

# **TDMQ for RocketMQ**

## **RocketMQ 4.x**

### **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

## RocketMQ 4.x

### Product Introduction

Overview

Architecture

Concepts

Strengths

Use Cases

Use Limits

Comparison with Apache RocketMQ

### Purchase Guide

Billing Overview

Exclusive Cluster

Virtual Cluster

Product Series

Purchase Methods

Payment Overdue

Refund

Bill Description

### Getting Started

Overview

Messaging over TCP

Resource Creation and Preparation

Downloading and Running Demo

### Operation Guide

Cluster Management

Namespace Management

Topic Management

Group Management

Role and Authentication

Access Management (CAM)

Access Authorization for Root Account

Access Authorization for Sub-Accounts

Granting Sub-Account Access to TDMQ for RocketMQ

Granting Operation-Level Permissions to Sub-Accounts

Granting Resource-Level Permissions to Sub-Accounts

## Granting Tag-Level Permissions to Sub-Accounts

### Tag Management

- Managing Resource with Tag

- Editing Tag

### Monitoring and Alarms

- Cluster Monitoring

- Group Monitoring

- Topic Monitoring

- Configuring Alarm

### Message Query

- Querying Message

- Querying Dead Letter Message

- Message Trace Description

### Migration to Cloud

- Metadata Migration

- Metadata Migration from Huawei Cloud

- Metadata Migration from ONS 4.0

- Message Service Data Flow Migration

### Development Guide

#### Message Types

- General Message

- Scheduled Message and Delayed Message

- Sequential Message

- Transactional Message

#### Message Filtering

#### Consumption Mode

#### Message Retry

#### Dead Letter Queue

### SDK Documentation

#### Access over TCP

##### Spring Boot Starter

- Sending and Receiving General Messages

- Sending and Receiving Filtered Messages

- Sending and Receiving Delayed Messages

##### Spring Cloud Stream

##### SDK for Java

- Sending and Receiving General Messages

- Sending and Receiving Delayed Messages

Sending and Receiving Sequential Messages

Sending and Receiving Transactional Messages

Sending and Receiving Filtered Messages

Sending and Receiving Broadcast Messages

SDK for C++

SDK for Go

SDK for Python

Access over HTTP

Access over HTTP

SDK for Java

Sending and Receiving General Messages

Sending and Receiving Sequential Messages

# RocketMQ 4.x

## Product Introduction

### Overview

Last updated : 2023-03-01 10:33:06

TDMQ for RocketMQ is distributed message middleware developed by Tencent Cloud based on Apache RocketMQ. It is fully compatible with all the components and principles of Apache RocketMQ and supports connection to open-source RocketMQ clients without any modifications.

Featuring low latency, high performance, reliability, and scalability, and trillions of QPS, TDMQ for RocketMQ can add async decoupling and peak shifting capabilities to distributed application systems. It also provides the capabilities necessary to internet applications, such as massive message retention, high throughput, and reliable retry mechanism.

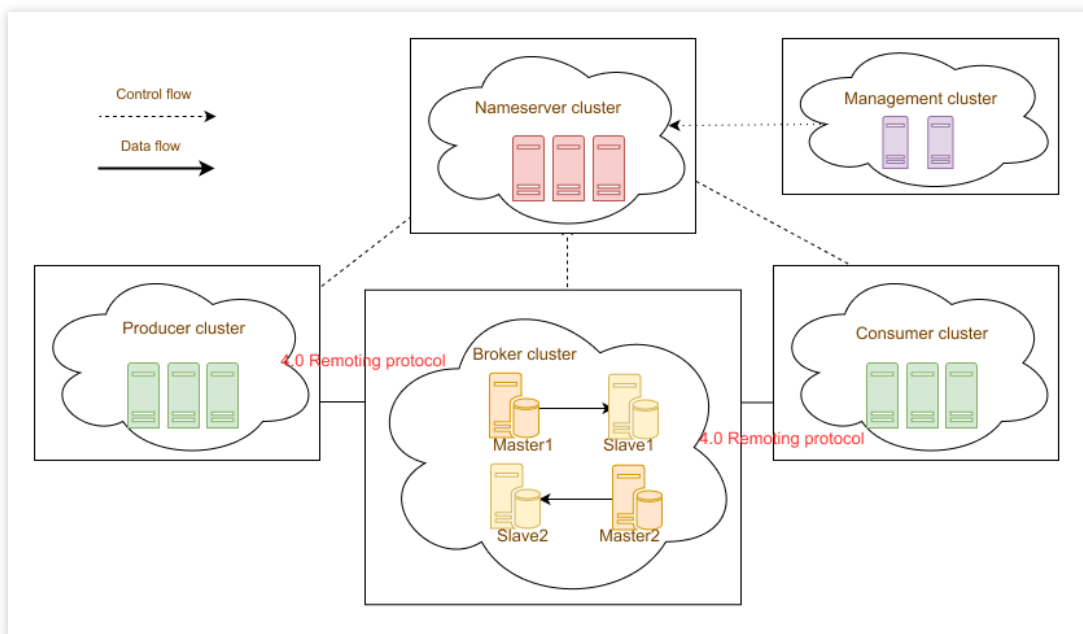
# Architecture

Last updated : 2023-04-14 16:50:11

This document describes the deployment architecture of TDMQ for RocketMQ to help you better understand its architectural principles.

## Deployment Architecture

The system deployment architecture of TDMQ for RocketMQ is shown in the following diagram:



The core concepts are as follows:

**Producer cluster:** Client-side application, which is responsible for producing and sending messages.

**Consumer cluster:** Client-side application, which is responsible for subscribing to and consuming messages.

**Nameserver cluster:** server-side application, which is responsible for address routing and broker heartbeat registration.

**Heartbeat registration:** Nameserver acts as the registration center. Machines in each role must regularly report their status to Nameserver. If a machine fails to report beyond the timeout period, Nameserver will consider it faulty and unavailable and remove it from the available list.

**Address routing:** Each Nameserver stores the entire routing information of the Broker cluster and the queue information used for client queries. Producers and consumers obtain the routing information of the entire Broker cluster through Nameserver to deliver and consume messages.

Broker cluster: Server application, which is responsible for receiving, storing, and delivering messages. It supports primary-secondary multi-copy mode where the deployment of secondary nodes is optional. The actual high reliability of data in the production environment on the public cloud directly depends on the three copies of the cloud disk.

Management cluster: Server application that is a visual management and control console. It is responsible for operating the entire cluster, such as source data sending/receiving and management.

For the advantages of TDMQ for RocketMQ over self-built open-source Apache RocketMQ, see [Comparison with Apache RocketMQ](#).



# Concepts

Last updated : 2023-10-19 10:38:05

This document lists the common concepts and their definitions in TDMQ for RocketMQ.

## Message ( `Message` )

A message is the physical carrier of information transmitted by the messaging system. It is the smallest unit of the produced or consumed data. A producer encapsulates the load and extended attributes of business data into messages and sends the messages to a TDMQ for RocketMQ broker. Then, the broker delivers the messages to the consumer based on the relevant semantics.

## Topic ( `Topic` )

A topic is the collection of a type of messages. It is the basic unit for message subscription in TDMQ for RocketMQ. Each topic contains several messages.

## Message Tag ( `MessageTag` )

Tags are used to categorize different types of messages in the same topic. Topic and tag are basically the first-level and second-level classifications of messages, respectively.

## Message Queue ( `MessageQueue` )

A message queue (also known as a message partition) is a physical entity for message storage, and a topic can contain multiple queues. Messages in a queue can only be consumed by one consumer rather than multiple consumers in one consumer group.

## Message Offset ( `MessageQueueOffset` )

Messages are stored in multiple queues of a specified topic based on the order in which they arrive at the TDMQ for RocketMQ broker. Each message has a unique coordinate of type `Long` in the queue, which is defined as the message offset.

## Consumption Offset ( `ConsumerOffset` )

A message is not removed from the queue immediately after it has been consumed by a consumer. TDMQ for RocketMQ will record the offset of the last consumed message based on each consumer group. Such an offset is defined as the consumption offset.

## Message Index ( `MessageKey` )

A message index is a message-oriented index property in TDMQ for RocketMQ. By setting the message index, you can quickly find the corresponding message content.

## Producer ( `Producer` )

A producer in TDMQ for RocketMQ is a functional messaging entity that creates messages and sends them to the broker. It is typically integrated into the business system to encapsulate data as messages and send them to the broker.

## Consumer ( `Consumer` )

A consumer is an entity that receives and processes messages in TDMQ for RocketMQ. It is usually integrated into the business system to obtain messages from TDMQ for RocketMQ brokers and convert the messages into information that can be perceived and processed by business logic.

## Group ( `Group` )

Groups include producer groups and consumer groups.

**Producer group:** It is the collection of the same type of producers that send the same type of messages with the same sending logic. If a producer sends transactional messages and crashes afterward, the broker will contact other producer instances in the producer group to commit or cancel the transaction.

**Consumer group:** It is the collection of the same type of consumers that consume the same type of messages with the same consumption logic. It can ensure load balancing and fault tolerance in the message consumption process.

Consumer instances in a consumer group must subscribe to the same topics.

## Message Type ( `MessageType` )

Messages are classified by message transmission characteristic for message type management and security verification. TDMQ for RocketMQ has four message types: general message, sequential message, transactional message, and scheduled/delayed message.

### General Message

The general message is a basic message type. After the produced general messages are delivered to a specified topic, they will be consumed by consumers that subscribe to this topic. A topic with general messages is sequence-insensitive. Therefore, you can use multiple topic partitions to improve message production and consumption efficiency. This approach performs best when dealing with high throughput.

### Sequential Message

In TDMQ for RocketMQ, the sequential message is an advanced message type. Sequential messages in a specified topic are published and consumed in a First In First Out (FIFO) manner, that is, the first produced messages are first consumed.

### Retry Letter Queue

A retry letter queue is designed to ensure that messages are consumed normally. When a message is consumed for the first time by a consumer but is not acknowledged, it will be placed in the retry letter queue and will be retried there until the maximum number of retries is reached. It will then be delivered to the dead letter queue.

In actual scenarios, messages may not be processed promptly due to temporary issues such as network jitter and service restart. The retry mechanism of the retry letter queue can be a good solution in this case.

### Dead Letter Queue

A dead letter queue is a special type of message queue used to centrally process messages that cannot be consumed normally. If a message cannot be consumed after a specified number of retries in the retry letter queue, TDMQ will determine that the message cannot be consumed under the current situation and deliver it to the dead letter queue.

In actual scenarios, messages may not be consumed due to service downtime or network disconnection. In this case, they will not be discarded immediately; instead, they will be persistently stored in the dead letter queue. After fixing the problem, you can create a consumer to subscribe to the dead letter queue to process such messages.

## Clustering Consumption

Clustering consumption: If the clustering consumption mode is used, each message only needs to be processed by any of the consumers in the cluster. This mode is suitable for scenarios where each message only needs to be processed once.

## Broadcasting Consumption

Broadcasting consumption: If the broadcasting consumption mode is used, each message will be pushed to all registered consumers in the cluster to ensure that the message is consumed by each consumer at least once. This mode is suitable for scenarios where each message needs to be processed by each consumer in the cluster.

## Message Filtering

A consumer can filter messages by subscribing to specified message tags to ensure that it only receives the filtered messages. The whole filtering process is completed in the TDMQ for RocketMQ broker.

## Consumption Offset Reset

Resetting the consumption offset means resetting a consumer group's consumption offset for subscribed topics within the persistent message storage period based on the time axis. After the offset is reset, the consumers will receive the messages that the producer sends to the TDMQ for RocketMQ broker after the set time point.

## Message Trace

The message trace records the entire lifecycle of a message from the time it is sent by the producer to the time it is received and processed by the consumer. With this feature, you can track the entire trace of a message, starting from its production by a producer, its storage and distribution within the TDMQ for RocketMQ broker, and finally its consumption by one or more consumers. This helps you troubleshoot any problems that may occur during message processing.

## Message Heap

Message heap occurs in scenarios where the producer has sent messages to the TDMQ for RocketMQ broker but the consumer fails to normally consume all these messages promptly due to its consumption capability limit. In this case, the unconsumed messages will be heaped in the broker. The message heap data is collected once every minute. After the message retention period (3 days by default) elapses, the unconsumed messages will no longer be heaped in the broker because they have been deleted by the broker.

# Strengths

Last updated : 2024-01-18 09:45:29

## Open-Source Version Compatibility

TDMQ for RocketMQ is compatible with open-source RocketMQ 4.3.0 and later. It supports access from open-source clients in Java, C, C++, Go, and other programming languages.

## Resource Isolation

TDMQ for RocketMQ offers a multi-level resource structure that allows for both namespace-based virtual isolation and cluster-level physical isolation, making it simple for you to enable namespace-level permission verification to distinguish clients in different environments.

## Rich Message Types

TDMQ for RocketMQ supports multiple message types such as general, sequential, delayed, and transactional messages. It also supports message retry and the dead letter mechanism, fully meeting the requirements in various business scenarios.

## High Performance

A single TDMQ for RocketMQ server can sustain a production/consumption throughput of up to 10,000 messages. With the distributed architecture and stateless services, the cluster can be scaled horizontally to increase the cluster throughput.

## Observability

TDMQ for RocketMQ supports various monitoring metrics in the console where message traces can be displayed. It also provides alarming capabilities and all the TencentCloud APIs you may need to integrate with your self-service Ops systems.

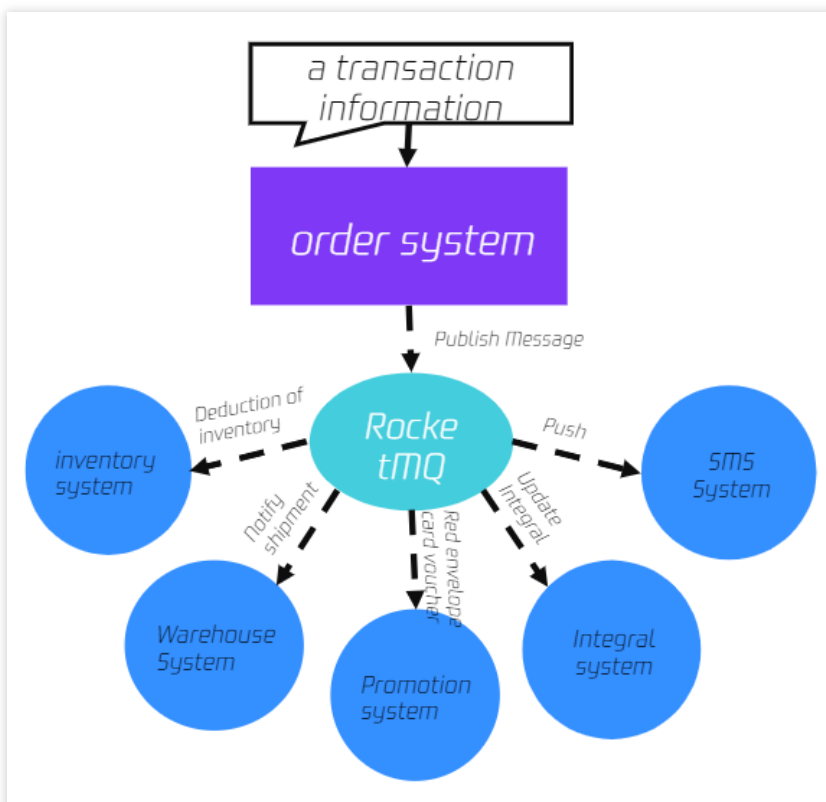
# Use Cases

Last updated : 2023-09-12 16:02:09

TDMQ for RocketMQ is a distributed message middleware based on Apache RocketMQ. It is applied to message communication between distributed systems or components. It has the characteristics of massive message heap, low latency, high throughput, high reliability, and strong transaction consistency, which meets the requirements of async decoupling, peak shifting, sequential sending and receiving, distributed transaction consistency, and log sync.

