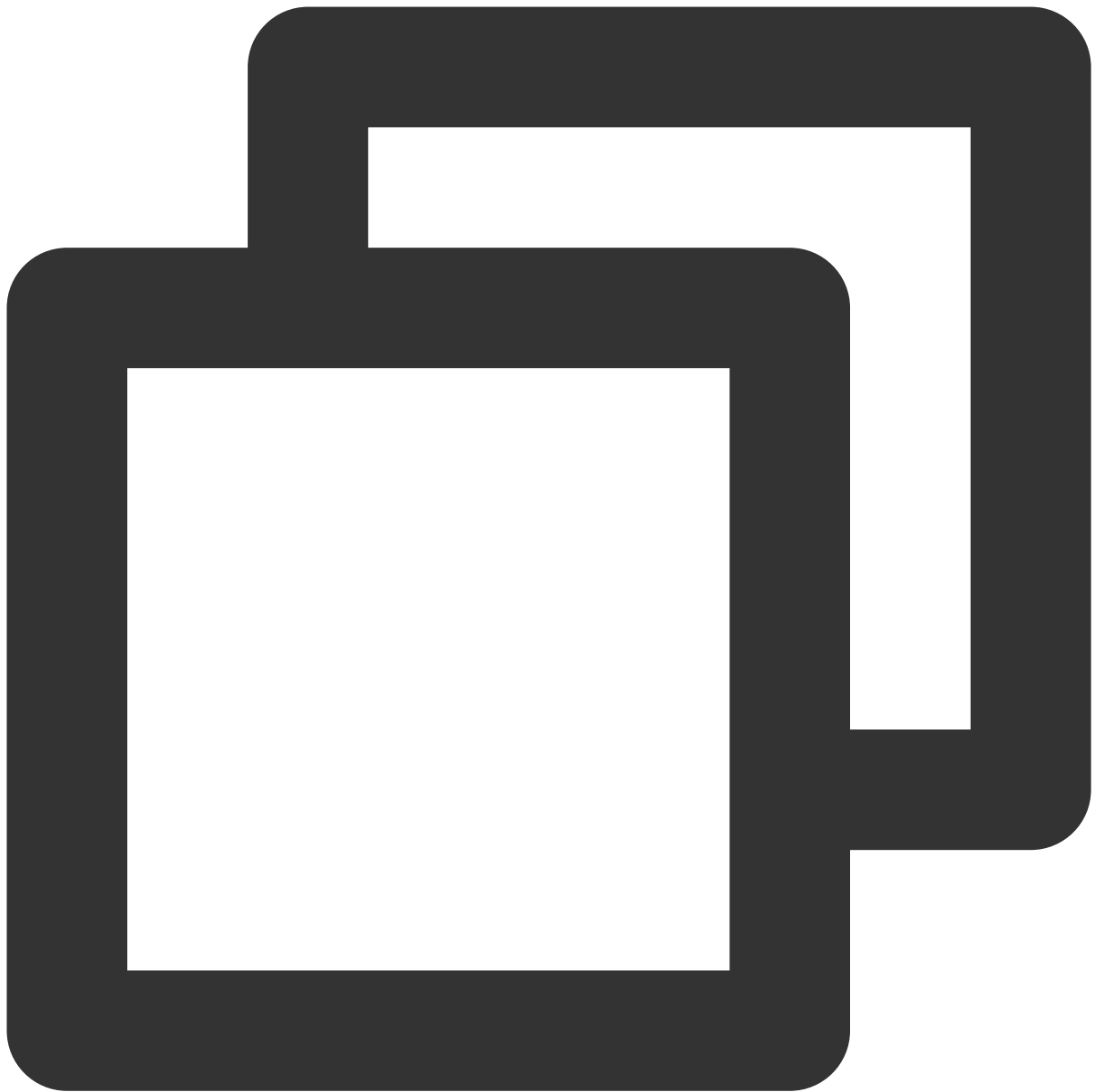




```
import pika
```

## Step 2. Produce messages

Create, compile, and run the message producing program `messageProducer.py` .



```
import pika

# Use the username and password to create a login credential object
credentials = pika.PlainCredentials('rolename', 'eyJr***')
# Create a connection
connection = pika.BlockingConnection(pika.ConnectionParameters(
    host='amqp-xx.rabbitmq.x.com', port=5672, virtual_host='amqp-xxx|Vhostname', cr
# Establish a channel
channel = connection.channel()
# Declare the exchange
channel.exchange_declare(exchange='direct_exchange', exchange_type="direct")
```

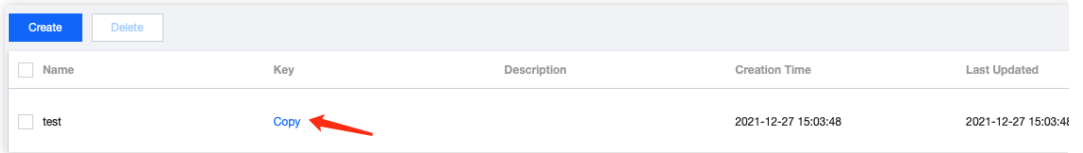
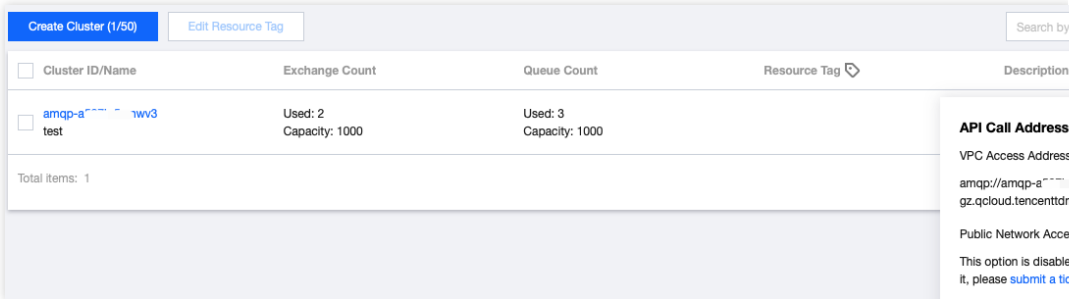
```

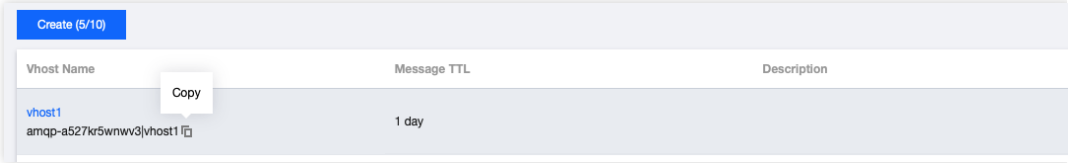
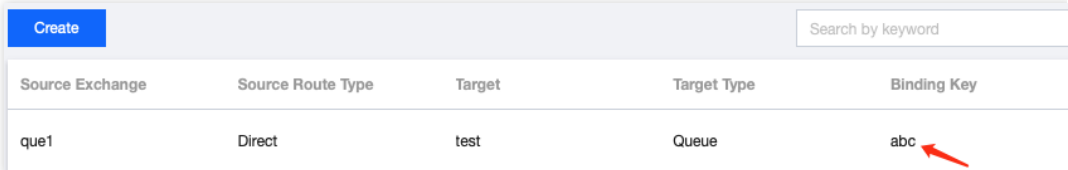
routingKeys = ['aaa.bbb.ccc', 'aaa.bbb.ddd', 'aaa.ccc.zzz', "xxx.yyy.zzz"]

for routingKey in routingKeys:
    # Send a message to the specified exchange
    # If you send a message without specifying the exchange, you need to specify the
    channel.basic_publish(exchange='direct_exchange',
                          routing_key=routingKey,
                          body=(routingKey + 'This is a new direct message.').encode(),
                          properties=pika.BasicProperties(
                              delivery_mode=2, # Set message persistence
                          ))

    print('send success msg to rabbitmq')
connection.close()

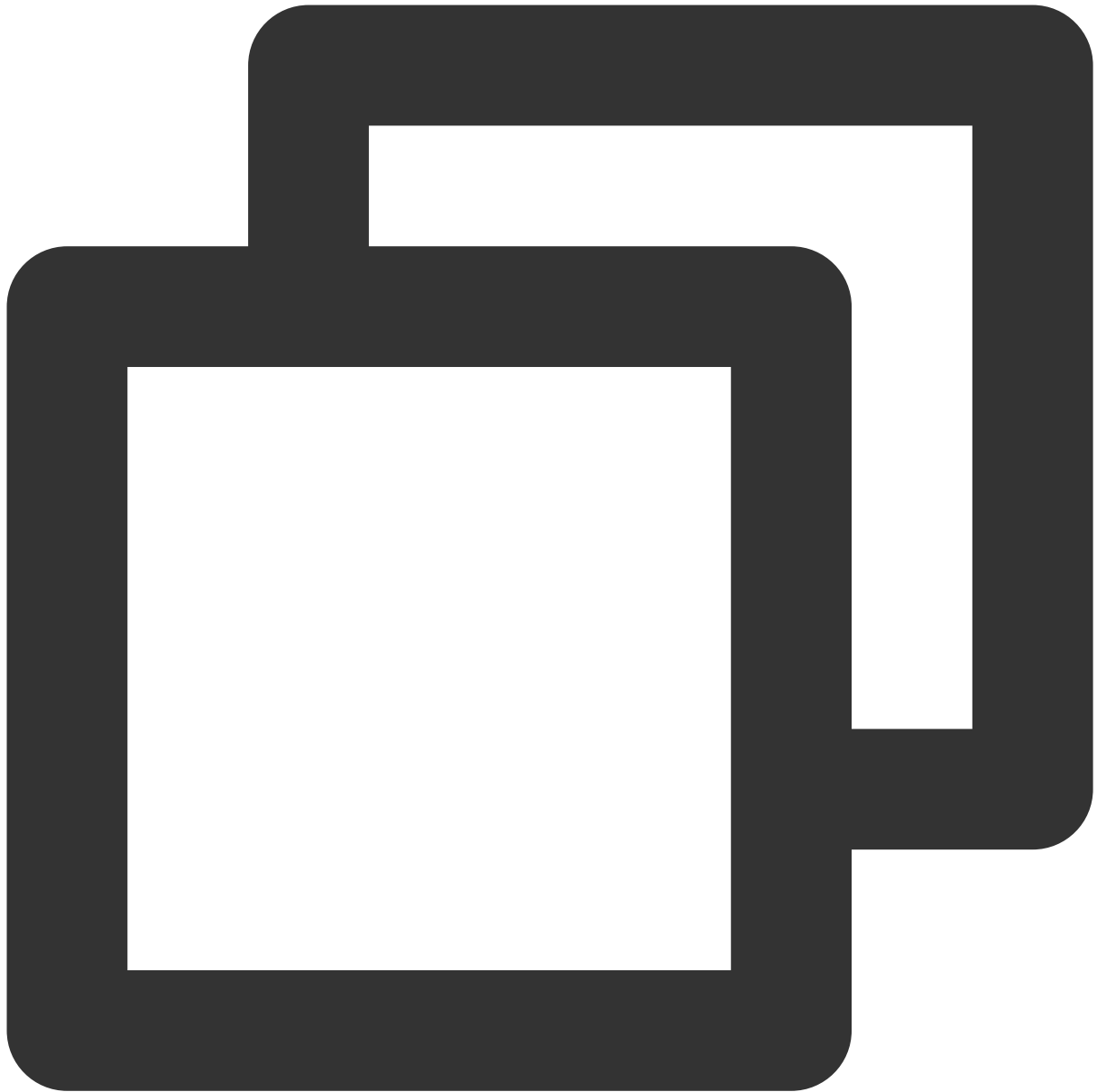
```

Parameter	Description
rolename	Role name, which can be copied on the <a href="#">Role Management</a> page.
eyJr***	<p>Role key, which can be copied in the <b>Key</b> column on the <a href="#">Role Management</a> page.</p> 
host	<p>Cluster access address, which can be obtained from <b>Access Address</b> in the <b>Operation</b> column</p> 
port	Cluster access port, which can be obtained from <b>Access Address</b> in the <b>Operation</b> column of the
virtual_host	Vhost name in the format of " <b>cluster ID +   + vhost name</b> ", which can be copied on the <b>Vhos</b>

	
direct_exchange	Exchange name, which can be obtained from the exchange list in the console.
routingKeys	<p>Message routing rule, which can be obtained in the <b>Binding Key</b> column in the binding list in the console.</p> 

### Step 3. Consume messages

Create, compile, and run the message consuming program `messageConsumer.py`.



```
import os
import pika
import sys

def main():
    # Use the username and password to create a login credential object
    credentials = pika.PlainCredentials('rolename', 'eyJr***')
    # Create a connection
    connection = pika.BlockingConnection(pika.ConnectionParameters(
        host='amqp-xx.rabbitmq.x.com', port=5672, virtual_host='amqp-xxx|Vhostname'
```

```

# Establish a channel
channel = connection.channel()

# Declare the message queue
channel.queue_declare(queue='route_queue1', exclusive=True, durable=True)

# Bind the message queue to the exchange and specify the routing key
routing_keys = ['aaa.bbb.ccc', 'aaa.bbb.ddd']
for routingKey in routing_keys:
    channel.queue_bind(exchange='direct_exchange', queue="route_queue1", routingKey=routingKey)

# Set that only one unacknowledged message can be received
channel.basic_qos(prefetch_count=1)

# Message consumption logic
def callback(ch, method, properties, body):
    print(" [Consumer1(Direct 'aaa.bbb.ccc'/'aaa.bbb.ddd')] Received (%r)" % body)
    # Manually return the ACK
    ch.basic_ack(delivery_tag=method.delivery_tag)

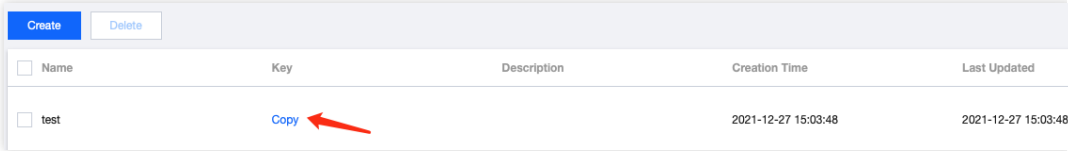
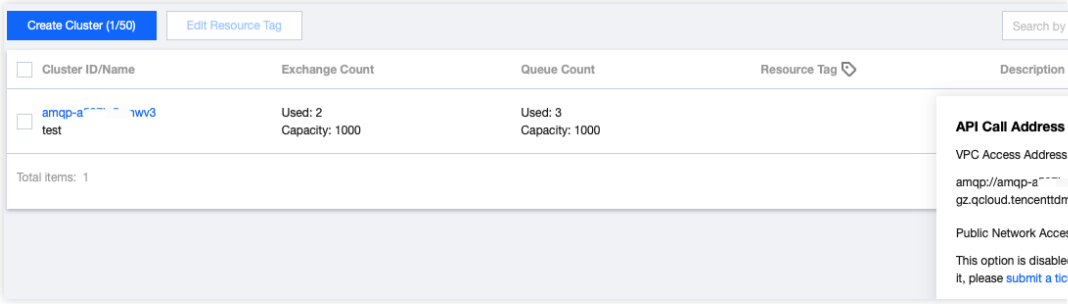
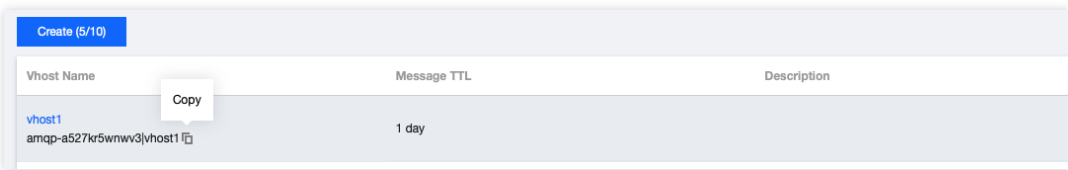
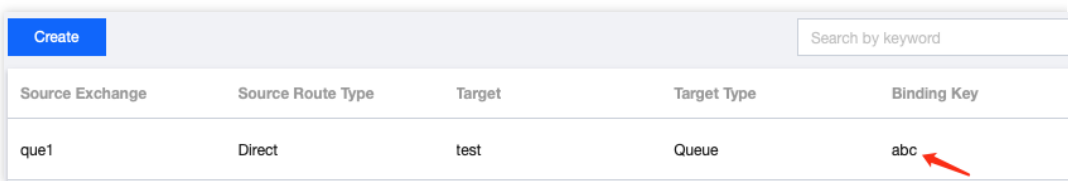
# Create a consumer to consume messages in the message queue
channel.basic_consume(queue='route_queue1',
                      on_message_callback=callback,
                      auto_ack=False) # Set to manual acknowledgment

print(" [Consumer1(Direct 'aaa.bbb.ccc'/'aaa.bbb.ddd')] Waiting for messages. To shutdown press any key")
channel.start_consuming()

if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        print('Interrupted')
        try:
            sys.exit(0)
        except SystemExit:
            os._exit(0)

```

Parameter	Description
rolename	Role name, which can be copied on the <a href="#">Role Management</a> page.
eyJr***	Role key, which can be copied in the <b>Key</b> column on the <a href="#">Role Management</a> page.

	
host	<p>Cluster access address, which can be obtained from <b>Access Address</b> in the <b>Operation</b> column</p> 
port	<p>Cluster access port, which can be obtained from <b>Access Address</b> in the <b>Operation</b> column or</p>
virtual_host	<p>Vhost name in the format of "<b>cluster ID +   + vhost name</b>", which can be copied on the <b>Vhos</b></p> 
direct_exchange	<p>Exchange name, which can be obtained from the exchange list in the console.</p>
route_queue1	<p>Queue name, which can be obtained from the queue list in the console.</p>
routingKey	<p>Message routing rule, which can be obtained in the <b>Binding Key</b> column in the binding list in the</p> 

## Step 4. View messages



To check whether messages are sent to TDMQ for RabbitMQ successfully, view the connected consumer status on the **Cluster** > **Queue** page in the console.

Basic InfoVhostExchangeQueueRouting

Current Vhostvhost1TTL (retention period for unconsumed messages) 1 dayMax TPS ⓘ 8000

Create (3/1000)

Queue NameMonitoringConsumer Info ↕Description

▼ testOnline Consumer 0TPS 0Total Heap 0

Basic Info

Message Heap0Dead Letter Exchange-

Creation Time2022-03-21 16:54:03Dead Letter RoutingKey-

Automatic DeletionCloseOnline Consumer0

Consumer List

Client Address	Consumer Tag	Creation Time
No data yet		

Total items: 020 ▼ / page

Total items: 120 ▼ / p

**Note:**

For the complete sample code and other use cases, see [Demo](#) or [RabbitMQ Tutorials](#).



# SDK for PHP

Last updated : 2024-01-03 11:45:32

## Overview

This document describes how to use open-source SDK to send and receive messages by using the SDK for PHP as an example and helps you better understand the message sending and receiving processes.