## Async Decoupling

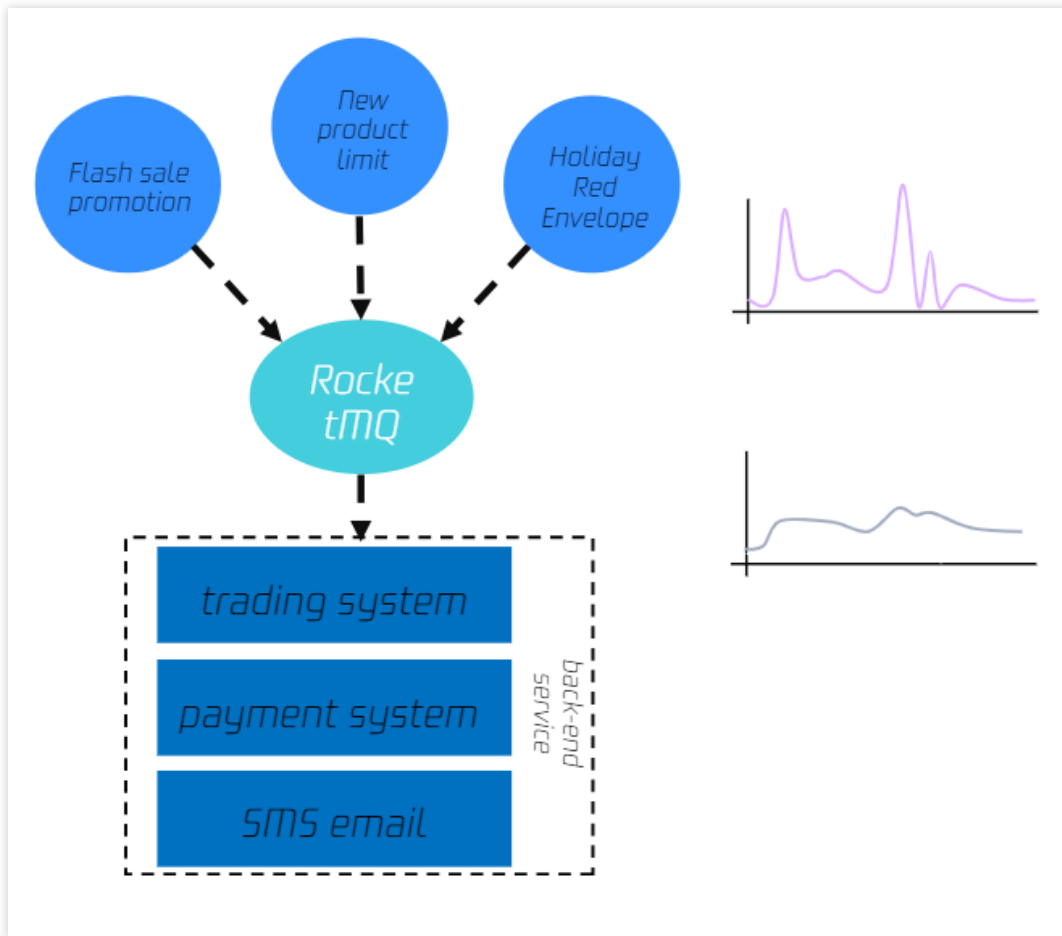
The transaction engine is the core system of Tencent billing. The data of each transaction order needs to be monitored by dozens of downstream business systems, including inventory system, warehousing system, promotion system, and points system. Such systems use different message processing logic, making it impossible for a single system to adapt to all associated business. In this case, TDMQ for RocketMQ can decouple the coupling between multiple business systems to reduce the impact between systems and improve the response speed and robustness of core business.



## Peak Shifting

Companies hold promotional campaigns such as new product launch and festival red packet grabbing from time to time, which often cause temporary traffic spikes and pose huge challenges to each backend application system. In this case, TDMQ for RocketMQ can withstand spikes in traffic. It heaps up messages during peak periods and consumes

them in the downstream during off-peak periods, which balances the processing capacities of upstream and downstream systems and improve system availability.



## Subscription Notifications

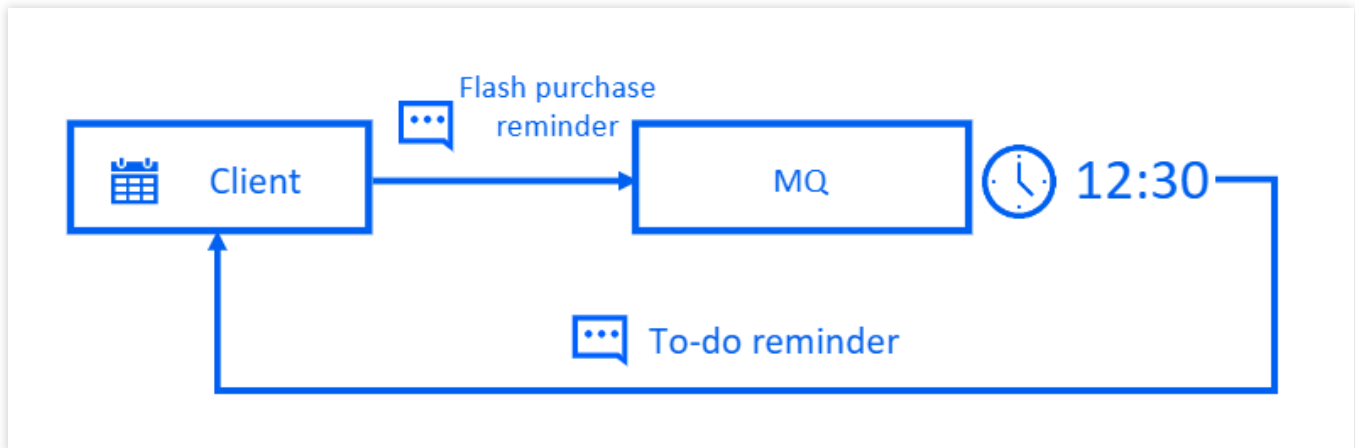
TDMQ for RocketMQ provides scheduled and delayed messages to can meet the ecommerce subscription notification scenarios.

**Scheduled message:** After a message is sent to the server, the business may want the consumer to receive it at a later time point rather than immediately. This type of message is called "scheduled message".

**Delayed message:** After a message is sent to the server, the business may want the consumer to receive it after a period of time rather than immediately. This type of message is called "delayed message".

For details about scheduled and delayed messages, see [Scheduled Message and Delayed Message](#).





## Consistency of Distributed Transactions

TDMQ for RocketMQ provides distributed transactional messages to loosely couple applications. Reliable transmission and multi-replica technology can ensure that messages are not lost, and the At-Least-Once feature ensures eventual data consistency.

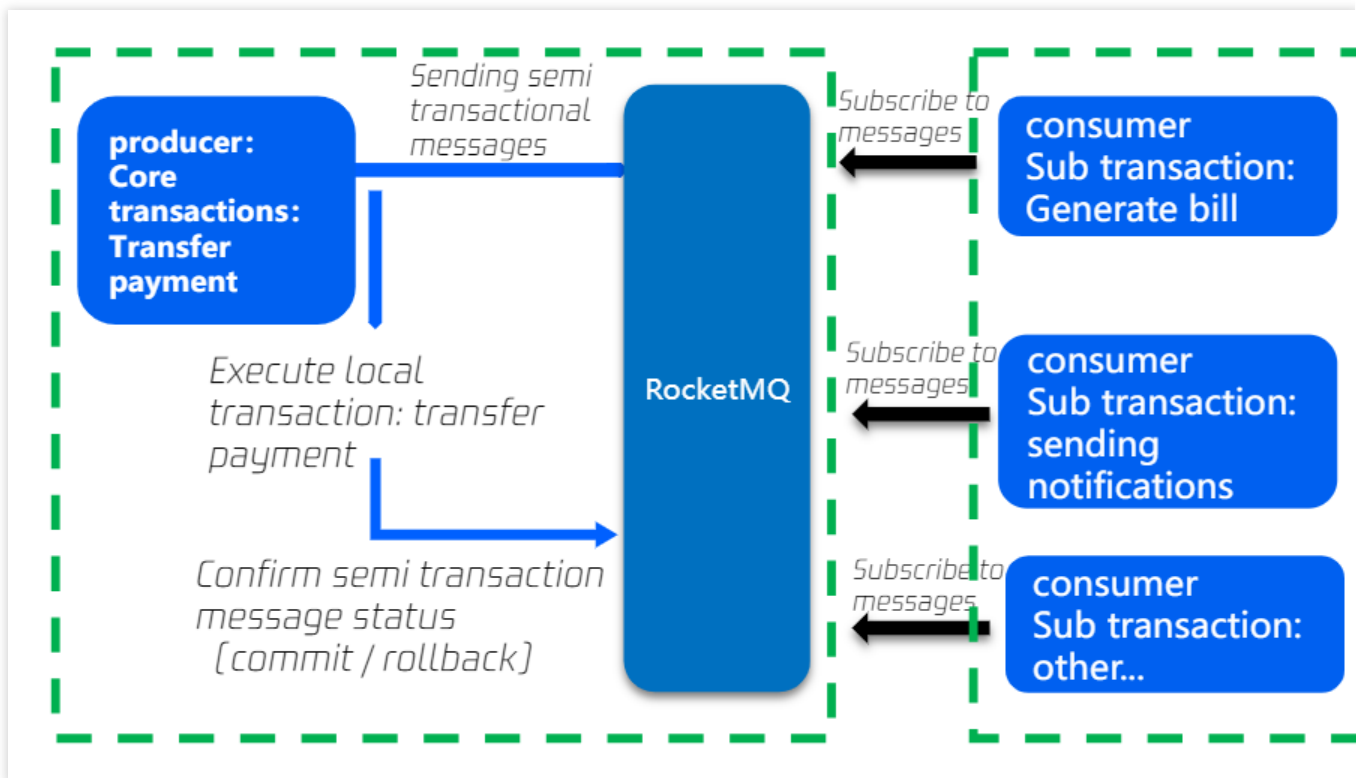
As a producer, the payment system forms a transaction with the message queue to ensure the consistency of local transactions and message sending.

Downstream business systems (bills, notifications, others) work as consumers to process in parallel.

Messages support reliable retries to ensure eventual data consistency.

The transaction messages of TDMQ for RocketMQ can be used to process transactions, which can greatly improve processing efficiency and performance. A billing system often has a long transaction linkage with a significant chance of error or timeout. TDMQ's automated repush and abundant message retention features can be used to provide transaction compensation, and the eventual consistency of payment tips notifications and transaction pushes can also be achieved through TDMQ for RocketMQ.

For details about transactional messages, see [Transactional Message](#).



## Sequential Message Sending/Receiving

Sequential message is an advanced message type provided by TDMQ for RocketMQ. For a specified topic, messages are published and consumed in strict accordance with the principle of First-In-First-Out (FIFO), that is, messages sent first are consumed first, and messages sent later are consumed later. Sequential messages are often used in the following business scenarios:

**Order creation:** In some ecommerce systems, an order's creation, payment, refund, and logistics messages must be produced or consumed in strict sequence, otherwise the order status will be messed up during consumption, which will affect the normal operation of the business. Therefore, the messages of this order must be produced and consumed in a certain sequence in the client and message queue. At the same time, the messages are sequentially dependent, and the processing of the next message must be dependent on the processing result of the preceding message.

**Log sync:** In the scenario of sequential event processing or real-time incremental data sync, sequential messages can also play a greater role. For example, it is necessary to ensure that database operations are in sequence when MySQL binlogs are synced.

**Financial scenarios:** In some matchmaking transaction scenarios like certain securities transactions, the first bidder is given priority in the case of the same bidding price, so it is necessary to produce and consume sequential messages in a FIFO manner.

For details about sequential messages, see [Sequential Message](#).

## Distributed Cache Sync

During sales and promotions, there are a wide variety of products with frequent price changes. When users query item prices multiple times, the cache server's network interface may be fully loaded, which makes page opening slower. After the broadcast consumption mode of TDMQ for RocketMQ is adopted, a message will be consumed by all nodes once, which is equivalent to syncing the price information to each server as needed in place of the cache.

# Use Limits

Last updated : 2023-09-12 17:53:17

This document lists the limits of certain metrics and performance in TDMQ for RocketMQ. Be careful not to exceed the limits during use so as to avoid exceptions.

## Cluster

Limit	Virtual Cluster	Exclusive Cluster
Maximum number of clusters per region	10	Unlimited
Cluster name length	3-64 characters	3-64 characters
Maximum TPS	4000	Above 4,000, subject to the node specification
Maximum bandwidth (production + consumption) per cluster	40 Mbps	Above 80 Mbps, subject to the node specification

## Namespace

Limit	Virtual Cluster	Exclusive Cluster
Maximum number of namespaces per cluster	10	10
Namespace name length	3-32 characters	3-32 characters

## Topic

Limit	Virtual Cluster	Exclusive Cluster
Maximum number of topics per cluster	150	200-500, subject to the node specification
Topic name length	3-64 characters	3-64 characters
Maximum number of producers per topic	1000	1000
Maximum number of consumers per topic	500	500

## Group

Limit	Virtual Cluster	Exclusive Cluster
Maximum number of groups per cluster	1500	2000-5000, subject to the node specification
Group name length	3-64 characters	3-64 characters

## Message

Limit	Virtual Cluster	Exclusive Cluster
Maximum message retention period	3 days	3 days
Maximum message delay	40 days	40 days
Message size	4 MB	4 MB
Consumption offset reset	3 days	3 days

# Comparison with Apache RocketMQ

Last updated : 2024-01-18 09:53:49

The performance comparison between TDMQ for RocketMQ and Apache RocketMQ is detailed below:

Feature Type	Feature	TDMQ for RocketMQ	Apache RocketMQ
Basic features	Scheduled message	The scheduled time is accurate down to the second and can be customized.	You can only specify the delay level.
	Visual management	Visual management for clusters, topics, and groups is supported. You can view the details of subscriptions and consumer status.	Visual management is supported but is less user-friendly. The console doesn't distinguish between topic types.
Availability	Elastic scaling	You don't need to manually deploy, configure, or scale up underlying computing resources because operations such as node registration are automatically performed in a visual manner. You can expand the number of nodes horizontally, increase the disk capacity, and upgrade the configurations of a single node vertically as needed at any time.	A self-built Ops team is required, and operations are performed in a less automatic or visualized manner.
	High reliability	With three data replicas, the server can be automatically restarted in seconds after the downtime, without affecting the message capacity and data.	Data can be replicated in sync or async mode. You need to design the deployment scheme and related parameters. The primary sync schemes won't be automatically used after the failover.
	Cross-AZ high-availability deployment	This feature is supported to avoid losses caused by data center-level failures.	This feature is supported, but it is time-consuming for you to design the deployment schemes and parameters.
Observability	Resource dashboard	You can monitor core metrics at a fine granularity and view production and consumption details.	This feature is supported but with fewer monitoring metrics.

	Alarming	With the capabilities provided by Cloud Monitor, alarms will be triggered in case of message heap or delayed message sending/receiving.	Not supported
Security management and control	Tenant namespace isolation	You can implement this feature in the console in a visual manner.	This feature is not supported. Namespaces cannot be truly isolated due to bugs.
	Root account and sub-account management	Supports authorization between Tencent Cloud CAM root accounts and sub-accounts and between enterprise accounts.	Not supported
Migration tool	Tool for migrating from Apache RocketMQ	You can easily migrate from Apache RocketMQ to TDMQ for RocketMQ by using scripts.	-

# Purchase Guide

## Billing Overview

### Exclusive Cluster

Last updated : 2023-10-19 10:42:05

This document describes the billing mode and billable items of TDMQ for RocketMQ exclusive cluster.

## Available Regions

Exclusive cluster is currently supported in the following regions:

Region	Code
Guangzhou	ap-guangzhou
Shanghai	ap-shanghai
Nanjing	ap-nanjing
Beijing	ap-beijing
Singapore	ap-singapore
Virginia	na-ashburn
Silicon Valley	na-siliconvalley
Hong Kong (China)	ap-hongkong

If you want to use the service in other regions, [submit a ticket](#) for application.

## Billing Mode

Item	Billing Mode	Description
Cluster instance	Monthly subscription – prepaid	When you purchase an exclusive cluster, the system will calculate the fees based on the cluster specification and service duration you select. You need to make upfront payment before using the



		cluster. This billing mode is suitable for long-term services with a relatively stable business traffic peak.
Public network bandwidth	Bill-by-hour bandwidth – postpaid	You are charged hourly for your actual public network bandwidth usage duration on a pay-as-you-go basis. This billing mode is suitable for short-term services with a relatively stable business traffic peak.

## Cluster Instance Pricing

### Billable items

TDMQ for RocketMQ exclusive cluster fees are calculated as follows:

Cluster price = minimum configuration price + compute configuration price + storage configuration price = minimum configuration price + node unit price *number of nodes* + storage unit price *storage size*.