## Prerequisites

[Install PHP 5.6 or later](#)

[Install PEAR](#)

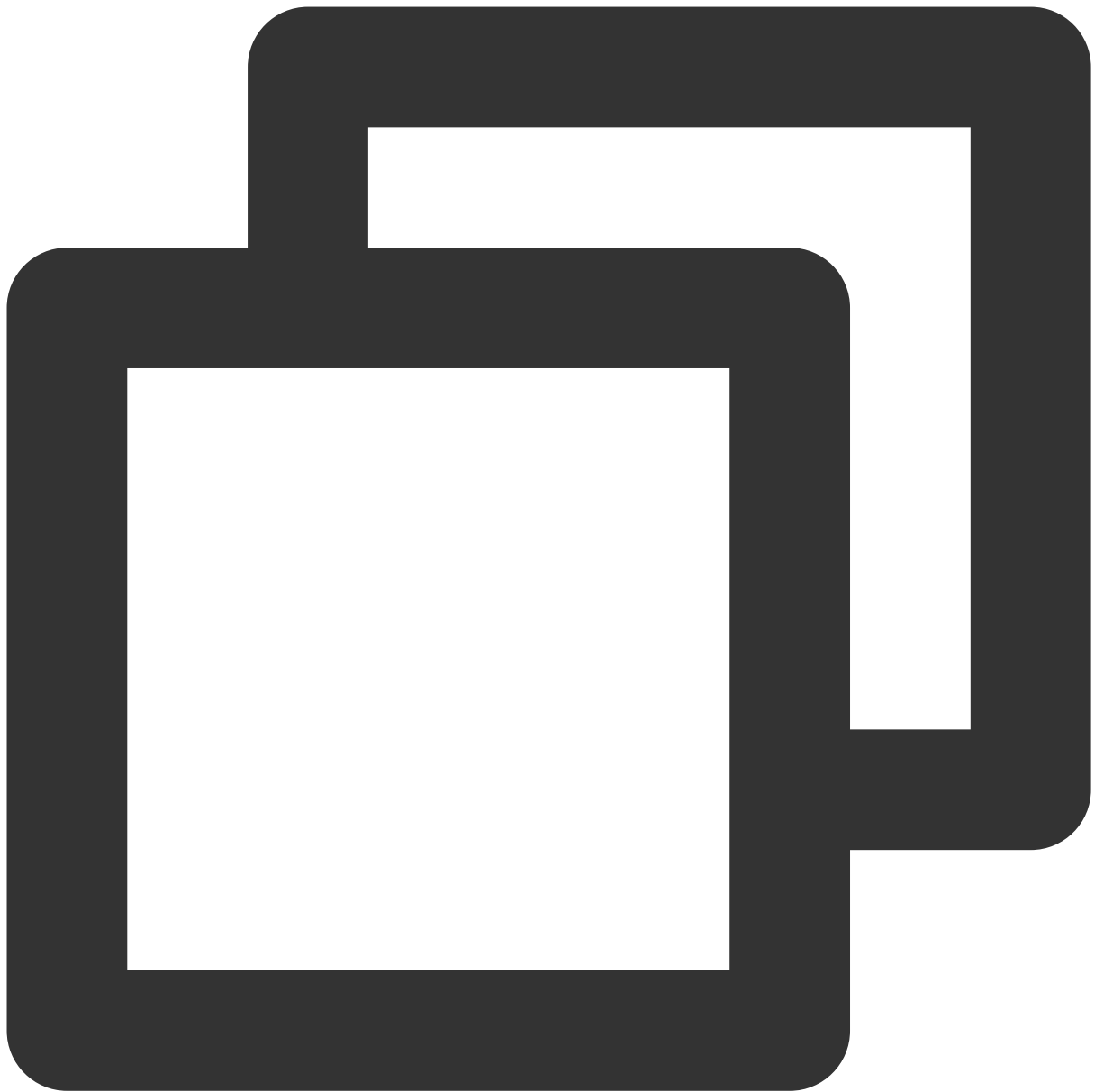
[You have downloaded the demo](#)

## Directions

### Step 1. Install the php-amqplib library

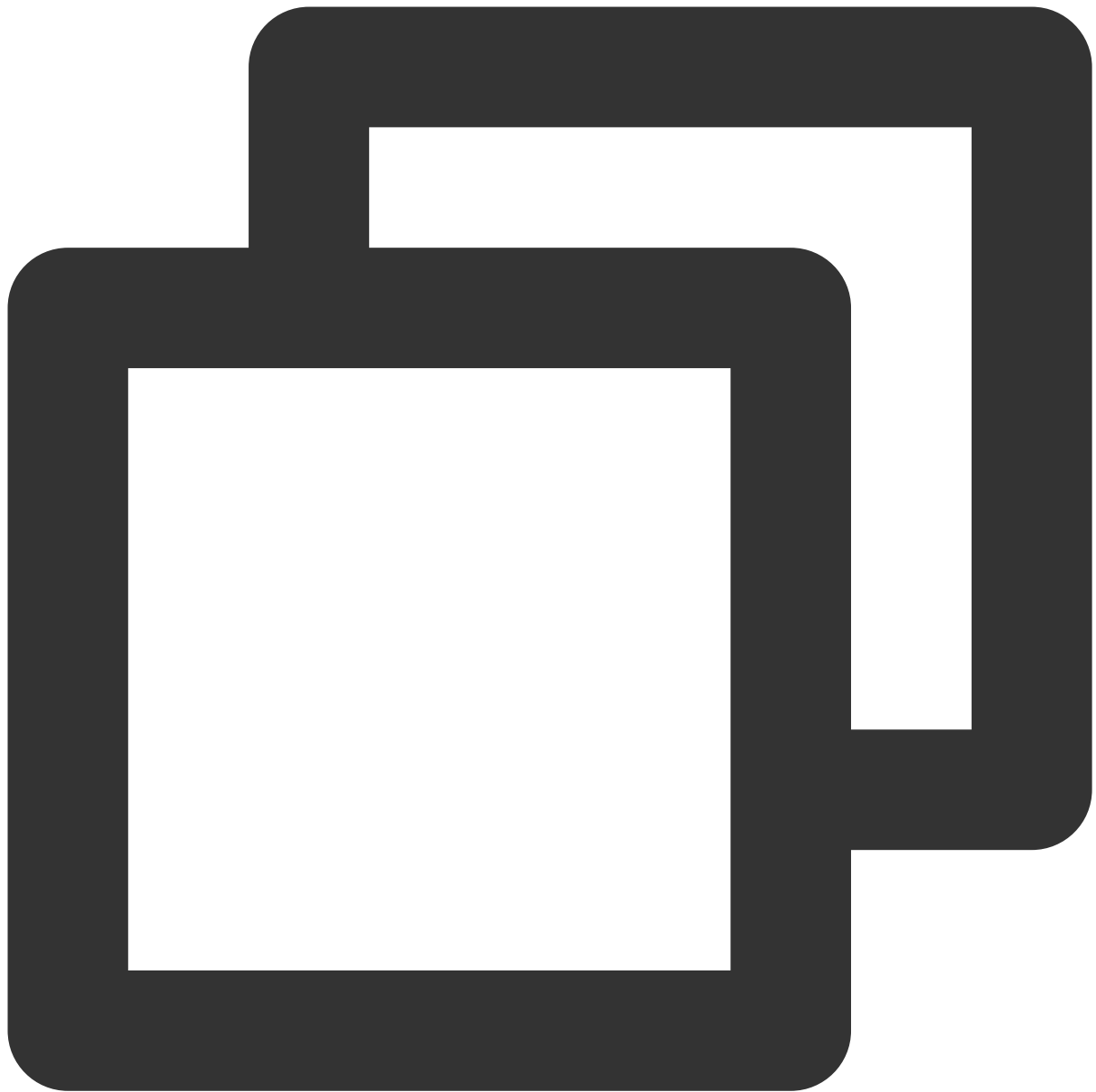
RabbitMQ officially recommends the `php-amqplib` client. First, you need to import the `php-amqplib` library into your project.

1. Add the `composer.json` file to your project.



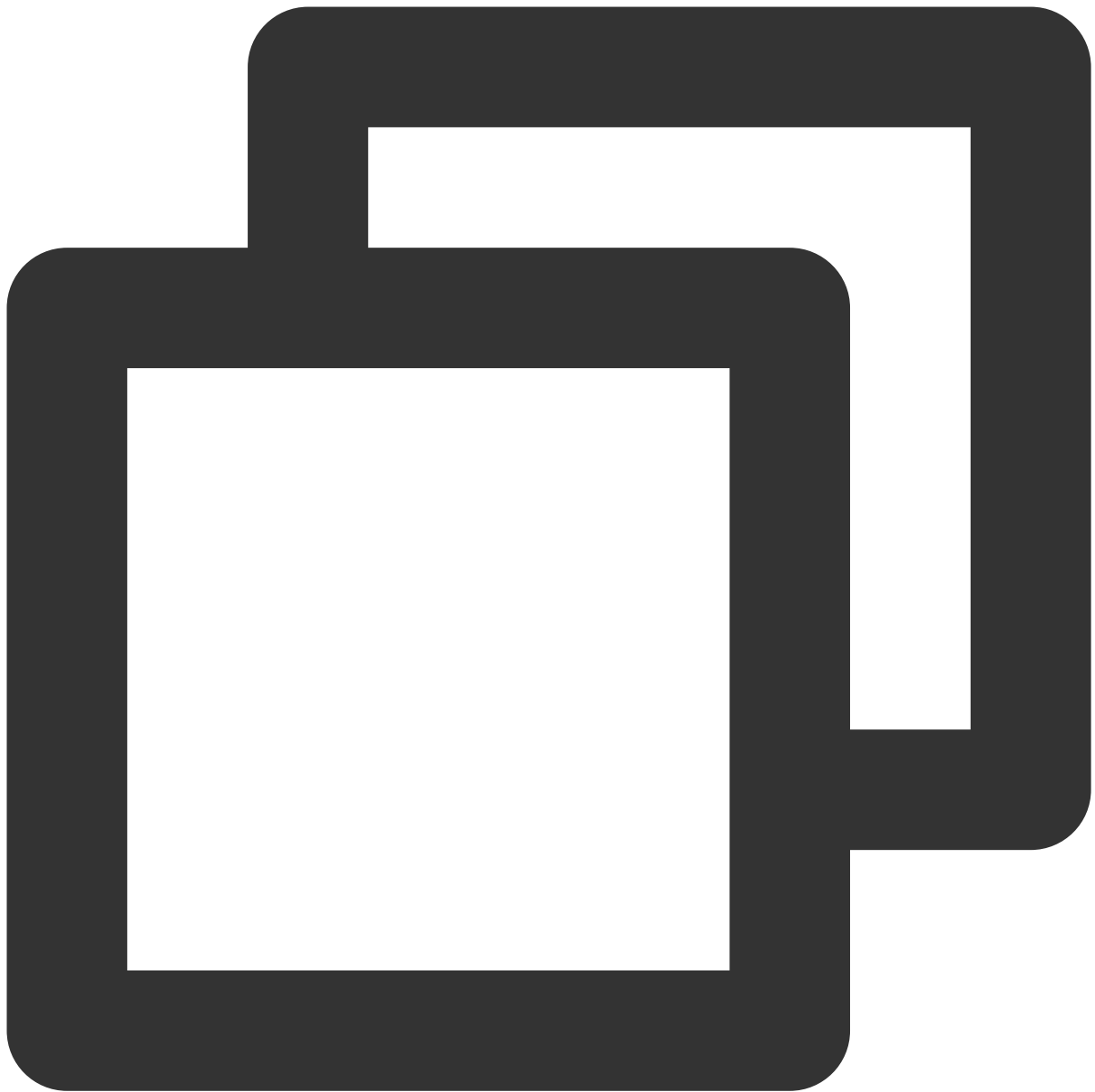
```
{  
  "require": {  
    "php-amqplib/php-amqplib": ">=3.0"  
  }  
}
```