Billable Item	Description
Minimum configuration	It refers to the fixed fees of a new cluster for basic services such as cluster management, message management, observability, and high availability, which will not increase as the cluster size increases.
Compute configuration	It refers to the compute service fees. Various node specifications are provided on demand. The price of a single node of each specification is fixed, and there are restrictions on the numbers of minimum and maximum nodes. The compute configuration price changes linearly based on the number of nodes.
Storage configuration	It refers to the storage service fees. You can customize the storage space in increments of 100 GB. The storage configuration price changes linearly based on the storage size.

### Pricing

The specific price is as displayed on the [purchase page](#). This section describes the performance differences between exclusive cluster specifications.

#### Note

If the number of nodes in current specifications cannot meet your business requirements, [submit a ticket](#) for assistance.

### Minimum configuration

The performance of the following specification is a reference value. The actual performance won't be any worse given the data measured during stress tests. Unexpected elastic capacity outside the specification won't cost anything either.

The calculation rules are the same for TPS and the number of API calls of virtual clusters.

Message type: TDMQ for RocketMQ has four message types: general message, scheduled and delayed message, transactional message, and sequential message. Except general messages, the other three message types are advanced messages.

Sending or consuming one general message is counted as 1 TPS, while sending or consuming one advanced message (such as a delayed or transactional message) is counted as 5 TPS. For example, if a topic sends two transactional messages once and consumes one transactional message, the TPS will be 15 ( $2 \times 5 + 1 \times 5 = 15$ ).

Message size: The maximum size of a single message is 4 MB, with 4 KB as the unit of measurement. A size of less than 4 KB is calculated as 4 KB.

Specification Type	Single-Node Specification
Basic	TPS (production + consumption): 2,000
Standard	TPS (production + consumption): 5,000
Advanced I	TPS (production + consumption): 10,000
Advanced II	TPS (production + consumption): 18,000

### Compute configuration

Compute configuration fees = node price \* number of nodes

Cluster performance description: The **cluster performance is equal to node performance \* number of nodes** and it changes linearly within the node range.

Specification Type	Minimum Nodes	Maximum Nodes	Minimum Single-Node TPS (in units of 4 KB)
Basic	2	10	2,000
Standard	2	10	5,000
Advanced I	2	20	10,000
Advanced II	2	20	18,000

### Storage configuration

Storage fees = storage space \* storage unit price.

Each exclusive cluster version provides a minimum storage of 200 GB for a single node. You can choose a higher storage space in increments of 100 GB based on your business needs.

## Public Network Bandwidth Price

Fees are charged hourly by the actual number of usage hours based on the public network bandwidth in Mbps.

Region	Price (USD/Mbps/Hour)	
	1-5 Mbps	6 Mbps or above
Guangzhou, Shanghai, Nanjing, Beijing	0.04	0.14
Singapore	0.035	0.12

# Virtual Cluster

Last updated : 2023-09-12 16:05:18

TDMQ for RocketMQ has ended its public beta of virtual clusters on December 28, 2022 and will start billing for such clusters. This document describes the billing mode and billable items of a TDMQ for RocketMQ virtual cluster.

## Available Regions

Virtual cluster is currently supported in the following regions:

Region	Value
Guangzhou	ap-guangzhou
Shanghai	ap-shanghai
Shanghai Finance	ap-shanghai-fsi
Beijing	ap-beijing
Nanjing	ap-nanjing
Beijing Finance	ap-beijing-fsi
Hong Kong (China)	ap-hongkong
Singapore	ap-singapore
Silicon Valley	na-siliconvalley
Frankfurt	eu-frankfurt
Seoul	ap-seoul
Mumbai	ap-mumbai
Virginia	na-ashburn
Jakarta	ap-jakarta

## Billing Mode

The billing method of a TDMQ for RocketMQ virtual cluster is **pay-as-you-go (postpaid)**. Pay-as-you-go is a payment method based on the actual usage of the resource specifications you purchased, which is suitable for testing or scenarios with unpredictable traffic peaks. You can use resources before making payment, and the fee is settled on every clock-hour.

## Billable Items

TDMQ for RocketMQ virtual clusters are sold in the form of clusters, and the billing formula in the pay-as-you-go mode is as follows:

Total fee = API calls fee + topic usage fee = (number of API calls for sending messages + number of API calls for consuming messages) x unit price of API calls + number of topics x number of days x unit price of topic.

Billable Items	Billing Rules
API calls fee	<p>The calculation rules for the number of API calls are based on message type and message size.</p> <p>Message type: TDMQ for RocketMQ has four message types: general message, scheduled and delayed message, transactional message, and sequential message. Except general messages, the other three message types are advanced messages.</p> <p>General message: Sending or consuming one general message is counted as one API call regardless of whether it is successfully sent or consumed, and the API call will be billed once initiated.</p> <p>Advanced messages: Sending or consuming one advanced message is counted as five API calls. For example, if a topic sends 2 transactional messages once and consumes 1 transactional message, the number of API calls is calculated as: <math>2 \times 5 + 1 \times 5 = 15</math> calls.</p> <p>Message size:</p> <p>The maximum size of a single message is 4 MB, with 4 KB as the unit of measurement. A message that is less than 4 KB is calculated to be 4 KB. For example, the request of an 18 KB message will be billed as <math>\lceil 18/4 \rceil = 5</math> API calls. <b><math>\lceil \rceil</math> means rounding up to the nearest integer.</b></p>
Topic usage fee	<p>The topic unit price will change on a tier basis based on the number of API calls generated by each topic on the day. Each topic will be charged a resource usage fee every day. The daily topic resource usage fee is the sum of the fees generated by all topics on the day. <b>No matter whether the topic has sent or received messages on the day, it will be billed once.</b></p>

## Pricing

## Free Tier

Each root account has a monthly quota of 20 million free API calls.

### Note

The free quota is temporarily provided during the new product promotion period. Before the official billing, you will be notified by announcements, SMS, Message Center, and emails.

## API Calls Fee

Pricing Tier	Number of Calls (Billion Calls/Month)	Unit Price (USD/Million Calls) for Public Cloud		
		Regions in Chinese Mainland	Regions Outside Chinese Mainland	Finance Zone
First tier	0-10	0.26	0.33	0.41
Second tier	10-50	0.21	0.13	0.29
Third tier	50-500	0.17	0.23	0.29
Fourth tier	> 500	0.14	0.19	0.23

### Note

The above tiers are based on the Tencent Cloud account (UIN) and the cumulative number of API calls by billing period (monthly).

### Billing example

Suppose your instance resides in Guangzhou region, the daily message sending and receiving is as follows:

50 million general messages are produced, the number of message consumption is 70 million times (including the number of retries upon message delivery failures), and the size of each message is 20 KB.

30 million transactional messages are produced, the number of message consumption is 30 million times, and the size of each message is 4 KB.

10 million delayed messages are produced, the number of message consumption is 10 million times, and the size of each message is 2 KB.

Then, the number of API calls generated on the day is calculated as:  $(50 \text{ million} + 70 \text{ million}) \times \lceil 20/4 \rceil + (30 \text{ million} + 30 \text{ million}) \times 5 \times \lceil 4/4 \rceil + (10 \text{ million} + 10 \text{ million}) \times 5 \times \lceil 2/4 \rceil = 1 \text{ billion calls}$ .

If 1 billion API calls are generated every day in September, the incurred API call fees are as follows:

On September 1, the number of API calls is 1 billion, and the cumulative number of API calls in the month is 1 billion, which falls within the first tier. The unit price is 0.26 USD, and the fee charged is  $10 \times 0.26 \times 100 = 260$  US dollars (100 means 101 million100 (i.e., 1 billion), the unit price is charged per million calls).

On September 2, the number of API calls is 1 billion, and the cumulative number of API calls in the month is 2 billion, which falls within the second tier. The unit price is 0.21 US dollars, and the fee charged is  $10 \times 0.21 \times 100 = 210$  US dollars (100 means 101 million100 (i.e., 1 billion), the unit price is charged per million calls).

On September 3, the number of API calls is 1 billion, and the cumulative number of API calls in the month is 3 billion, which falls within the second tier. The unit price is 0.21 USD, and the fee charged is  $10 \times 0.21 \times 100 = 210$  US dollars (100 means 101 million100 (i.e., 1 billion), the unit price is charged per million calls).

September 4 and the remaining September days are calculated in a similar manner.

On October 1, the cumulative number of API calls will start from 0.

## Topic Usage Fee

The topic unit price will change on a tier basis based on the number of API calls generated by each topic on the day. The daily topic resource usage fee is the sum of the fees generated by all topics on the day. A topic created less than one day will still be charged because it is considered to have existed for a day.

### Note

Since the billing cycle is calculated from 0 o'clock, if you delete the topic resources on a day, the resource usage fee will still be charged on the day, so the fee will still be displayed in the bill of next day; but no fees will be charged after that.

Pricing Tier	Number of Calls (10,000 Times/Topic/Day)	Unit Price for Public Cloud (USD/Topic/Day)		
		Regions in Chinese Mainland	Regions Outside Chinese Mainland	Finance Zone
First tier	0-100	0.26	0.33	0.41
Second tier	100-1000	0.13	0.17	0.21
Third tier	> 1000	0	0	0

### Billing example

Suppose your instance resides in Guangzhou region, and you have created a total of three topics.

Topic 1 has 200,000 API calls on a day, which falls within the first tier, and the topic usage fee will be 0.26 USD on the day.

Topic 2 has 2 million API calls on the same day, which falls within the second tier, and the topic usage fee will be 0.13 USD on the day.

Topic 3 has 500,000 API calls on the same day, which falls within the first tier, and the topic usage fee will be 0.26 USD on the day.

Then, the sum of topic usage fees charged on the day =  $0.26 + 0.13 + 0.26 = 0.65$  USD.



# Product Series

Last updated : 2023-09-12 16:07:02

TDMQ for RocketMQ is available in forms of exclusive cluster and virtual cluster as compared below:

Feature	Exclusive Cluster	Virtual Cluster
Version compatibility	Compatible with open-source version 4.9 and earlier	Compatible with open-source version 4.9 and earlier
Instance type	Physical isolation of resources	Logical isolation of resources, where underlying physical resources are shared.
Billing mode	<b>Monthly subscription</b> as priced on the <a href="#">purchase page</a>	<b>Pay-as-you-go</b> as priced on the <a href="#">purchase page</a>
TPS range	On-demand purchase based on different node specifications	Suitable for TPS below 4000
Scaling	Flexible scaling, where the number of nodes, node specifications (coming soon), and storage space can be expanded separately.	Not supported
Broker repair and upgrade time	Few time required for upgrade.	Longer time required for upgrade as it is subject to shared cluster resources.
Availability	99.99%	99.95%
High availability	Custom multi-AZ deployment in the same region is supported to improve the disaster recovery capabilities.	Multi-AZ deployment in the same region is not supported.
Technical support	Parameter optimization consulting services are supported, helping you customize parameter configurations for special business scenarios. You can <a href="#">submit a ticket</a> for application.	Basic troubleshooting and bug fixing services are supported.
Event support	Event support is provided for major events such as product upgrade, business launch, and promotion campaign to ensure smooth business operations.	Not supported

# Purchase Methods

Last updated : 2024-01-18 10:02:30

TDMQ for RocketMQ exclusive clusters are **monthly subscribed (prepaid)**. You can purchase a cluster in the following steps:

1. Log in to the [TDMQ console](#).
2. Select **RocketMQ > Cluster** on the left sidebar and click **Create** to enter the purchase page.
3. On the purchase page, select the region, AZ, cluster type, and cluster specification.
4. Click **Buy Now** and make the payment as prompted.

# Payment Overdue

Last updated : 2023-07-21 15:19:27

## Note :

If you are a customer of a Tencent Cloud partner, the rules regarding resources when there are overdue payments are subject to the agreement between you and the partner.

## Notes

When you no longer use clusters, terminate them as soon as possible to avoid further fee deductions.

After clusters are terminated or repossessed, their data will be deleted and cannot be recovered.

## Pay-as-You-Go

TDMQ for RocketMQ virtual cluster is pay-as-you-go daily; that is, the billing system measures and issues a bill for your service usage for a calendar day on the next calendar day and deducts the service fees from your account accordingly.

If your account balance is insufficient, but the current usage is within the free tier, you can continue to use the service.

If your account balance becomes insufficient and your account isn't eligible for the non-stop feature, you can continue to use TDMQ for RocketMQ for 24 hours, and we will continue to bill you for this period. After 24 hours, the TDMQ for RocketMQ service will be stopped, you cannot send/receive messages or use the console and TencentCloud API, but resource usage fees will still be incurred.

After the service is stopped, the system will process TDMQ for RocketMQ as follows:

Time After Service Suspension	Description
≤ 7 days	If your account is topped up to a positive balance, the billing will continue, and you can restart TDMQ for RocketMQ.
	If your account balance remains negative, TDMQ for RocketMQ cannot be restarted.
> 7 days	If your account is not topped up to a positive balance, your pay-as-you-go TDMQ for RocketMQ resources will be terminated. All data will be deleted and cannot be recovered. When your resources are terminated, your Tencent Cloud account creator and all collaborators will be notified by email and SMS.

# Monthly Subscription

TDMQ for RocketMQ exclusive cluster is monthly subscribed.

## Expiration alert

Seven days before your monthly subscribed cluster expires, the system automatically pushes an expiration alert message to you every other day. All alert messages are sent to the Tencent Cloud account creator and all collaborators by **email and SMS**.

## Overdue payment reminder

From the day when your monthly subscribed cluster expires, the system pushes an alert message of resource isolation due to overdue payment to you every other day. All alert messages are sent to the Tencent Cloud account creator and all collaborators by **email and SMS**.

## Overdue payment policy

If your account balance is sufficient and you previously enabled auto-renewal, the resource will be automatically renewed on the expiration date.

Your cluster can be used normally within 7 days after expiration. If you renew it during this period, **the billing cycle of the renewed cluster will start from the expiration date of the previous cycle**.

If you don't renew your cluster within 7 days after expiration, your cluster service will be suspended, and resources will be terminated. All data will be deleted and cannot be recovered. When your resources are terminated, your Tencent Cloud account creator and all collaborators will be notified by email and SMS.

## Notes

Once you receive an overdue payment reminder, top up your account in the [console](#) as soon as possible to prevent your business from being affected.

If you have any questions about bills, check your bill details on the [Resource Bills](#) page in the console.

If you have any questions about fees, see [Purchase Guide](#) for the description of each billable item and billing rules.

You can also configure alarms for overdue payments through the balance alert feature in the Billing Center. For more information, see [Balance Alerts](#).

# Refund

Last updated : 2023-03-31 11:44:53

## Pay-as-You-Go

Pay-as-you-go clusters can be terminated at any time, and then the billing will stop.

## Monthly Subscription

### Refund policy

Unit prices and discounts are subject to the current system offers.

If the policy of the campaign where the product was purchased conflicts with the refund policy, the former shall prevail.

If the campaign policy denies refunds, no refund can be made.

Currently, self-service refund is unavailable for orders placed from promotion rewarding channels. You can submit a ticket to apply for a refund.

Tencent Cloud has the right to reject any suspected abnormal or malicious application for return.

### Refund amount and method

#### Refund amount = paid order amount - consumed resource amount

Such amounts are calculated based on the **usage duration**:

Consumed resource amount = (usage duration / total amount) *original order price* current discount

### Notes

A usage duration less than 1 day will be calculated as 1 day, and the current system discount matching the usage duration applies.

# Bill Description

Last updated : 2023-04-12 11:20:50

## Overview

If you have any doubts about the fee deduction of TDMQ for RocketMQ, you can go to the billing center to view the consumption details.

## Directions

1. Log in to the [TDMQ console](#).
2. On the top right of the page, hover over **Expense**, and click **Bills** in the drop-down box to enter Billing Center.
3. On the **Bills > Bill Overview** page, you can view the consumption overview of all products under your account.

**2023-4 Bill Summary** (Unit: USD)

Total Cost (Including Tax): 16.47USD

Product Name	Total Amount After Discount (Excluding Tax)	Voucher Deduction	Tax Amount	Total Cost
TencentDB for MongoDB	11.86 USD	0.00 USD	0.00 USD	
Tencent Distributed Message Queue	3.42 USD	0.00 USD	0.00 USD	

4. On the **Bills > Bill Details** page, you can view the consumption records of each product unit under your account within the billing cycle.

**Note:**

The bill by instance is issued on the 1st day of the next month. As there may be a delay in the data of bill by instance, the query results are for reference only. You need to check the detailed bill for real-time deduction data.