2. Use Composer for installation.



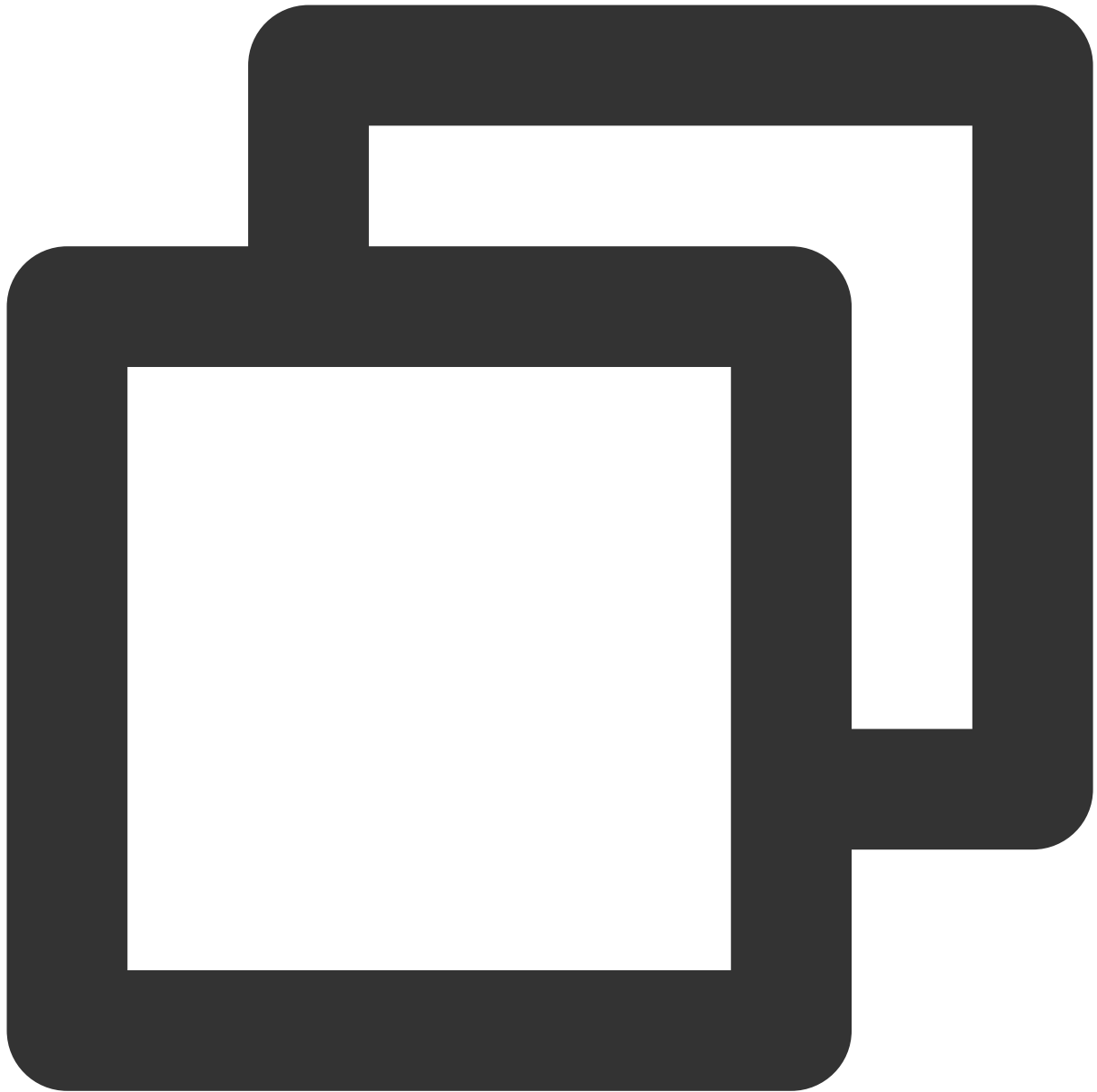
```
composer.phar install
```

You can also run the following command:



```
composer install
```

3. Import the library file into the client files to create a client.

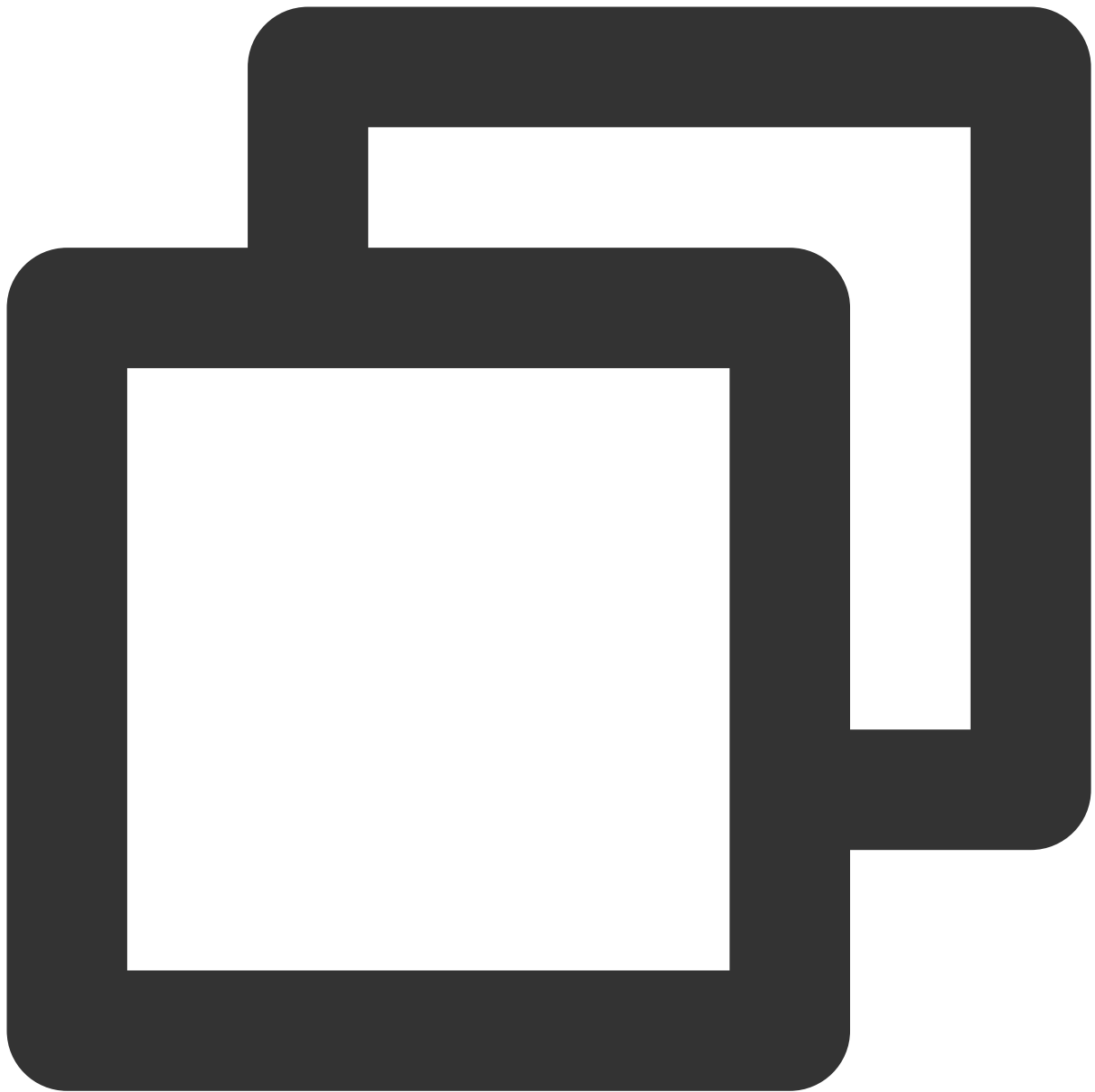


```
require_once('../vendor/autoload.php');
```

After completing the above steps, you can create a connection for interaction with the server.

## Step 2. Send messages

Create and compile a message producing program (with a direct exchange as an example).



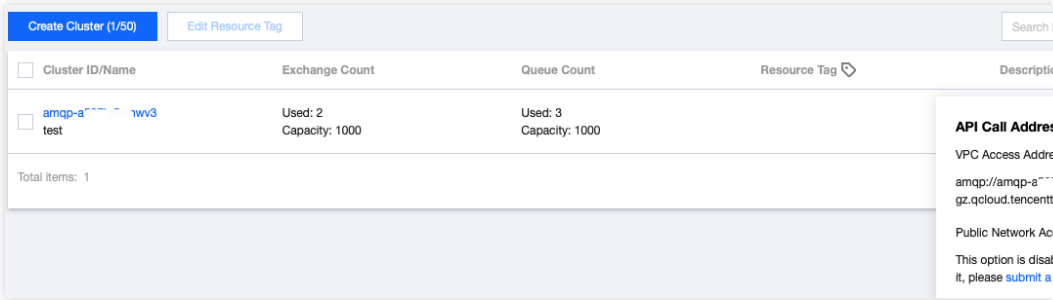
```
require_once('../vendor/autoload.php');

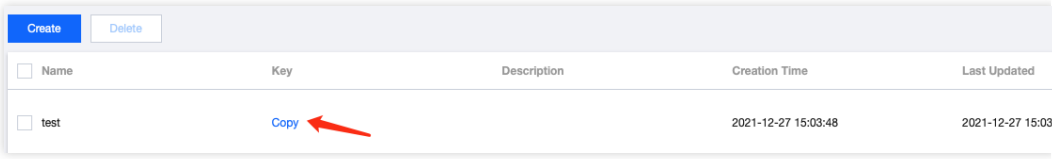
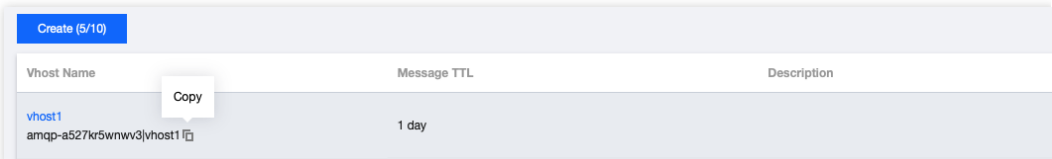
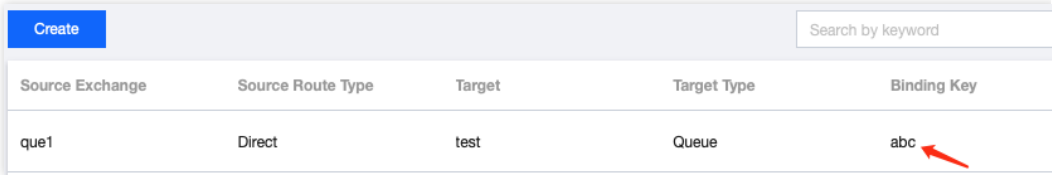
use PhpAmqpLib\\Connection\\AMQPStreamConnection;
use PhpAmqpLib\\Message\\AMQPMessage;

$exchange_name = 'exchange_name';
$exchange_type = 'direct';

// Create a connection
$connection = new AMQPStreamConnection(
    $host,
```

```
$port,  
$username,  
$password,  
$vhost,  
false,  
'PLAIN');  
  
// Establish a channel  
$channel = $connection->channel();  
  
// Declare the exchange  
$channel->exchange_declare($exchange_name, $exchange_type, false, true, false);  
  
// Set the message routing key  
$routing_keys = array('info', 'warning', 'error');  
  
for ($x = 0; $x < count($routing_keys); $x++) {  
    // Message content  
    $msg = new AMQPMessage('This is a direct[' . $routing_keys[$x] . '] message!');  
    // Send a message to the specified exchange and set the routing key  
    $channel->basic_publish($msg, $exchange_name, $routing_keys[$x]);  
  
    echo " [Producer(Routing)] Sent '" . $msg->body . "'\n";  
}  
  
// Release resources  
$channel->close();  
$connection->close();
```

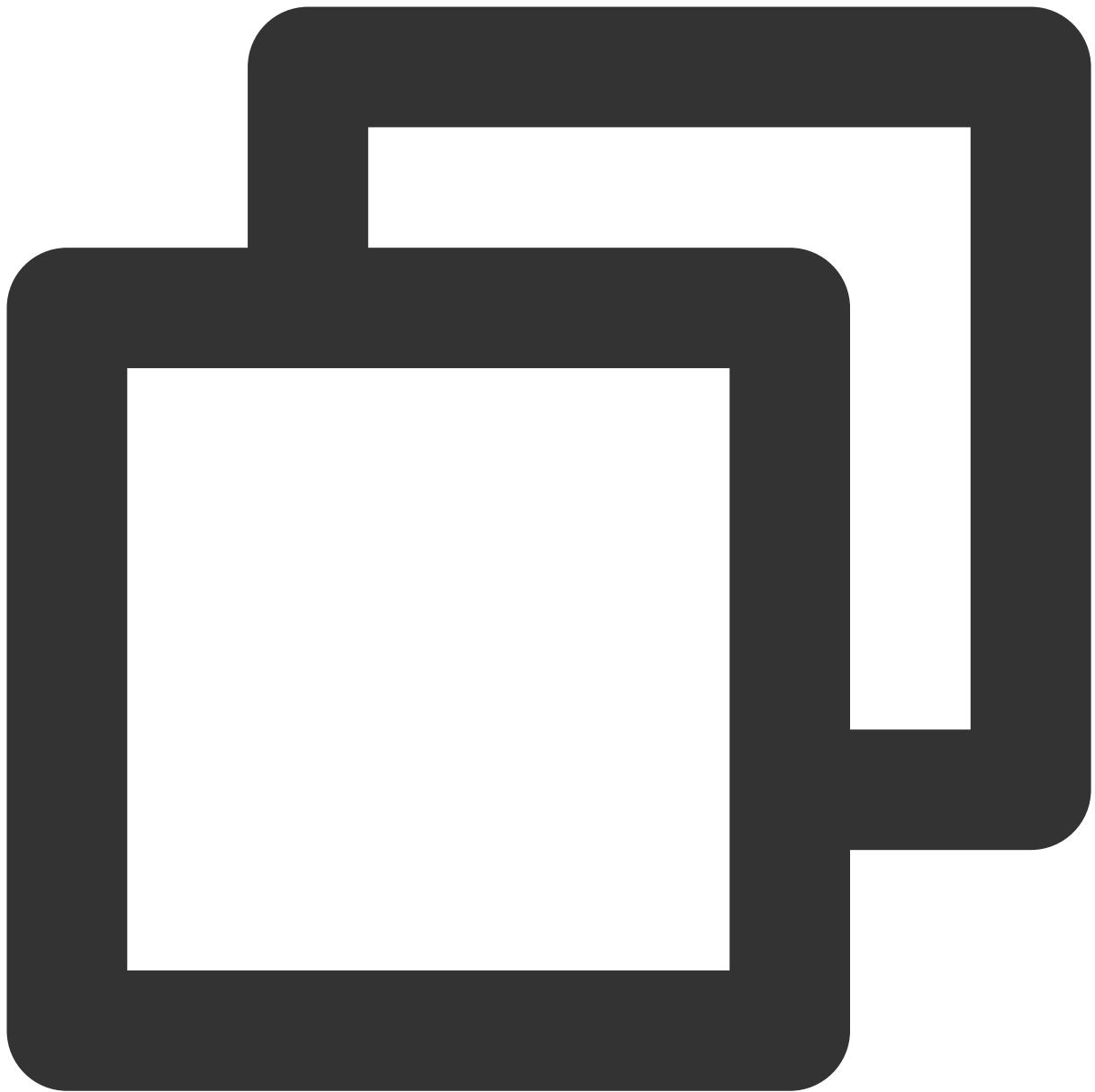
Parameter	Description
\$exchange_name	Exchange name, which can be obtained from the exchange list in the console.
\$exchange_type	It must be the same as the type of the above exchange.
\$host	<p>Cluster access address, which can be obtained from <b>Access Address</b> in the <b>Operation</b> column of the console.</p> 
\$port	Cluster access port.
\$username	Role name, which can be copied on the <a href="#">Role Management</a> page.

\$password	<p>Role key, which can be copied in the <b>Key</b> column on the <a href="#">Role Management</a> page.</p> 
\$vhost	<p>Vhost name in the format of "<b>cluster ID +   + vhost name</b>", which can be copied on the <b>Vho</b></p> 
\$routing_keys[\$x]	<p>Routing key bound to the consumer message queue, which is also the message routing rule a <b>Key</b> column in the binding list in the console.</p> 