Bill by instance: Select the **Bill by Instance** tab on the bill details page. Taking TDMQ for RocketMQ as an example, select **TDMQ** for the product and **TDMQ for RocketMQ** for the subproduct, and then you can view the consumption details of each instance of TDMQ for RocketMQ.

The screenshot shows the Tencent Cloud Billing Center interface. The left sidebar contains the 'Billing Center' menu with options like Account Info, Order Management, Renewal Management, Reserved Instance, Payment Management, Bills, and Consumption Bill. The 'Bills' section is expanded, and 'Bill Details' is selected. The main content area shows 'Bill Details' for the month of 2023-03. A 'Bill by Instance' tab is active. A blue information banner states: 'Expense figures in Bill Details are accurate up to 8 decimal places. Expense figures in Bill by Instance are rounded off to 2 decimal places. Actual ded Current Bills.' Below this, there are filters for 'Tencent Distributed Message Q' (tdmq\_rocketmq), 'All Projects', 'All Regions', 'All transaction types', and 'All Tags'. A checkbox for 'Do not display \$0 transactions' is present. The summary shows: 'Total Cost (Including Tax) 2.57 USD = Total Amount After Discount (Excluding Tax) 2.57 USD - Voucher Deduction 0.00 USD +'. A table lists the instance details:

Instance ID	Instance Name	Product Name	Subproduct Name	Billing Mode
1-1305469081-topic...		Tencent Distributed Message Queue	tdmq_rocketmq	Pay-As-You-Go resourc

Total items: 1

Bill details: Select the **Bill Details** tab on the bill details page. Taking TDMQ for RocketMQ as an example, select **TDMQ** for the product and **TDMQ for RocketMQ** for the subproduct, and then you can view the fees of topic resource occupation and API calls for each application in the billing cycle.



Tencent Cloud
Overview Products Cloud Kafka Cloud Load Balancer Cloud Virtual Machine TencentDB for MongoDB

**Billing Center**

- Account Info
- Order Management
- Renewal Management
- Reserved Instance
- Payment Management
- Bills**
  - Bill Overview
  - Bill Details
  - Bill Download
  - Invoicing
  - Cost Allocation Tags
  - Cost Management
  - Consumption Bill

### Bill Details 2023-03

Bill by Instance **Bill Details**

**Expense figures in Bill Details are accurate up to 8 decimal places. Expense figures in Bill by Instance are rounded off to 2 decimal places. Actual details of Current Bills.**

All products Please choose one product Please choose one subproduct All Projects

All Billing Modes All transaction types  Do not display \$0 transactions

Total Cost (Including Tax) **1,653.54244695 USD** = Total Amount After Discount (Excluding Tax) **1,653.54244637 USD** - Voucher **USD** + Tax Amount **0.00000000 USD**

Instance ID	Instance Name	Product Name	Billing Mode	Instance Type
test_queue-001		Tencent Distributed Message Queue	Pay-As-You-Go resources	-
test_queue-001		Tencent Distributed Message Queue	Pay-As-You-Go resources	-
queue006		Tencent Distributed Message Queue	Pay-As-You-Go resources	-
test_queue-001		Tencent Distributed Message Queue	Pay-As-You-Go resources	-
test_queue-001		Tencent Distributed	Pay-As-You-Go resources	-

# Getting Started

## Overview

Last updated : 2023-04-14 16:54:57

TDMQ for RocketMQ supports using multi-language client SDKs to send and receive messages over the TCP and HTTP protocols. This document describes the operation process of sending and receiving general messages over these two protocols.

## Notes

TDMQ for RocketMQ supports four types of messages: general, timed/delayed, sequential messages, and transactional. This document takes general message as an example. For other types of messages, refer to [Message Type](#).

### Note:

Topics of different message types cannot be mixed, so the topics you create for general messages cannot be used to send and receive messages of other types.

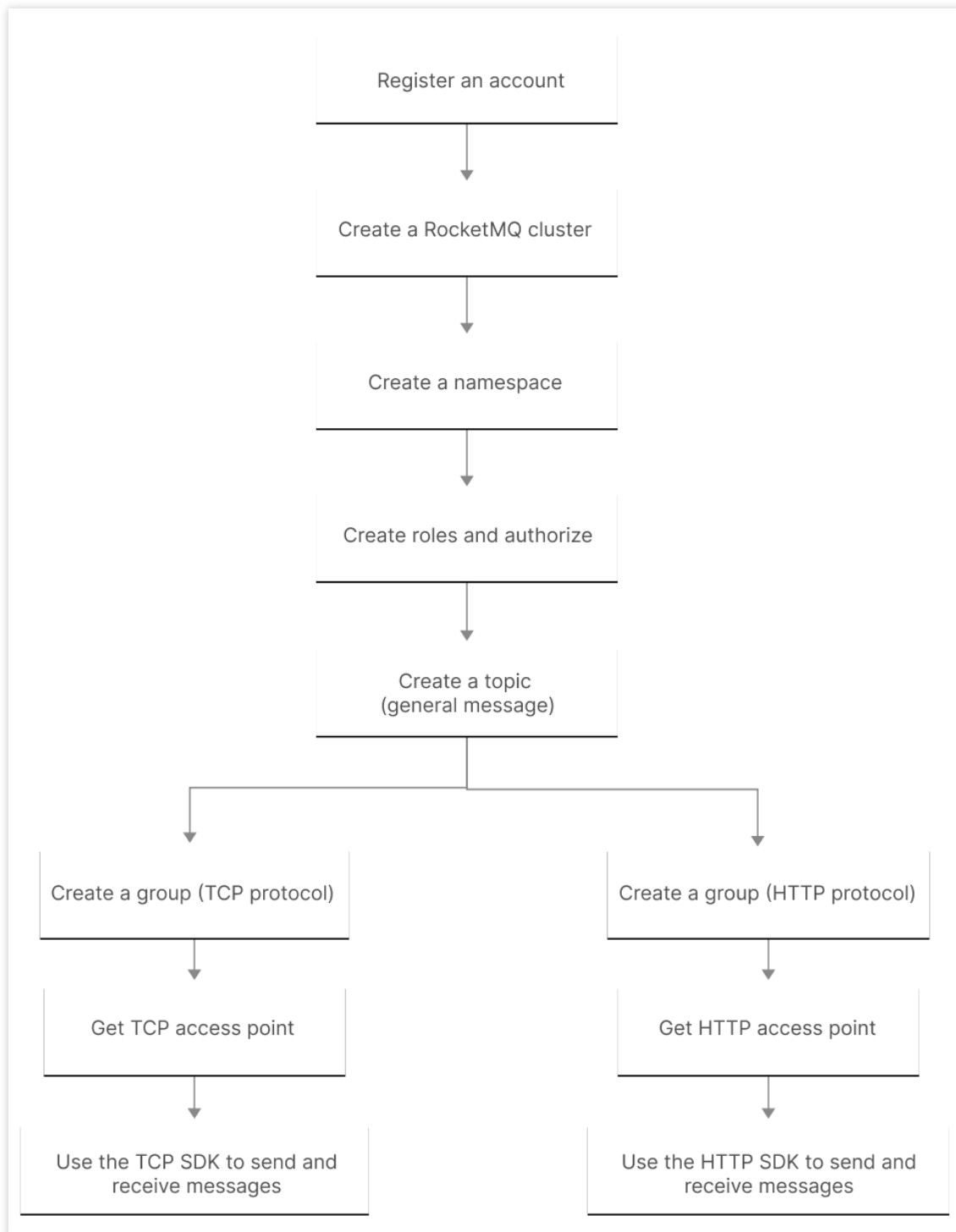
TDMQ for RocketMQ supports accesses over the TCP and HTTP protocols. Therefore, we recommend that you create corresponding types of groups for these two protocols. If multiple consumers use the same group to consume messages, with some using the TCP protocol and others using the HTTP protocol, this may result in consumption failure and message repetition or loss.

Both the TCP and HTTP protocols support the public network and VPC access addresses, and VPC is used in the production environment by default. Public network access is not enabled by default. If you are using a virtual cluster, you can [submit a ticket](#) for application to enable it. If you are using an exclusive cluster, you can [adjust public network bandwidth](#) to enable or disable it. We recommend that you use public network access only in scenarios such as testing and debugging that do not affect the production environment.

### Note:

The TCP and HTTP protocols can be supported in all regions. If the region where your current instance resides does not support the HTTP protocol and you need to use it, you can [submit a ticket](#) for application.

## Directions



# Messaging over TCP

## Resource Creation and Preparation

Last updated : 2023-09-12 16:08:15

### Overview

Before using SDK to send and receive messages over TCP, you need to create resources such as clusters and topics in the TDMQ for RocketMQ console, and configure related resource information when running the client.

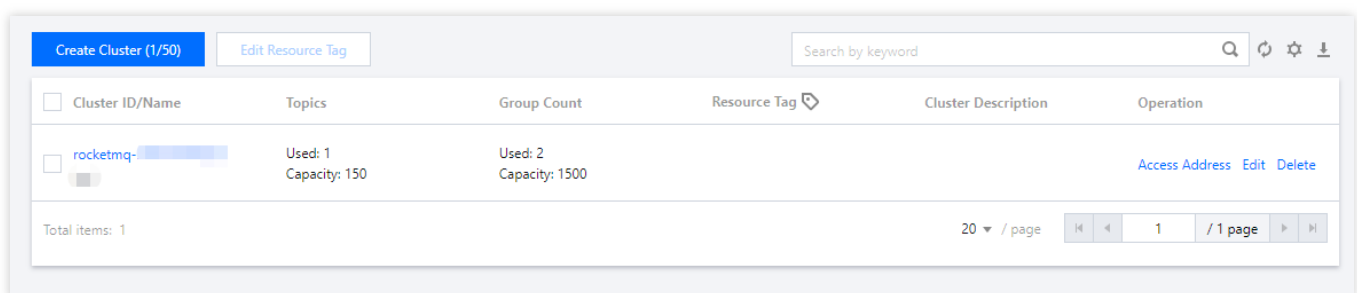
### Prerequisites

You have signed up for a Tencent Cloud account as instructed in [Signing Up](#).

### Directions

#### Step 1. Create a cluster

1. Log in to the [TDMQ console](#), enter the **Cluster** page, and select the target region.
2. Click **Create Cluster** and select **Virtual cluster**. Then, enter the cluster name and description, and click **OK** to create a cluster.



3. On the cluster list page, click the ID of the cluster you just created. In the network module of the cluster's basic information page, you can view the access point information of the cluster.

## Network

VPC Access Address <http://rocketmq-5zkv3ew9qqqr.rocketmq.ap-gz.qcloud.tencenttdmq.com:5098>

### Step 2. Create a namespace

1. On the **Cluster** list page, click the ID of the cluster created in **Step 1** to enter the cluster's basic information page.
2. Select the **Namespace** tab at the top, click **Create**, and set the namespace name and description to create a namespace.

Basic Info	Namespace	Topic	Group
<a href="#">Create (1/10)</a> <input type="text" value="Enter a keyword"/> <input type="button" value="Q"/> <input type="button" value="Refresh"/>			
Namespace Name	Message Retention Period ⓘ	Description	Operation
sdaa rocketmq	3 days		<a href="#">Configure Permission</a> <a href="#">Edit</a> <a href="#">Delete</a>
Total items: 1			20 / page <input type="button" value="H"/> <input type="button" value="Left"/> <input type="text" value="1"/> <input type="button" value="Right"/> / 1 page <input type="button" value="Next"/>

### Step 3. Create a role and configure permissions

1. Select **Role Management** on the left sidebar and click **Create** to create a role.
2. On the **Cluster** page, click the ID of the cluster you just created in **step 1** to enter the **cluster** details page.
3. Select the **Namespace** tab at the top and click **Configure Permissions** in the **Operation** column of the namespace you just created.
4. On the **Configure Permission** page, click **Add Role** to add production and consumption permissions to the role you just created.

**Create** ✕

Role:  ↕ ↻

Unable to find a role? Please configure a role and token on the [Role Management](#) page.

Permission:  Message production  
 Message consumption

For more permission type information, see [Permission Description](#).

#### Step 4. Create a topic

1. On the **Namespace** list page, select the **Topic** tab at the top to enter the **Topic** list page.
2. Select the namespace created in [Step 3](#) and click **Create**. Then, enter the topic name, select **General message** as the message type, and click **OK** to create a topic.

#### Note

This document takes sending and receiving general messages as an example. Therefore, the topic of general messages created by referring to the above steps cannot be used for messages of other types.

Basic Info		Namespace	<b>Topic</b>	Group		
Current Namespace	<input type="text" value="sdaa"/>	Message Retention Period	3 days	Max TPS <span style="font-size: small;">ⓘ</span> 4000		
<input type="button" value="Create (1/150)"/>		<input type="text" value="Enter a keyword"/> <span style="margin-left: 10px;">🔍</span> <span style="margin-left: 10px;">↻</span> <span style="margin-left: 10px;">⚙️</span> <span style="margin-left: 10px;">⏴</span>				
Topic Name	Monitoring	Type	Partition Count	Subscribed Groups	Description	Operation
dsada	<span style="font-size: small;">📊</span>	General message	7	0		<a href="#">Edit</a> <a href="#">Delete</a>
Total items: 1		20 / page		<span style="font-size: small;">⏴</span> <span style="font-size: small;">⏵</span> <span style="font-size: small;">1</span> / 1 page		

#### Step 5. Create a group

1. On the **Topic** list page, select the **Group** tab at the top to enter the **Group** list page.
2. Select the namespace you just created and click **Create**. Then, enter the group name, select **TCP** as the protocol type, and click **OK** to create a group.

#### Note

TDMQ for RocketMQ supports the TCP and HTTP protocols. Therefore, we recommend that you create corresponding types of groups for these two protocols. If multiple consumers use the same group to consume messages, with some using the TCP protocol and others using the HTTP protocol, this may result in consumption failure and message repetition or loss.

Basic Info   Namespace   Topic   **Group**

Current Namespace:    Message Retention Period: 3 days   Max TPS ⓘ: 4000

[Create \(2/1500\)](#)  

Group Name	Consumer Info ↕	Consumption Mode	Description	Operation
	Online Consumer   0   TPS   0   Total Heap   0 <input type="button" value="🔄"/>	Unknown		<a href="#">Consumer Details</a> <a href="#">Reset Offset</a> <a href="#">E</a> <a href="#">Delete</a>
	Online Consumer   0   TPS   0   Total Heap   0 <input type="button" value="🔄"/>	Unknown		<a href="#">Consumer Details</a> <a href="#">Reset Offset</a> <a href="#">E</a> <a href="#">Delete</a>

Total items: 2   20 ▾ / page     1 / 1 page

# Downloading and Running Demo

Last updated : 2023-03-28 10:15:36

## Overview

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

### Note

The following takes the Java client as an example. For clients in other languages, see [SDK Documentation](#).