### Step 3. Consume messages

Create and compile a message consuming program.





```
<?php

require_once('../vendor/autoload.php');
require_once('../Constant.php');

use PhpAmqpLib\\Connection\\AMQPStreamConnection;

$exchange_name = 'exchange_name';
$exchange_type = 'direct';
$queue_name = 'route_queue1';
```

```
// Create a connection
$connection = new AMQPStreamConnection(
    $host,
    $port,
    $username,
    $password,
    $vhost,
    false,
    'PLAIN');
// Establish a channel
$channel = $connection->channel();
// Declare the exchange
$channel->exchange_declare($exchange_name, $exchange_type, false, true, false);
// Declare the message queue
$channel->queue_declare($queue_name, false, true, false, false);

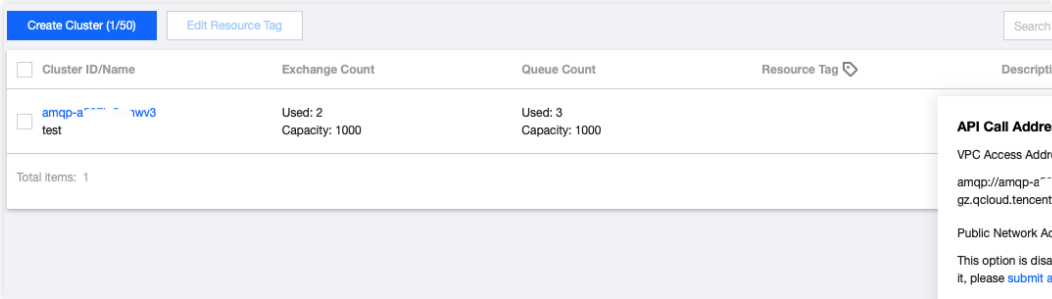
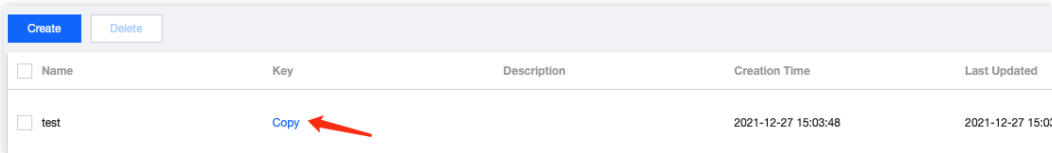
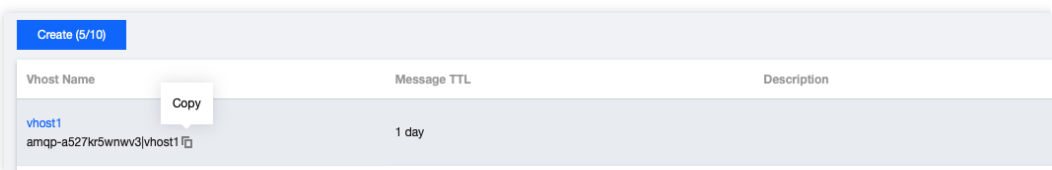
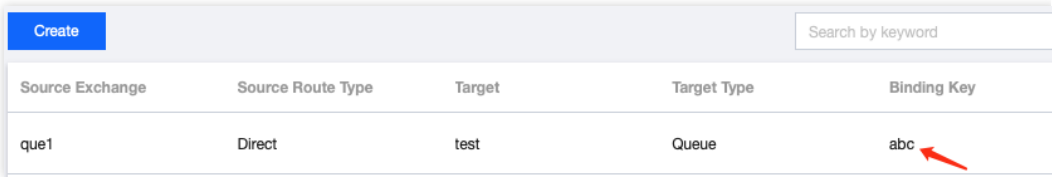
// Set the queue routing key
$routing_keys = array('info', 'waring', 'error');
for ($x = 0; $x < count($routing_keys); $x++) {
    // Bind the message queue to the specified exchange and set `routingKey`
    $channel->queue_bind($queue_name, $exchange_name, $routing_keys[$x]);
}

echo " [Consumer1(Routing: info/waring/error)] Waiting for messages. To exit press

// Message callback (message consumption logic)
$callback = function ($msg) {
    echo ' [Consumer1(Routing: info/waring/error)] Received ', $msg->body, "\\n";
};
// Create a consumer to listen on the specified message queue
$channel->basic_consume($queue_name, '', false, true, false, false, $callback);

while ($channel->is_open()) {
    $channel->wait();
}
// Disable the resource
$channel->close();
$connection->close();
```

Parameter	Description
\$exchange_name	Exchange name, which can be obtained from the exchange list in the console.
\$exchange_type	It must be the same as the type of the above exchange.
\$queue_name	Queue name, which can be obtained from the queue list in the console.

\$host	<p>Cluster access address, which can be obtained from <b>Access Address</b> in the <b>Operation</b> column.</p> 
\$port	Cluster access port.
\$username	Role name, which can be copied on the <a href="#">Role Management</a> page.
\$password	<p>Role key, which can be copied in the <b>Key</b> column on the <a href="#">Role Management</a> page.</p> 
\$vhost	<p>Vhost name in the format of "<b>cluster ID +   + vhost name</b>", which can be copied on the <b>Vhost</b> page.</p> 
\$routing_keys[\$x]	<p>Routing key bound to the consumer message queue, which is also the message routing rule in the <b>Key</b> column in the binding list in the console.</p> 

## Step 4. View messages

To check whether messages are sent to TDMQ for RabbitMQ successfully, view the connected consumer status on the [Cluster](#) > **Queue** page in the console.

Basic InfoVhostExchangeQueueRouting

Current Vhostvhost1TTL (retention period for unconsumed messages) 1 dayMax TPS ⓘ 8000

Create (3/1000)

Queue NameMonitoringConsumer Info ↕Description

▼ test📊Online Consumer 0TPS 0Total Heap 0🔄

Basic Info

Message Heap0Dead Letter Exchange-

Creation Time2022-03-21 16:54:03Dead Letter RoutingKey-

Automatic DeletionCloseOnline Consumer0

Consumer List

Client AddressConsumer TagCreated

No data yet

Total items: 02

Total items: 1

**Note:**

For the complete sample code and other use cases, see [Demo](#) or [RabbitMQ Tutorials](#).

# SDK for C++

Last updated : 2024-08-16 17:39:06

## Overview

This document describes how to use open-source SDK to send and receive messages by using the SDK for C++ as an example and helps you better understand the message sending and receiving processes.

## Prerequisites

[You have created the required resources.](#)

[You have downloaded the demo](#)

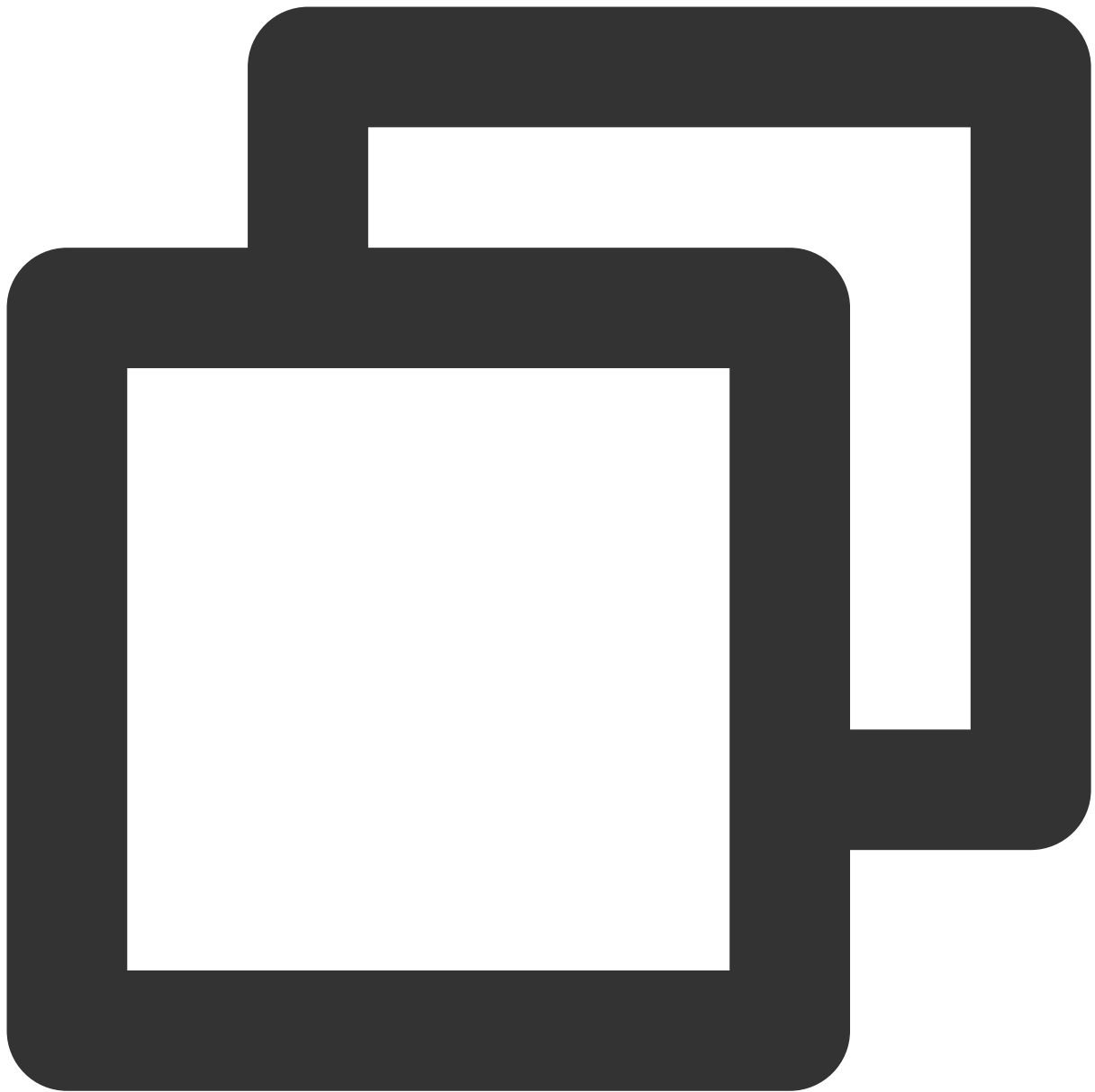
## Directions

### Step 1. Prepare the environment

1. Install a [C and C++ client library](#). This document uses AMQP-CPP as an example.
2. Import the dynamic library and header file.

### Step 2. Produce messages

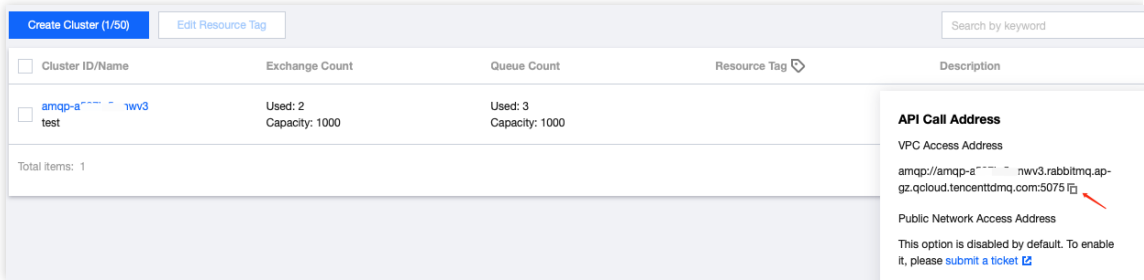
1. Establish a connection.



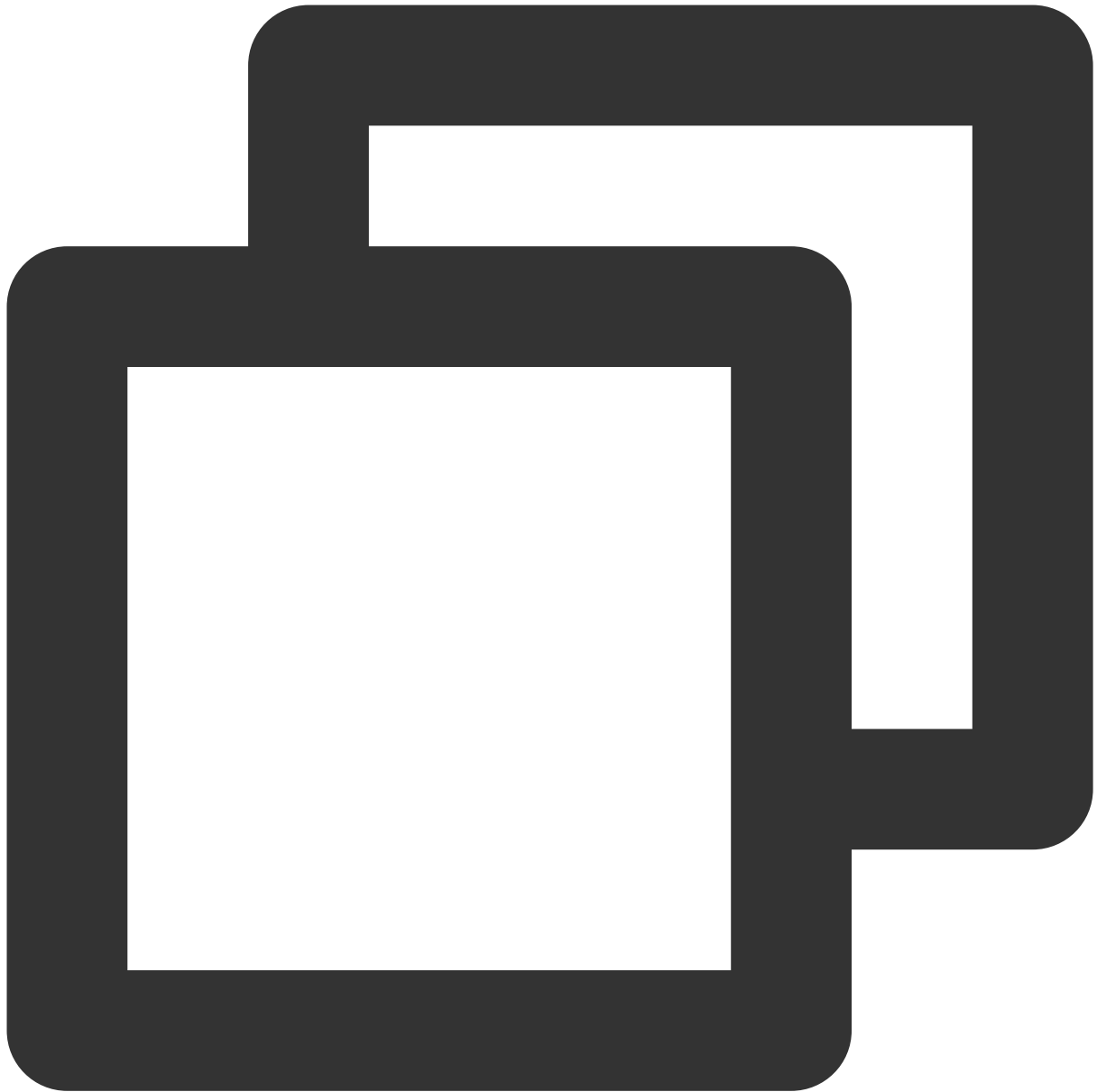
```
auto evbase = event_base_new();
LibEventHandlerMyError hndl(evbase);

// Establish a connection
AMQP::TcpConnection connection(&hndl, AMQP::Address(
    "amqp://admin:eyJrZXlJZC...@amqp-xxx.rabbitmq.ap-sh.public.tencenttdmq.com:5
    // The service address is in the format of "amqp://username:password@host:po
// Create a channel
AMQP::TcpChannel channel(&connection);
channel.onError([&evbase](const char *message) {
    std::cout << "Channel error: " << message << std::endl;
```

```
event_base_loopbreak(evbase);
});
```

Parameter	Description
String	<p>Cluster access address, which can be obtained in the console by clicking Access Address in the “Operations” of the Cluster page.</p> 
port	Port number in the cluster access address.
name-server	Role name, which can be copied on the <a href="#">Role Management</a> page.
password	Role token, which can be copied in the Token column on the <a href="#">Role Management</a> page.
vhost	Vhost name in the format of “cluster ID +   + vhost name”.

## 2. Send messages.



```
// Declare an exchange
channel.declareExchange(exchange_name, AMQP::ExchangeType::direct);

// Send messages to the exchange
channel.publish(exchange_name, routing_key, "Hello client this is a info message");

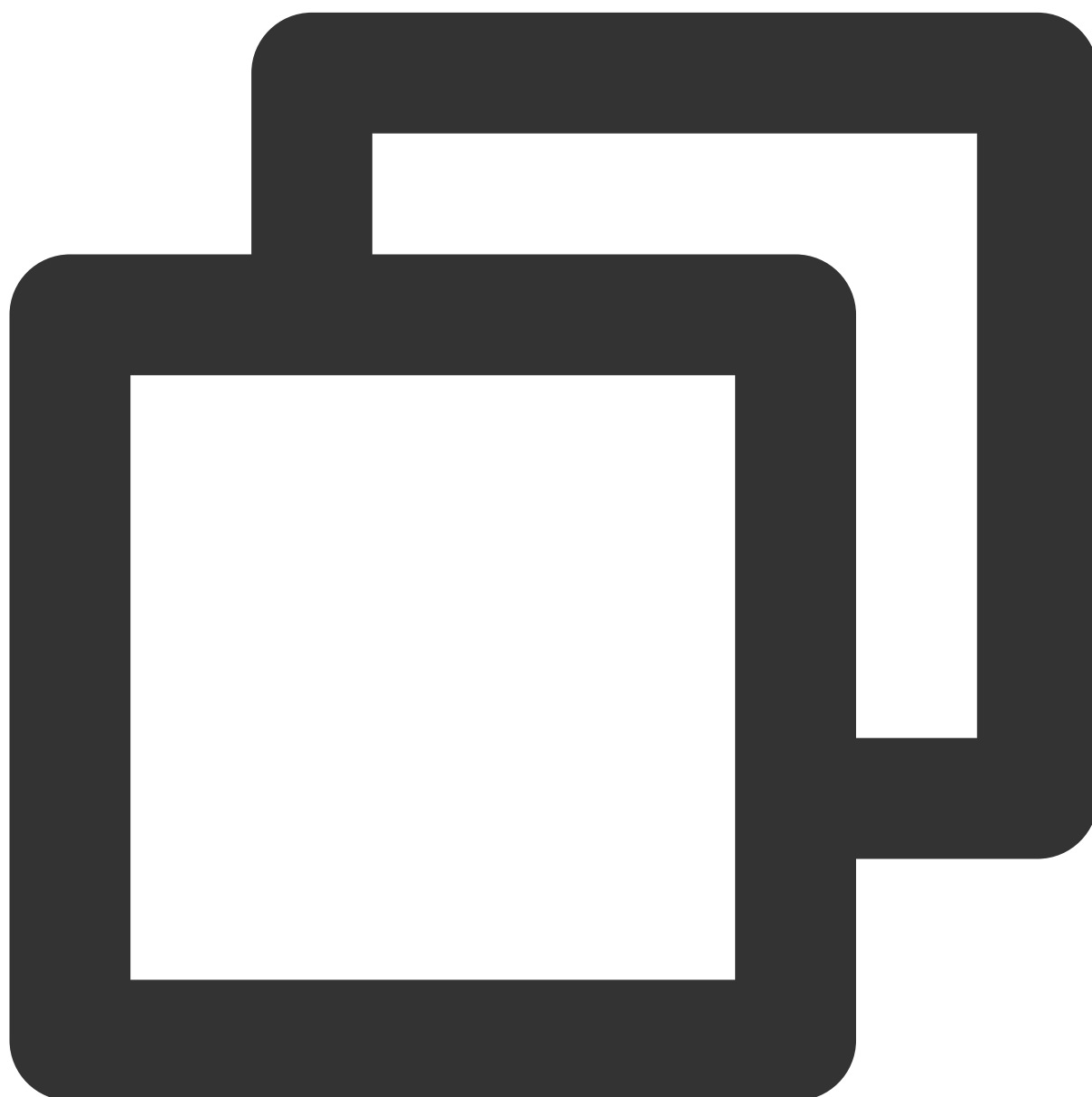
event_base_dispatch(evbase);
event_base_free(evbase);
```



Parameter	Description
exchange_name	Exchange name, which can be obtained from the exchange list in the console.
routing_key	The routing key supported by the message queue.