## 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

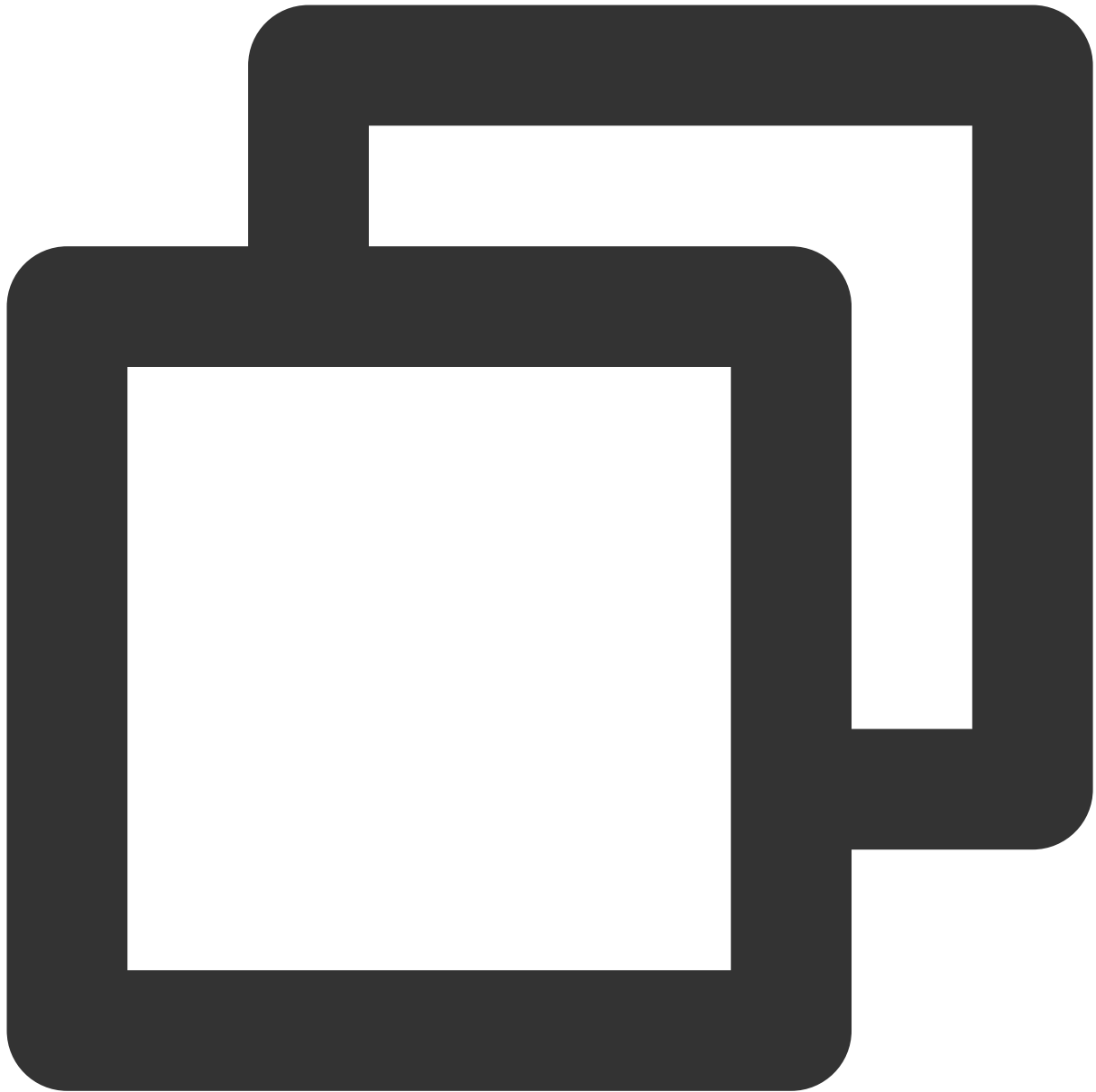
### Step 1. Install the Java dependent library

Introduce dependencies in a Java project and add the following dependencies to the `pom.xml` file. This document uses a Maven project as an example.

### Note

The dependency version must be v4.9.3 or later.





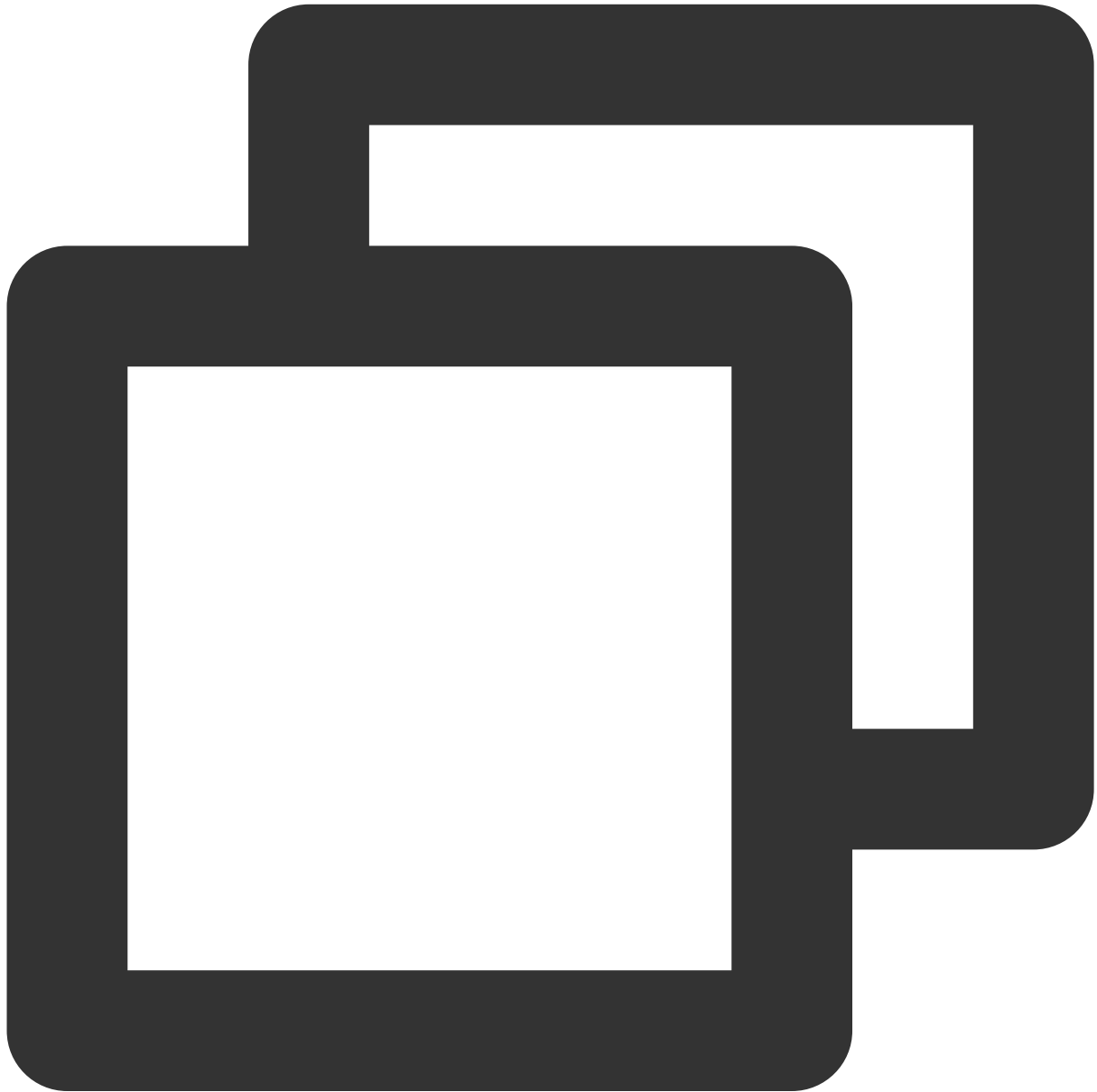
```
<!-- in your <dependencies> block -->
<dependency>
  <groupId>org.apache.rocketmq</groupId>
  <artifactId>rocketmq-client</artifactId>
  <version>4.9.3</version>
</dependency>

<dependency>
  <groupId>org.apache.rocketmq</groupId>
  <artifactId>rocketmq-acl</artifactId>
  <version>4.9.3</version>
```

```
</dependency>
```

## Step 2. Produce messages

### 1. Create message producers

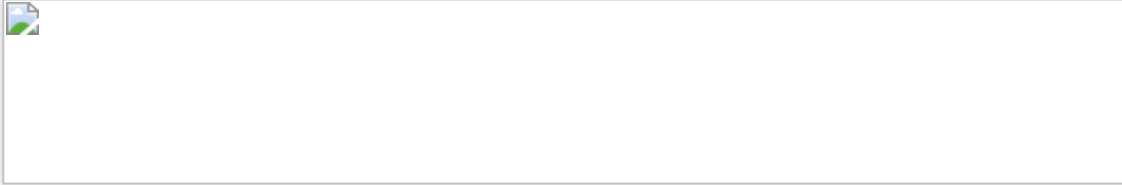



```
// Instantiate the message producers
DefaultMQProducer producer = new DefaultMQProducer(
    namespace,
    groupName,
    new AclClientRPCHook(new SessionCredentials(accessKey, secretKey))
```

```

    // ACL permission
);
// Set the NameServer address
producer.setNamesrvAddr(nameserver);
// Start the producer instances
producer.start();

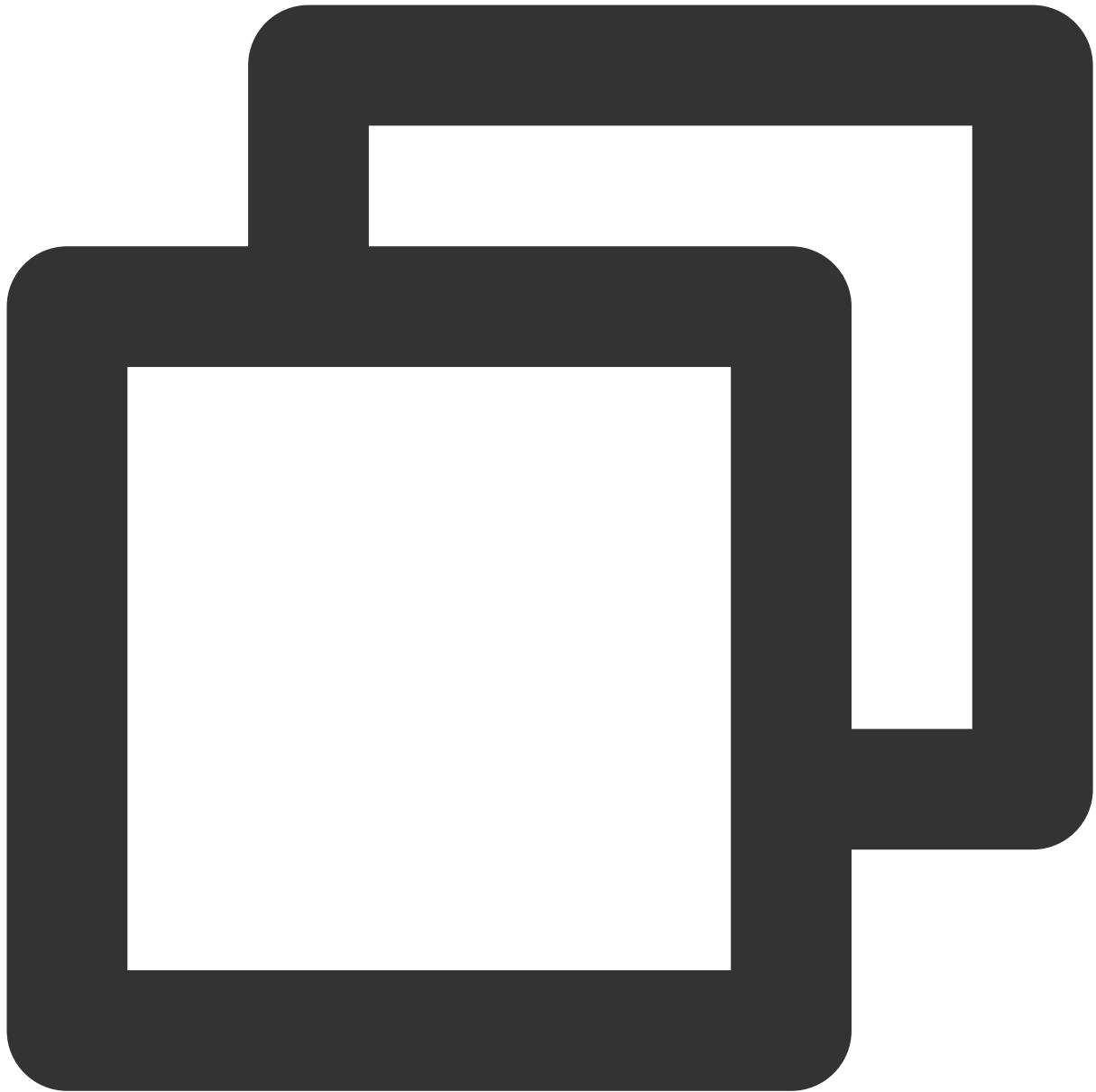
```

Parameter	Description
namespace	Namespace name, which can be copied on the <b>Namespace</b> page in the console. 
groupName	Producer group name, which can be copied under the <b>Group</b> tab on the <b>Cluster</b> page in the console.
nameserver	Cluster access address, which can be copied under the <b>Network</b> module on the cluster's basic info page.
secretKey	Role name, which can be copied on the <a href="#">Role Management</a> page.
accessKey	Role token, which can be copied in the <b>Token</b> column on the <a href="#">Role Management</a> page. 

## 2. Send messages

Messages can be sent in the sync, async, or one-way mode.

Sync sending

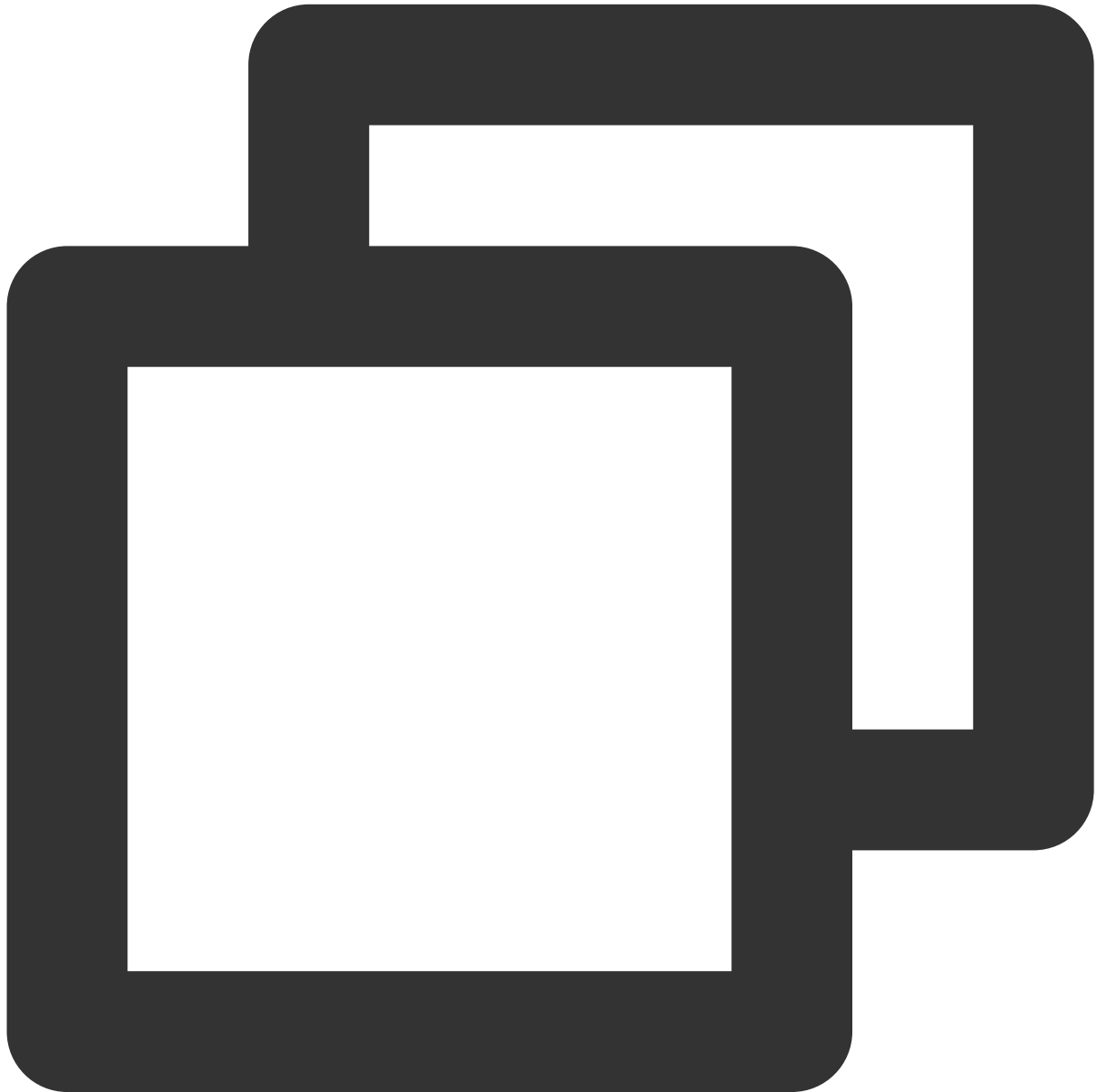


```
for (int i = 0; i < 10; i++) {  
    // Create a message instance and set the topic and message content  
    Message msg = new Message(topic_name, "TAG", ("Hello RocketMQ " + i).getBytes()  
    // Send the message  
    SendResult sendResult = producer.send(msg);  
    System.out.printf("%s\n", sendResult);  
}
```

Parameter	Description

topic_name	Topic name, which can be copied under the <b>Topic</b> tab on the <b>Cluster</b> page in the console.
tag	A parameter used to set the message tag.

### Async sending



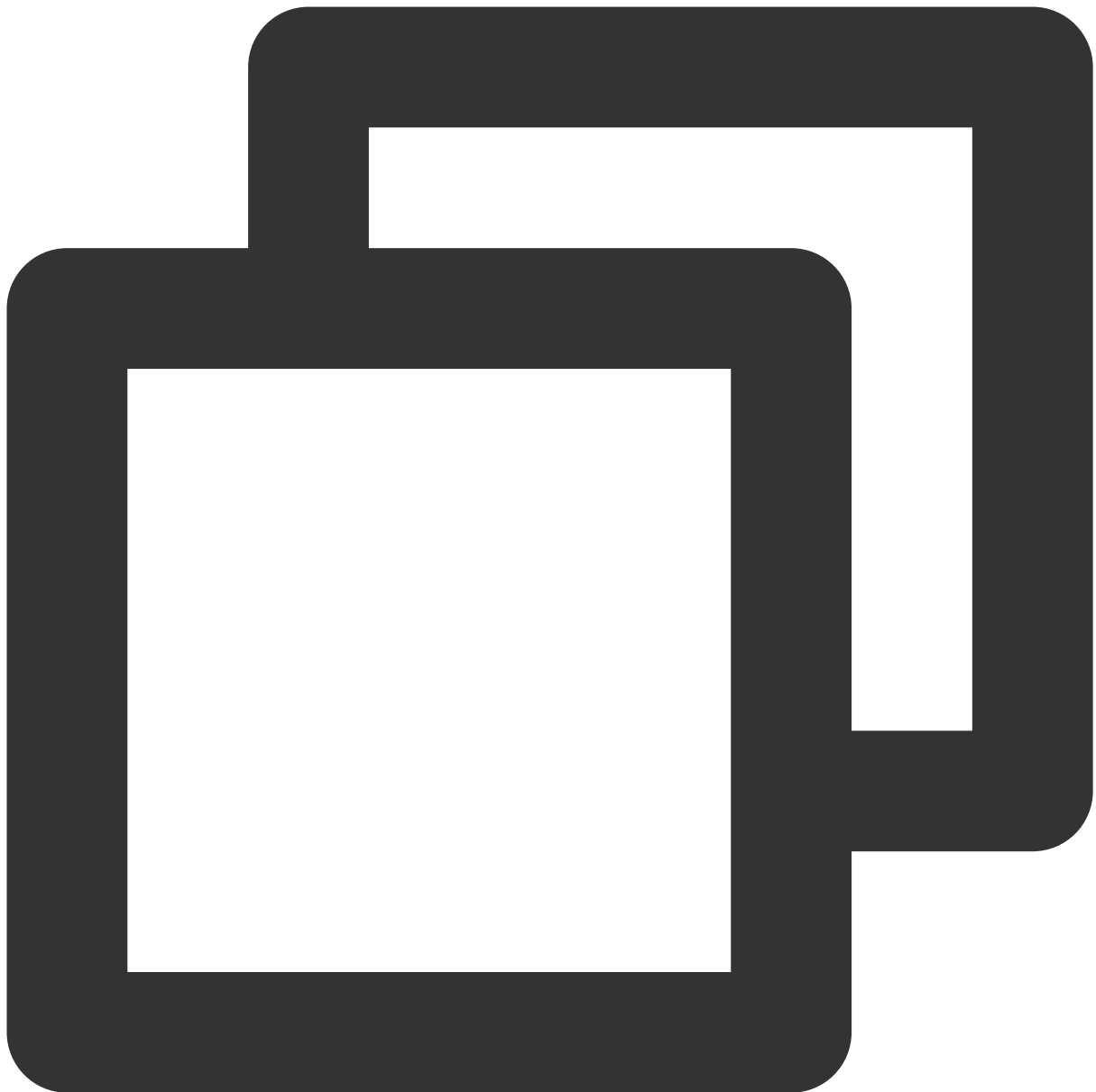
```
// Disable retry upon sending failures
producer.setRetryTimesWhenSendAsyncFailed(0);
// Set the number of messages to be sent
int messageCount = 10;
```

```
final CountdownLatch countDownLatch = new CountdownLatch(messageCount);
for (int i = 0; i < messageCount; i++) {
    try {
        final int index = i;
        // Create a message instance and set the topic and message content
        Message msg = new Message(topic_name, "TAG", ("Hello rocketMq " + index));
        producer.send(msg, new SendCallback() {
            @Override
            public void onSuccess(SendResult sendResult) {
                // Logic for message sending successes
                countDownLatch.countDown();
                System.out.printf("%-10d OK %s %n", index, sendResult);
            }

            @Override
            public void onException(Throwable e) {
                // Logic for message sending failures
                countDownLatch.countDown();
                System.out.printf("%-10d Exception %s %n", index, e);
                e.printStackTrace();
            }
        });
    } catch (Exception e) {
        e.printStackTrace();
    }
}
countDownLatch.await(5, TimeUnit.SECONDS);
```

Parameter	Description
topic_name	Topic name, which can be copied under the <b>Topic</b> tab on the <b>Cluster</b> page in the console.
tag	A parameter used to set the message tag.

### One-way sending



```
for (int i = 0; i < 10; i++) {  
    // Create a message instance and set the topic and message content  
    Message msg = new Message(topic_name, "TAG", ("Hello RocketMQ " + i).getBytes()  
    Send one-way messages  
    producer.sendOneway(msg);  
}
```

Parameter	Description
topic_name	Topic name, which can be copied under the <b>Topic</b> tab on the <b>Cluster</b> page in the

---

	console.
tag	A parameter used to set the message tag.

**Note**

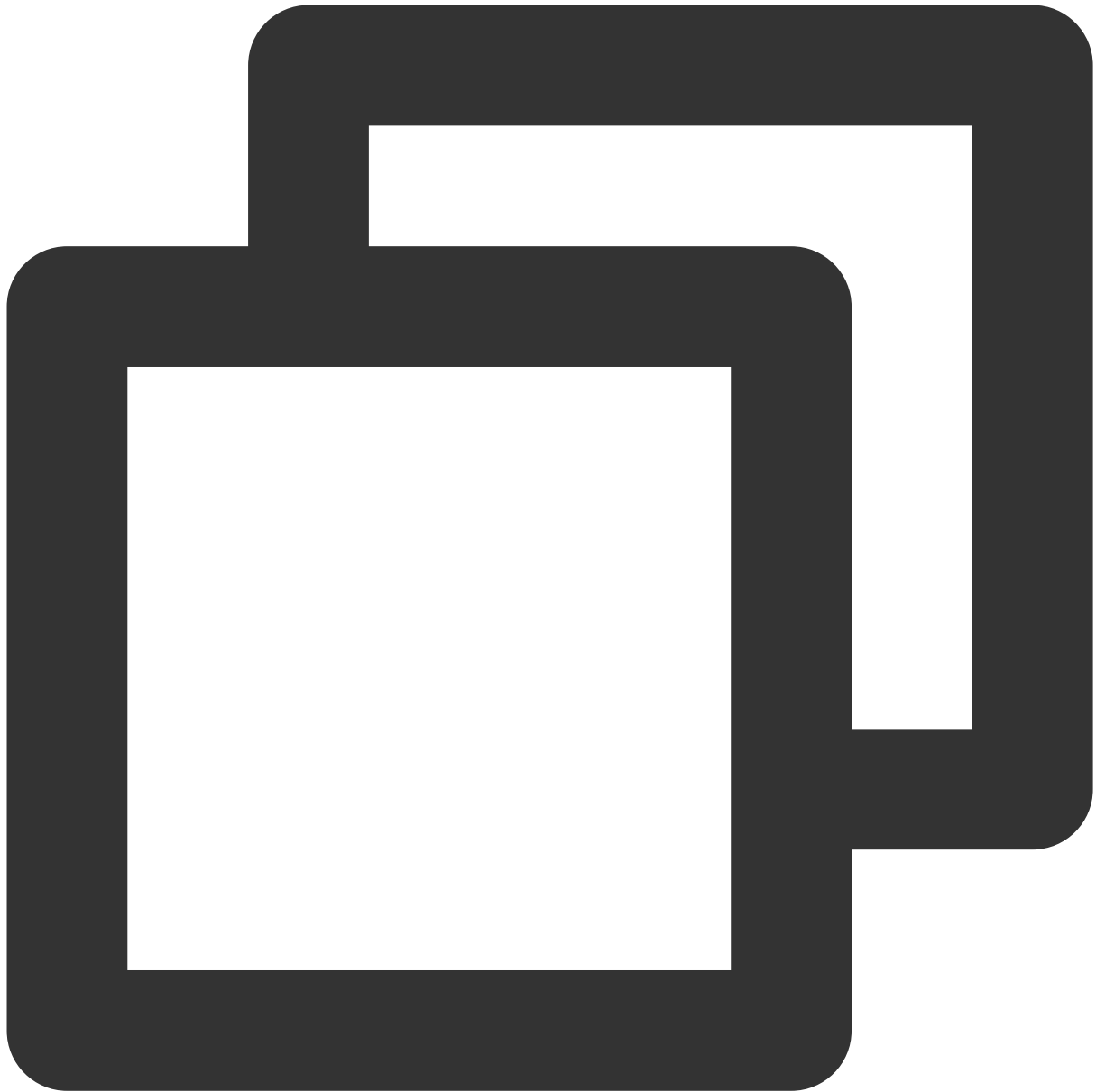
For more information on batch sending or other scenarios, see [Demo](#) or [RocketMQ documentation](#).

**Step 3. Consume messages****1. Create a consumer**

TDMQ for RocketMQ supports two consumption modes: push and pull.

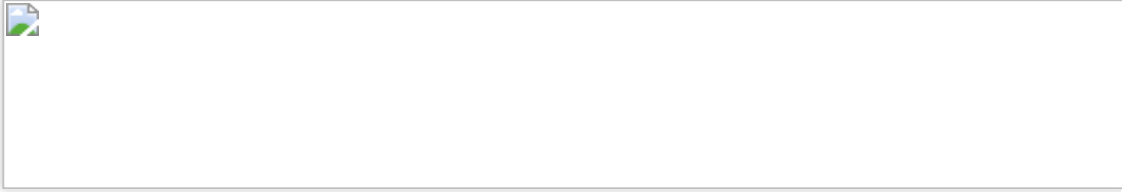

For consumers using the push mode:



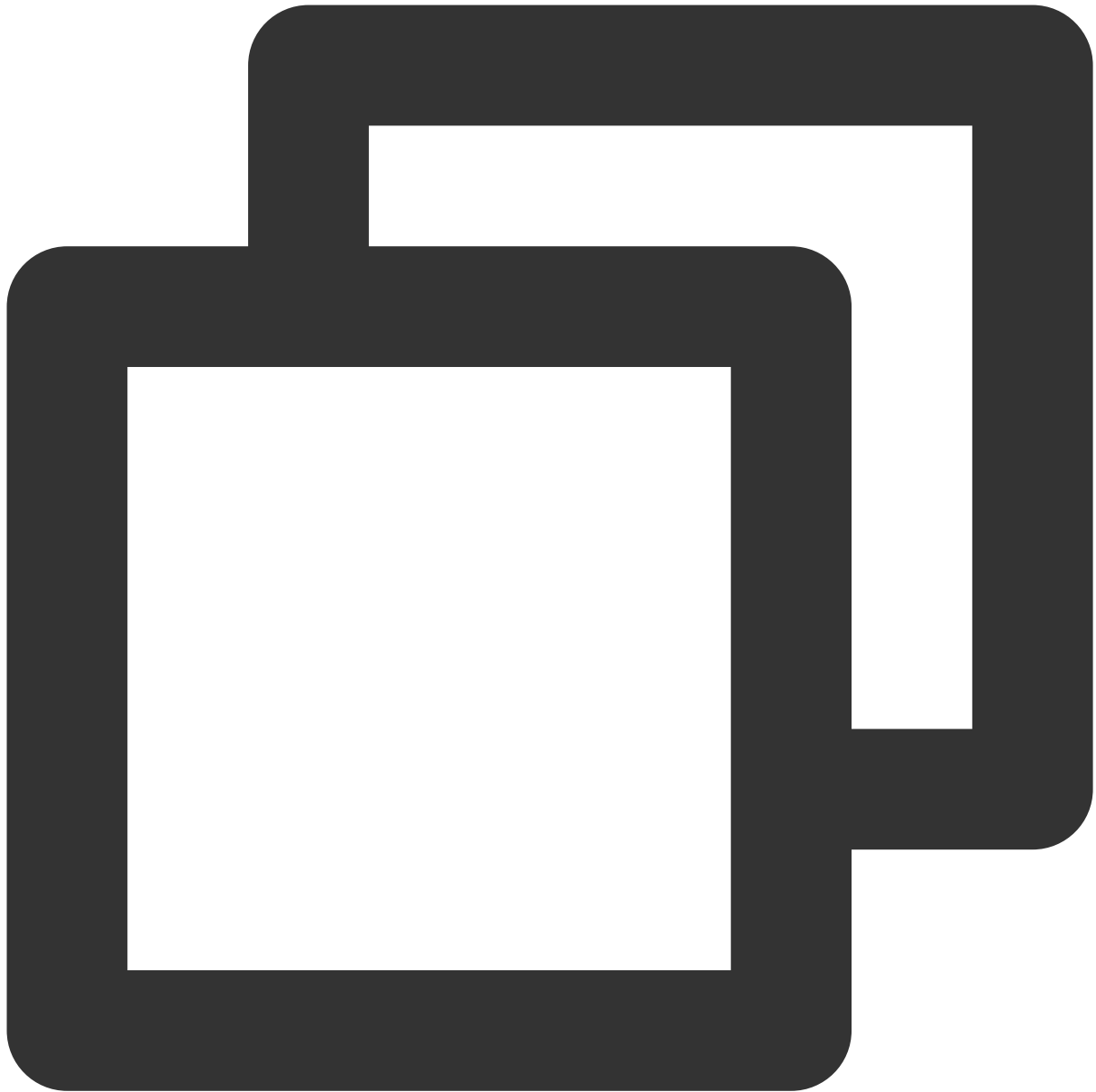


```
// Instantiate the consumer
DefaultMQPushConsumer pushConsumer = new DefaultMQPushConsumer(
    namespace,
    groupName,
    new AclClientRPCHook(new SessionCredentials(accessKey, secretKey))); //ACL per
// Set the NameServer address
pushConsumer.setNamesrvAddr(nameserver);
```


Parameter	Description

namespace	Namespace name, which can be copied on the <b>Namespace</b> page in the console. 
groupName	Producer group name, which can be copied under the <b>Group</b> tab on the <b>Cluster</b> page in the console.
nameserver	Cluster access address, which can be copied under the <b>Network</b> module on the cluster's basic info page.
secretKey	Role name, which can be copied on the <a href="#">Role Management</a> page.
accessKey	Role token, which can be copied in the <b>Token</b> column on the <a href="#">Role Management</a> page. 

For consumers using the pull mode:



```
// Instantiate the consumer
DefaultLitePullConsumer pullConsumer = new DefaultLitePullConsumer(
    namespace,
    groupName,
    new AclClientRPCHook(new SessionCredentials(accessKey, secretKey)));
// Set the NameServer address
pullConsumer.setNamesrvAddr(nameserver);
// Specify the first offset as the start offset for consumption
pullConsumer.setConsumeFromWhere(ConsumeFromWhere.CONSUME_FROM_FIRST_OFFSET);
```

Parameter	Description
namespace	Namespace name, which can be copied under the <b>Namespace</b> tab in the console. Its format is clus
groupName	Producer group name, which can be copied under the <b>Group</b> tab on the <b>Cluster</b> page in the consol
nameserver	Cluster access address, which can be obtained from <b>Access Address</b> in the <b>Operation</b> column or page in the console.
secretKey	Role name, which can be copied on the <a href="#">Role Management</a> page.
accessKey	Role token, which can be copied in the <b>Token</b> column on the <a href="#">Role Management</a> page. 