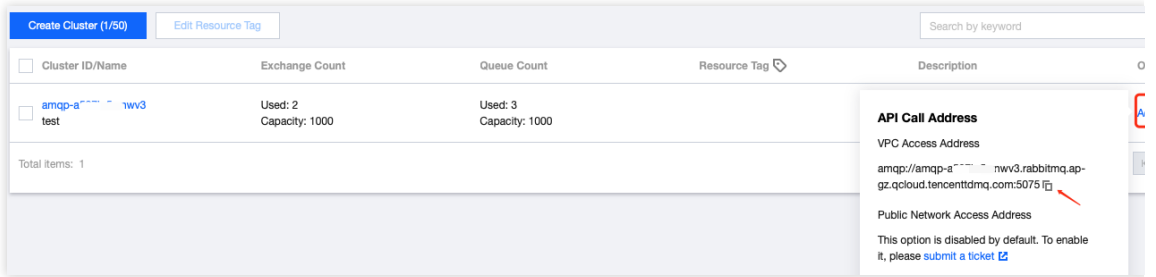
### Step 3. Consume messages

1. Establish a connection.

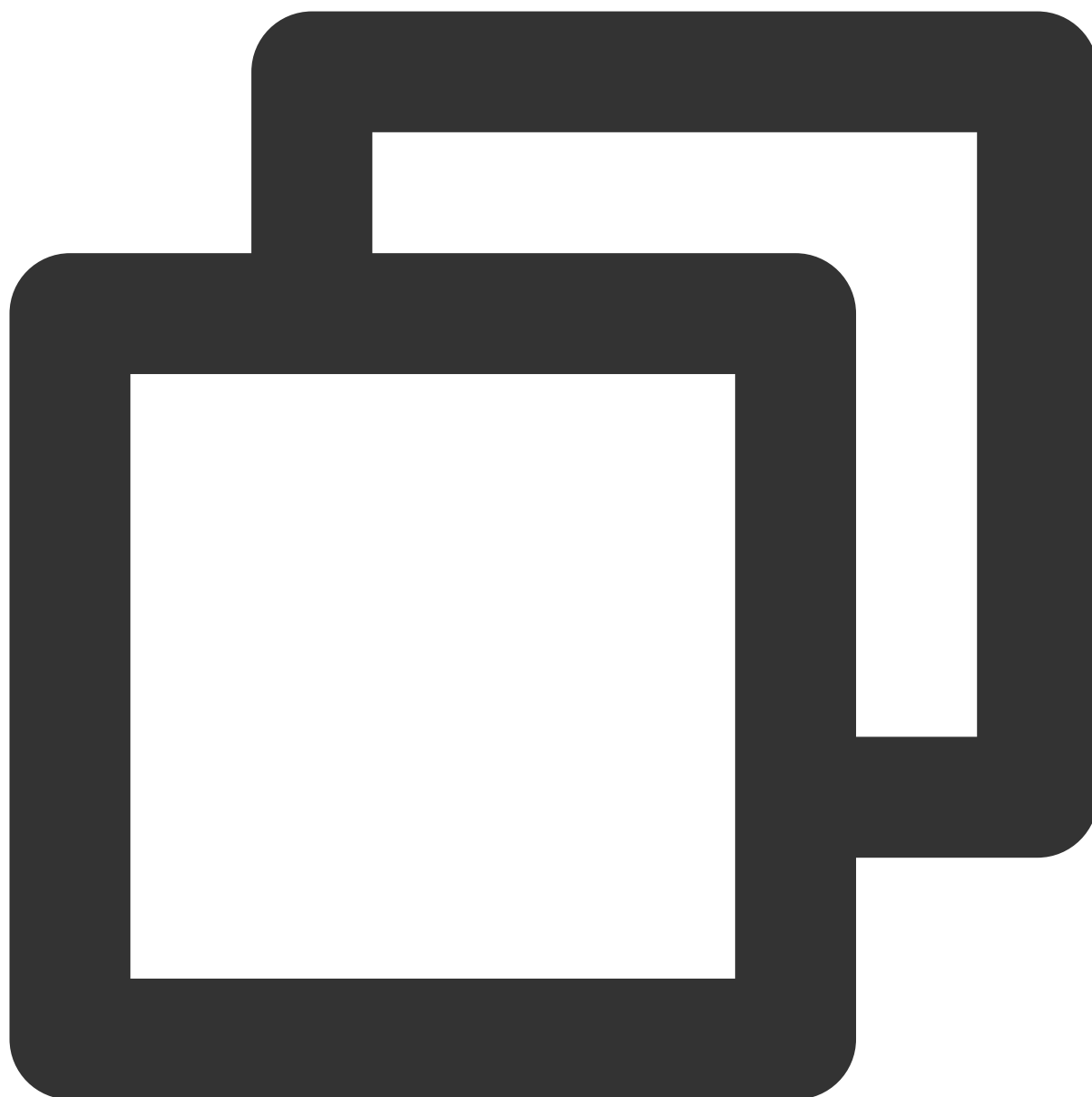


```
ConnHandler handler;
```

```
// Establish a connection
AMQP::TcpConnection connection(handler, AMQP::Address(host, part, AMQP::Login(usern
// Create a channel
AMQP::TcpChannel channel(&connection);
channel.onError([&handler](const char *message) {
    std::cout << "Channel error: " << message << std::endl;
    handler.Stop();
});
```

Parameter	Description
String	<p>Cluster access address, which can be obtained in the console by clicking Access Address in the “Operations” of the Cluster page.</p> 
port	Port number in the cluster access address.
name-server	Role name, which can be copied on the <a href="#">Role Management</a> page.
password	Role token, which can be copied in the Token column on the <a href="#">Role Management</a> page.
vhost	Vhost name in the format of “cluster ID +   + vhost name”.

2. Declare an exchange and a message queue and bind them.

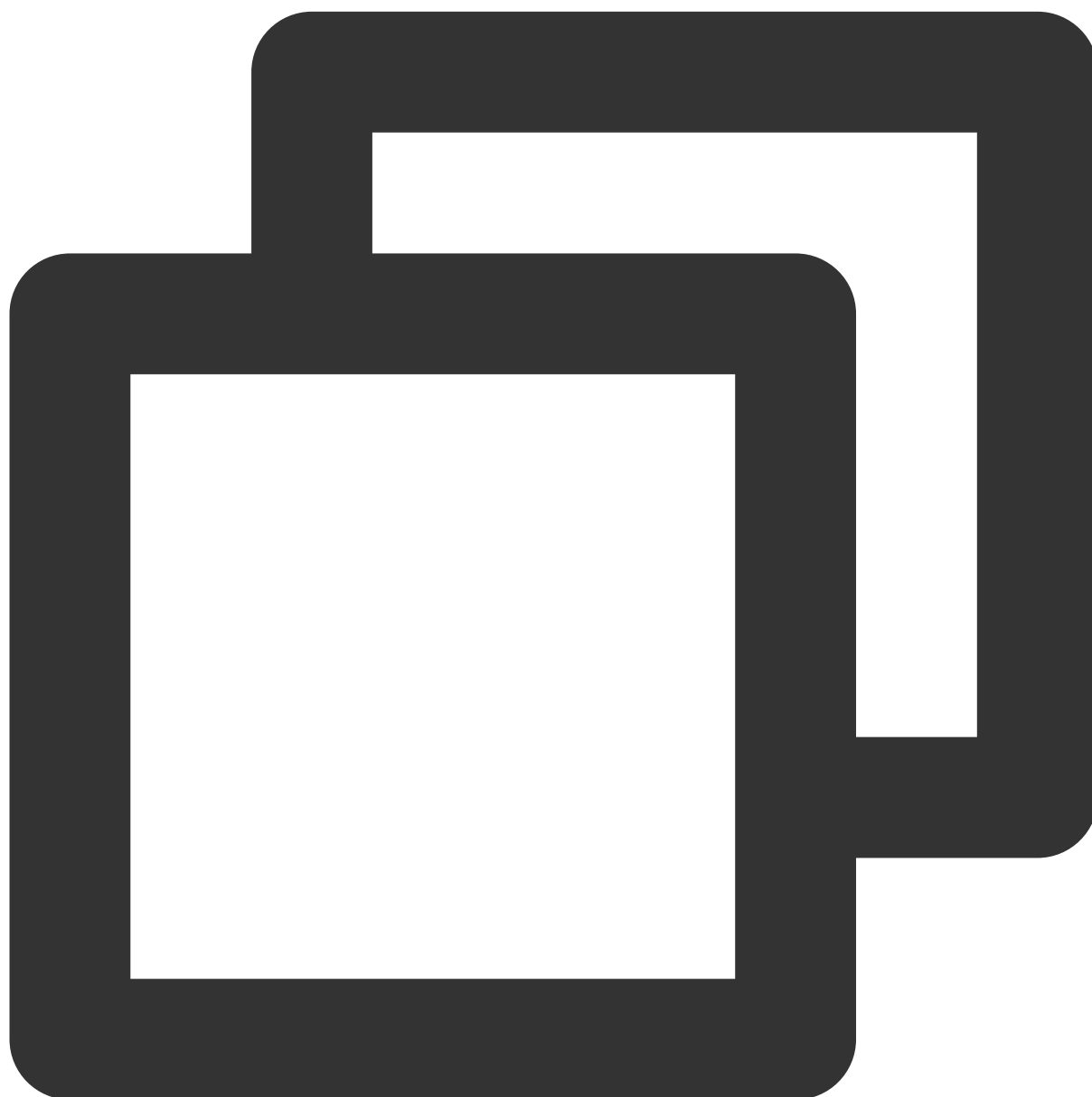


```
// Declare an exchange
channel.declareExchange(exchange_name, AMQP::ExchangeType::direct);
// Declare a message queue
channel.declareQueue(queue_name, AMQP::durable);
// Bind the message queue to the exchange
channel.bindQueue(exchange_name, queue_name, routing_key);
```

Parameter	Description

exchange_name	Exchange name, which can be obtained from the exchange list in the console.
queue_name	Message queue name, which can be obtained from the queue list in the console.
routing_key	The routing key supported by the message queue.

### 3. Subscribe to messages



```
// Subscribe to the message
channel.consume(queue_name)
    .onReceived
```

```
(
    [&channel](const AMQP::Message &msg, uint64_t tag, bool redelivered) {
        // Process the message
        std::cout << "Received: " << msg.body() << std::endl;
        // Acknowledge the message, or reject it if the consumption failed
        channel.ack(tag);
    }
);
handler.Start();
```

Parameter	Description
queue_name	Message queue name, which can be obtained from the queue list in the console.

**Note:**

For a complete sample or more information, see [Demo](#), [AMQP-CPP](#), or [AMQP-CPP Examples](#).