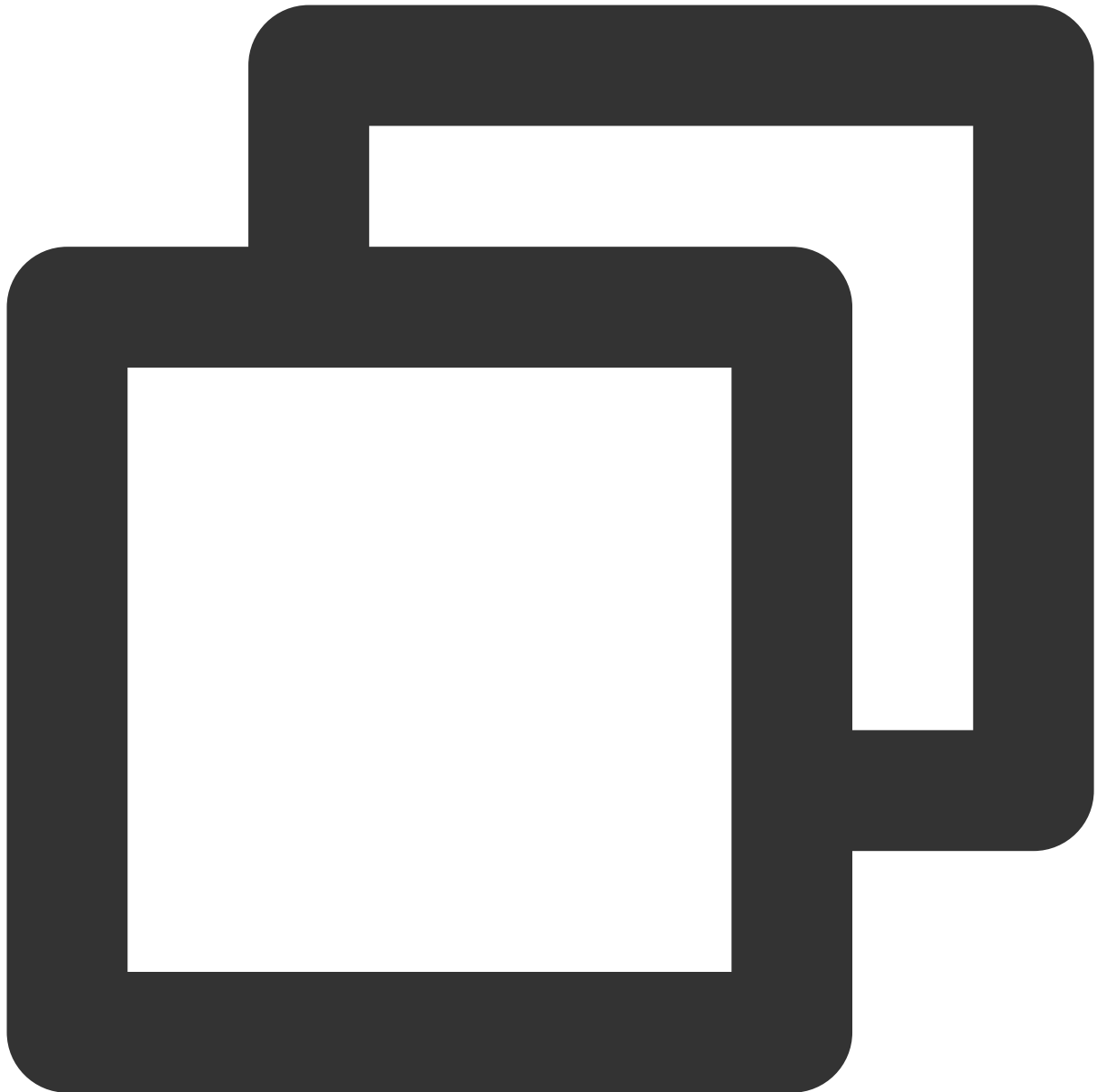
**Note**

For more consumption mode information, see [Demo](#) or [RocketMQ documentation](#).

**2. Subscribe to messages**

The subscription modes vary by consumption mode.

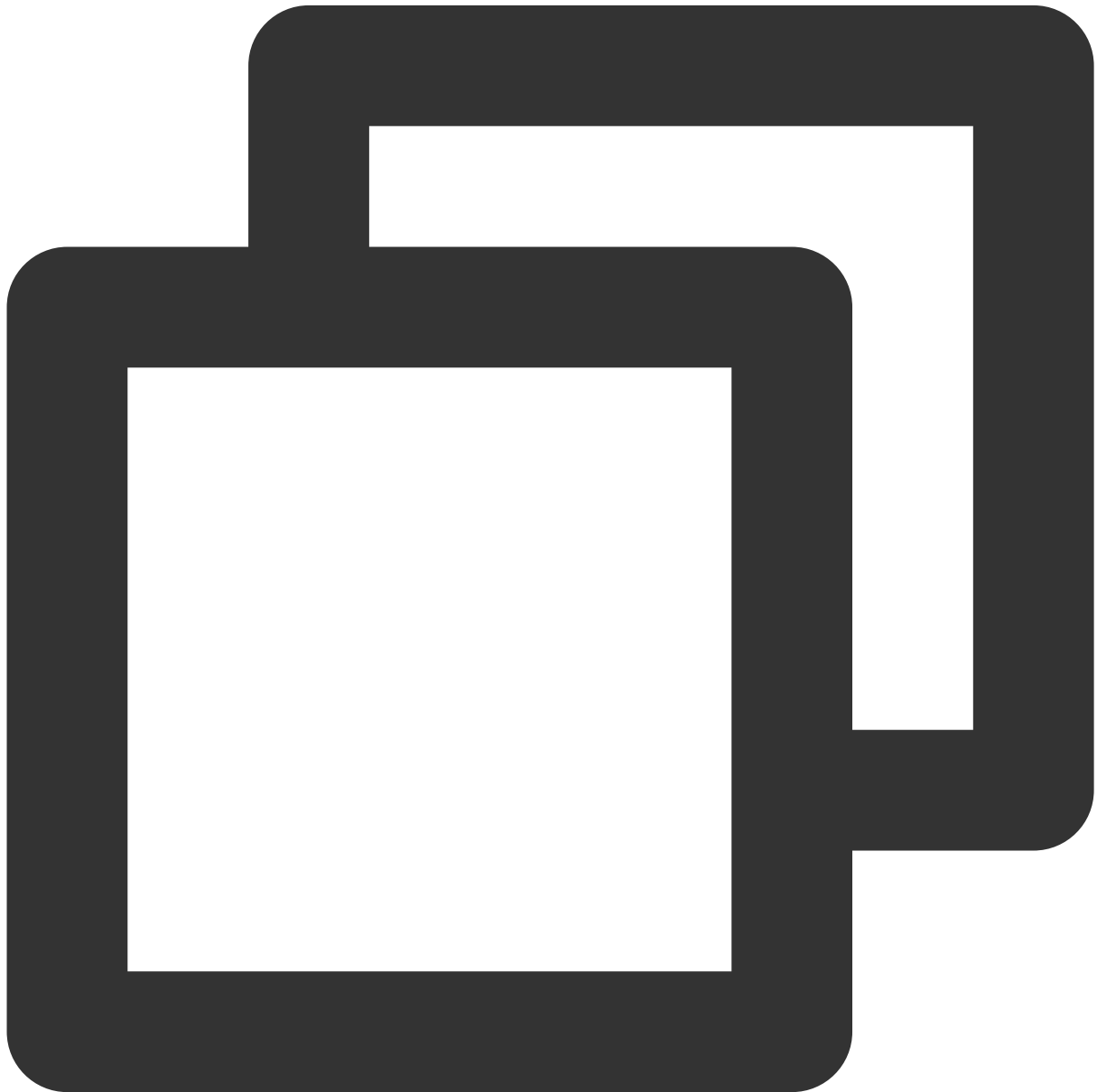
Subscription in push mode



```
// Subscribe to a topic
pushConsumer.subscribe(topic_name, "*");
// Register a callback implementation class to process messages pulled from the bro
pushConsumer.registerMessageListener((MessageListenerConcurrently) (msgs, context)
    // Message processing logic
    System.out.printf("%s Receive New Messages: %s %n", Thread.currentThread().get
    // Mark the message as being successfully consumed and return the consumption
    return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
});
// Start the consumer instance
pushConsumer.start();
```

Parameter	Description
topic_name	Topic name, which can be copied under the <b>Topic</b> tab on the <b>Cluster</b> page in the console.
"*"	If the subscription expression is left empty or specified as asterisk (*), all messages are subscribed to. <code>tag1    tag2    tag3</code> means subscribing to multiple types of tags.

### Subscription in pull mode



```
// Subscribe to a topic
```

```
pullConsumer.subscribe(topic_name, "*");
// Start the consumer instance
pullConsumer.start();
try {
    System.out.printf("Consumer Started.%n");
    while (true) {
        // Pull the message
        List<MessageExt> messageExts = pullConsumer.poll();
        System.out.printf("%s%n", messageExts);
    }
} finally {
    pullConsumer.shutdown();
}
```

Parameter	Description
topic_name	Topic name, which can be copied under the <b>Topic</b> tab on the <b>Cluster</b> page in the console.
"*"	If the subscription expression is left empty or specified as asterisk (*), all messages are subscribed to. <code>tag1    tag2    tag3</code> means subscribing to multiple types of tags.

#### Step 4. View consumption details

Log in to the [TDMQ console](#), go to the **Cluster > Group** page, and view the list of clients connected to the group. Click **View Details** in the **Operation** column to view consumer details.

**Note**

Above is a brief introduction to message publishing and subscription. For more information, see [Demo](#) or [RocketMQ documentation](#).



# Operation Guide

## Cluster Management

Last updated : 2023-07-25 14:55:55

### Overview

Cluster is a resource dimension in TDMQ for RocketMQ, and namespaces, topics, and groups of different clusters are completely isolated from each other. Each cluster has its own resource limits, such as the total number of topics and message retention period. It is a common practice to use respective dedicated clusters for development, test, and production environments.

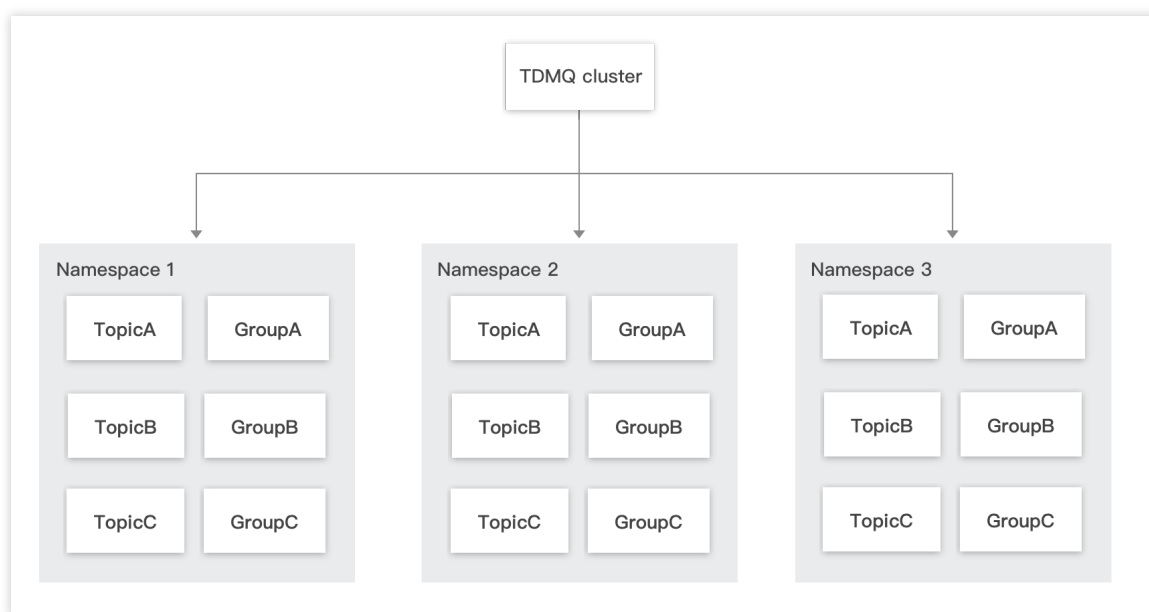
Clusters are divided into virtual clusters and exclusive clusters:

**Exclusive cluster:** Physical resources are exclusive, and data is secure. There are almost no use limits.

**Virtual cluster:** Virtual computing and storage resources are used and automatically allocated based on usage.

There are certain use limits.

#### TDMQ for RocketMQ resource hierarchy



### Directions

#### Creating a cluster

1. Log in to the [TDMQ console](#) and enter the **Cluster** page.
2. On the **Cluster Management** page, select the region and click **Create Cluster** to enter the **Create Cluster** window.

Cluster Type: Select **Exclusive cluster** or **Virtual cluster**.

Exclusive cluster: Physical resources are exclusive, and data is secure. There are almost no use limits.

Virtual cluster: Virtual computing and storage resources are used and automatically allocated based on usage. There are certain use limits.

Billing Mode: Exclusive clusters are **monthly subscribed**, while virtual clusters are **pay-as-you-go**.

Region: Select a region closest to your business. Tencent Cloud products in different regions cannot communicate with each other over the private network. Select a region with caution, as it cannot be changed after purchase. For example, CVM instances in Guangzhou region cannot access clusters in Shanghai region over the private network. If you need cross-region communication over the private network, see [Creating Intra-account Peering Connection](#).

Cluster Specification: Select an appropriate cluster specification as needed.

VPC: Authorize to bind the domain name of the new cluster's access point to the specified VPC.

Public Network Access: After the public network bandwidth is enabled, additional fees will be incurred, and you need to configure a security policy. If you don't configure any security policy, all IPs will be denied access by default.

Exclusive cluster: For pricing information, see [Exclusive Cluster > Public Network Bandwidth Pricing](#).

Virtual cluster: Billing by the public network bandwidth hasn't been started, and we recommend that you use the public network access only in testing and debugging environments. The public network access is not enabled by default. To enable it, [submit an application](#).

Cluster Name: Enter the cluster name, which can contain 3–64 digits, letters, hyphens, and underscores.

Tag: Tags are used to manage resources by category in different dimensions. For detailed directions, see [Managing Resource with Tag](#).

3. Click **Buy Now**.

### Next steps:

1. Create a namespace in the cluster and get the access address and connection information of the server. For more information, see [Namespace Management](#).
2. Create a [role](#) in the cluster and grant it the production/consumption permissions of the namespace.
3. Create a [topic](#) in the namespace.
4. Write a demo as instructed in the [SDK documentation](#) and configure the connection information for message production/consumption.

## Viewing cluster details

Exclusive Cluster

Virtual Cluster

On the **Cluster Management** list page, click **View Details** in the **Operation** column to enter the cluster details page.

On the details page, you can query:

Cluster Statistics: This section displays the current cluster's message consumption speed, message production rate, number of heaped messages, cluster consumption traffic per second, and cluster production traffic per second in the selected time range.

Cluster's basic information (cluster name/ID, region, creation time, remarks, and resource tag)

Network: This section displays the VPC, public network bandwidth, as well as the private and public network access points for TCP and HTTP.

Cluster Overview: This section displays the number of resources, resource quota usage, and message type distribution in the current cluster.

Top Cluster Resource Consumption: This section displays the ranking of groups and topics using major resources in the current cluster, including the ranking of heaped message count and dead letter count of the groups, the ranking of production and consumption speed of the topics, and the ranking of occupied storage space of the topics.

After the exclusive cluster is created, you can modify the message retention time of the cluster based on the metrics of storage usage and business peak conditions. On the cluster list page, click **Edit** to modify the message retention time on the cluster information page.

On the **Cluster Management** list page, click **View Details** in the **Operation** column to enter the cluster details page.

On the details page, you can query:

Cluster overview (numbers of topics, currently heaped messages, and month-to-date API calls)

Cluster's basic information (cluster name/ID, region, creation time, remarks, and resource tag)

Cluster configuration:

Instance Configuration	Configuration Description
Max TPS	The upper limit for production TPS does not apply to consumption TPS.
Max Namespaces	Maximum number of namespaces that can be created.
Max Topics	Maximum number of topics that can be created.
Max Groups	Maximum number of groups that can be created.
Max Retention Period	Maximum message retention period that can be configured.

Network: This section displays the private and public network access points for TCP and HTTP.

Top Cluster Resource Consumption: This section displays the ranking of groups and topics using major resources in the current cluster, including the ranking of heaped message count and dead letter count of the groups as well as the ranking of production and consumption speed of the topics.

**Cluster Overview**

Topics	Messages Produced in Last 24 Hours	Currently Heaped Messages
1	0	0

**Basic Info** [Edit](#)

Name	test
ID	rocketmq-5zlv3ew9qqqr
Region	South China(Guangzhou)
Creation Time	2022-03-10 10:45:37
Description	-
Resource Tag	No tag ✓

**Instance Configuration**

Max TPS Per Topic Partition	4000
Max Namespaces	10 (1 used/10)
Max Topics	150 (1 used/150)
Max Groups	1500 (2 used/1500)
Max Retention Period	3 days
Max Message Delay	40 days

**Network**

VPC Access Address <http://rocketmq-5zlv3ew9qqqr.rocketmq.ap-gz.tencentcloud.com:5098>

## Upgrading the cluster specification

If the current cluster specifications cannot meet your business needs, you can increase the node specification, node quantity, and storage specification in the console.

### Note

**Currently, cluster specification upgrade is supported only for exclusive clusters.**

Currently, you can only upgrade but not downgrade node count and node storage specification.

During the upgrade of exclusive clusters, whether the operations involve adding new nodes, upgrading node specifications or upgrading storage specifications, Tencent Cloud ensures that the upgrade process is smooth and imperceptible, without causing downtime on the client-side applications.

1. On the **Cluster** list page, click **Edit** in the **Operation** column of the target cluster.
2. Select the target node specification and click **Confirm the Adjustment**.

Target Node Count: You can only increase but not decrease it.

Single-Node Storage Specification: You can only increase but not decrease it. The new single-node storage specification takes effect for all nodes in the cluster.

## Downgrading the cluster specification

After purchasing an exclusive cluster, you can adjust its node specifications based on the business volume and cluster usage.

1. On the **Cluster** list page, click **Downgrade** in the **Operation** column of the target cluster.
2. Select the target node specification and click **Confirm the Adjustment**.

### Note

Currently, the single-node storage specification cannot be downgraded because of the constraints of the underlying components. In order to avoid data loss during the downgrade, you can only modify the node specifications of the cluster, but not the number of nodes.

## Adjusting the public network bandwidth

### Note

**Currently, this is supported only for exclusive clusters.**

After creating the exclusive cluster, you can enable/disable public network access by adjusting the public network bandwidth, modify the public network bandwidth value, and configure the public network security policy to restrict user access.

There are two ways to adjust the public network bandwidth:

Option 1: On the cluster list page, click **More** > **Adjust Public Network Bandwidth** in the **Operation** column.

Option 2: On the cluster's **Basic Info** page, click **Edit** for **Public Network Bandwidth** in the **Network** section.

Public Network Access: After the public network bandwidth is enabled, additional fees will be incurred. For pricing information, see [Exclusive Cluster > Public Network Bandwidth Pricing](#).

Public Network Bandwidth: Select the target public network bandwidth.

Public Network Access Policy: Enter the IPs that are allowed to access. If you don't configure any security policy, all IPs will be denied access by default. If the newly added rule and an existing rule overlap, the new rule will be matched first.

**Editing a cluster**

1. On the **Cluster** list page, click **Edit** in the **Operation** column of the target cluster.
2. Enter the cluster name and description in the pop-up window and click **Submit**.

**Deleting a cluster****Note**

After a cluster is deleted, all its configurations will be cleared and cannot be recovered. Therefore, proceed with caution.

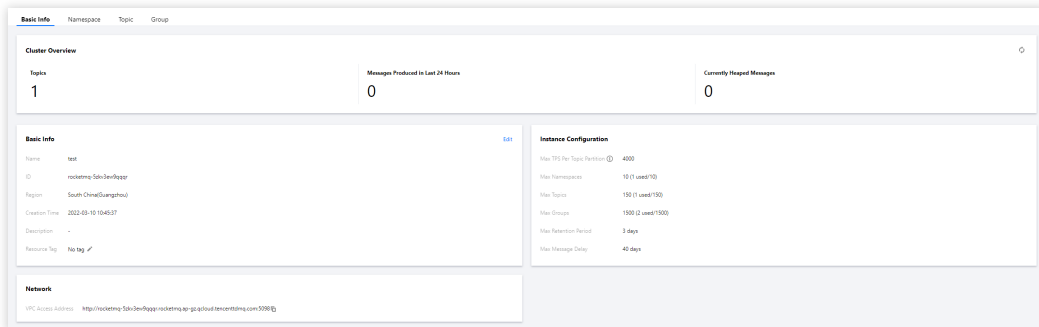
You can delete a cluster in the following steps:

1. On the **Cluster** list page, click **Delete** in the **Operation** column of the target cluster.
2. In the deletion confirmation pop-up window, click **Delete**.

**Note**

In order to prevent misoperations from causing internal cluster data (such as topic and group) to be deleted, when you delete a cluster, the resources in your cluster will be verified in the console. If there are undeleted resources such as namespaces, topics or groups, the cluster cannot be deleted.

When deleting a namespace, you need to delete the topic and group in the namespace first. Currently, you can batch delete topics and groups in the console. Before deleting a namespace, you also need to delete the role bound to the namespace. To unbind the role from the namespace, go to the namespace list page, click **Configure Permissions**.



After entering the permission configuration page, click **Delete** to unbind the two from each other, and then you can go to the namespace page again to delete the namespace.

# Namespace Management

Last updated : 2023-03-14 15:22:27

## Overview

In TDMQ for RocketMQ, namespace is a resource management concept. It can generally be used to isolate different businesses via customized configurations such as message retention period. Topics, subscriptions, and role permissions in one namespace are all kept separate from those in another.

This document describes how to create multiple namespaces in TDMQ for RocketMQ to use the same TDMQ for RocketMQ cluster in different scenarios.

### Notes

Topic and group names must be unique in the same namespace.

## Directions

### Creating a namespace

1. Log in to the [TDMQ console](#), select the region, and click the ID of the target cluster to enter the **Basic Info** page.
2. Select the **Namespace** tab at the top of the page and click **Create** to enter the namespace creation page.
3. In the **Create Namespace** window, configure the namespace attributes:



Namespace Name: Enter the namespace name, which cannot be modified after creation and can contain 3–32 letters, digits, backslashes, and underscores.

Message Retention Period: Set the retention period (0 seconds to 3 days) of unconsumed messages. After a message is successfully produced, it will be stored for a period of time no matter whether it is consumed or not. It will be automatically deleted upon expiration.

Namespace Description: Enter the remarks of the namespace.

4. Click **Save**.

### Notes

Up to ten namespaces can be created in one cluster.

After the above steps, you can create a topic in the namespace to produce and consume messages as instructed in [Topic Management](#).

### Getting the access address

On the **Namespace** list page, you can get the namespace access addresses for TCP in the **VPC Access Address** and **Public Network Access Address** columns. You can view the HTTP access point on the **Basic Info** of the cluster.



## Modifying a namespace

You can modify a namespace in the following steps:

1. On the **Namespace** list page, click **Edit** in the **Operation** column of the target namespace to enter the editing page.
2. Modify the message TTL, message retention period, or namespace description and click **Save**.

## Deleting a namespace

You can delete a created namespace in the following steps:

1. On the **Namespace** list page, click **Delete** in the **Operation** column of the target namespace.
2. In the deletion confirmation pop-up window, click **OK**.

### Notes

A namespace with topics cannot be deleted.

# Topic Management

Last updated : 2023-05-16 10:37:15

## Overview

Topic is a core concept in TDMQ for RocketMQ. It is usually used to categorize and manage various messages produced by the system in a centralized manner; for example, messages related to transactions can be placed in a topic named "trade" for other consumers to subscribe to.

In actual application scenarios, a topic often represents a business category. You can decide how to design different topics based on your system and data architectures.

This document describes how to use topics to categorize and manage messages in TDMQ for RocketMQ.

## Prerequisites

You have created a namespace.

## Directions

### Creating a topic

1. Log in to the [TDMQ console](#), select the region, and click the ID of the target cluster to enter the **Basic Info** page of the cluster.
2. Click the **Topic** tab at the top, select a namespace, and click **Create** to enter the **Create Topic** page.
3. In the **Create Topic** window, enter the following information:

### Create Topic ✕

**i** There is currently 1 topic, and 149 more can be created.

Current Namespace `sdaa`

Topic Name \*   
This field is required and can only contain 3-64 letters, digits, hyphens (-), and underscores (\_). You can enter 64 more characters.

Type \*   
For information on message types, see [here](#) **🔗**

Partition Count \*  3 32     
You can select multiple partitions to optimize the message production and consumption performance of a single topic, but this does not guarantee that the messages will be produced or consumed in the proper sequence.

Topic Description   
The remarks can contain up to 128 characters.

Topic Name: enter the topic name, which cannot be modified after creation and can contain 3-64 letters, digits, hyphens, and underscores.

Type: Select the message type, including general, sequential, delayed, and transactional. For the description of the message type, see [Message Type](#).

Partition Count: Select the number of partitions, which can be up to 16. Using multiple partitions can improve the production/consumption performance of a single topic but cannot guarantee the sequence.

Remarks: enter the topic remarks.

4. Click **Submit**, and you can see the created topic in the topic list.

## Sending a test message

You can manually send a message to the specified topic in the TDMQ for RocketMQ console.

1. In the topic list, click **Send Test Message** in the operation column of the target topic.
2. Enter the message key, tag and content in the pop-up window, and click **Submit**.

## Viewing subscribed groups

1. In the topic list, click the number under **Number of Subscribed Groups** of the target topic.
2. You will be redirected to the group list, which displays the information of the groups subscribed to the topic.

## Querying a topic

You can search for topics by topic name in the search box in the top-right corner of the **Topic** list page. TDMQ for RocketMQ will perform a fuzzy match and display the search results.

## Editing a topic

1. In the topic list, click **Edit** in the **Operation** column of the target topic.
2. In the pop-up window, you can edit the topic remarks.
3. Click **Submit** to complete your edits.

## Deleting a topic

**Batch deletion:** In the topic list, select all the topics that need to be deleted, and click **Batch Delete** in the upper left corner. Then, in the pop-up window, click **Submit** to complete the deletion.

**Single deletion:** In the topic list, find the topic that needs to be deleted, and click **Delete** in the **Operation** column. Then, in the pop-up window, click **Submit** to complete the deletion.

### Note

After a topic is deleted, all unconsumed messages retained in it will be cleared; therefore, proceed with caution.

# Group Management

Last updated : 2023-05-16 11:13:47

## Overview

A group is used to identify a type of consumers, which usually consume the same type of messages and use the same message subscription logic.

This document describes how to create, delete, and query a queue in the TDMQ for RocketMQ console.

## Prerequisites

You have created a namespace.

You have created a message producer and consumer based on the SDK provided by TDMQ, and they run properly.

## Directions

### Creating a group

1. Log in to the [TDMQ console](#), select the region, and click the ID of the target cluster to enter the cluster's basic information page.
2. Click the **Group** tab at the top, select a namespace, and click **Create** to enter the **Create Group** page.
3. Enter the group information.

### Create Group ✕

**i** There are currently 0 groups, and 10000 more can be created.

Current Namespace **test**

Group Name \*

This field is required. Please enter 3-64 letters, digits, or symbols ("-" or "\_").

Group Description

**Advanced Settings** ▲

Enable Consumption

If this option is disabled, all consumers in the group will stop consumption, but will resume consuming if it is enabled again.

Enable Broadcasting

If this option is disabled, all consumers using the broadcasting mode in the group will stop consumption, but will resume consuming if it is enabled again.

Group Name: enter the group name, which cannot be modified after creation and can contain 3–64 letters, digits, hyphens, and underscores.

Protocol Type: HTTP and TCP are supported.

Group Remarks: enter the group remarks.

4. Click **Submit**.

#### Note

In order to ensure the stability of the online cluster and avoid metadata redundancy in the console, TDMQ for RocketMQ has disabled the configuration of automatic group creation at the end of February 2023. When starting the consumer client, you need to create the corresponding group in the console first.

## Viewing consumer details

1. In the **group** list, click **Consumer Details** in the **Operation** column to enter the subscription list.
2. After expanding the list, you can view the group's basic information, client addresses, and subscriptions.

**Basic Info**

Group Name	group-364733	Creation Time	2022-03-11 15:13:15
Consumption Mode	Unknown	Client Protocol	TCP
Total Heaped Messages	0	Consumer Type	Unknown

**Client Address** Subscription

Client Address	Client Language	Client Version	Message Heap	Operation
No data yet				

Total items: 0      20 / page      1 / 1 page

### Basic info

Consumption Mode: cluster mode or broadcast mode.

Cluster consumption: if the cluster consumption mode is used, any message only needs to be processed by any consumer in the cluster.

Broadcast consumption: if the broadcast consumption mode is used, each message will be pushed to all registered consumers in the cluster to ensure that the message is consumed by each consumer at least once.

Client Protocol: TCP and HTTP are supported.

Message Retention: total number of retained messages.

Consumer Type: ACTIVELY or PASSIVELY.

Client Address: The list of connected clients. Click **View Details** in the **Operation** column of the client to view consumer details.

**Basic Info**

Group Name	group-364733	Creation Time	2022-03-11 15:13:15
Consumption Mode	Unknown	Client Protocol	TCP
Total Heaped Messages	0	Consumer Type	Unknown

**Client Address** Subscription

Topic	Type	Partition Count	Filter Condition	Subscription Consistency
No data yet				

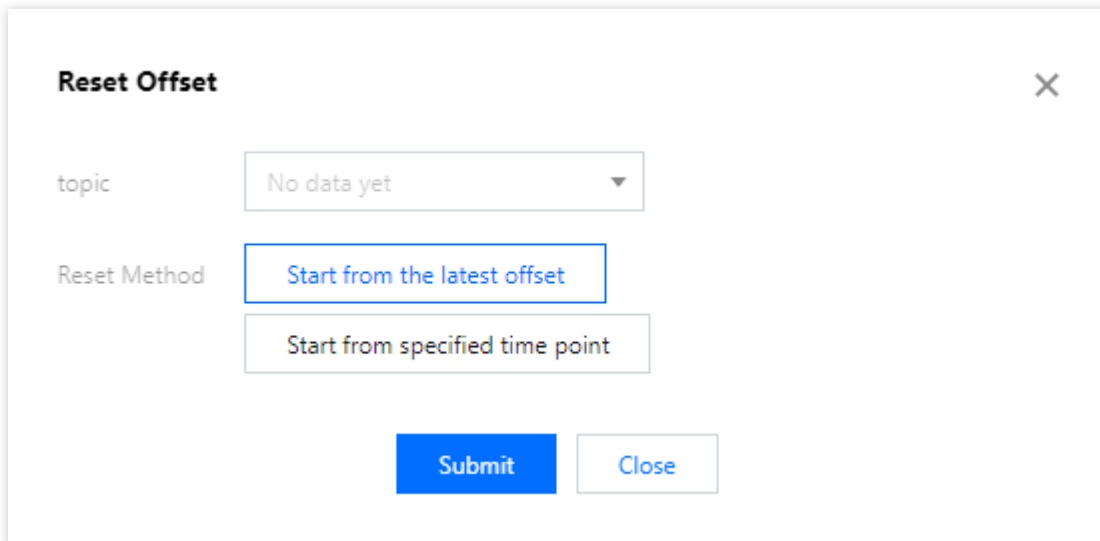
Total items: 0      20 / page      1 / 1 page

Subscription: The list of topics subscribed to by this group and the subscription attribute.

## Setting an offset

1. In the group list, click **Reset offset** in the **Operation** column of the target group.

- In the pop-up window, you can choose **the latest offset** or **the specified time point** to set the topic's **consumption offset** (that is, specify from where the consumers under the subscription start to consume messages).
- Click **Submit**.



**Reset Offset**

topic: No data yet

Reset Method:

- Start from the latest offset
- Start from specified time point

Submit Close

**Note**

TDMQ for RocketMQ supports resetting the consumption offset for offline groups only in the push consumption mode; otherwise, the reset will fail.

**Editing a group**

- In the group list, click **Edit** in the **Operation** column of the target group.
- In the pop-up window, edit the group information.
- Click **Submit**.

**Deleting a group**

**Batch deletion:** In the group list, select all the groups that need to be deleted, and click **Batch Delete** in the upper left corner. Then, in the pop-up window, click **Submit** to complete the deletion.

**Single deletion:** In the group list, find the group that needs to be deleted, and click **Delete** in the **Operation** column. Then, in the pop-up window, click **Submit** to complete the deletion.

**Note**

After a group is deleted, consumers identified by the group will immediately stop receiving messages, and all the configurations under it will be cleared and cannot be recovered; therefore, caution should be exercised with this operation.



# Role and Authentication

Last updated : 2023-05-16 10:38:37

## Glossary

**Role:** different from a role in Tencent Cloud, a role in TDMQ is a proprietary concept. It is the smallest unit of permission division performed by you in TDMQ. You can add multiple roles and assign them the production/consumption permissions of different namespaces.

**Token:** it is an authentication tool in TDMQ. You can add a token in a client to access TDMQ for message production/consumption. Tokens correspond to roles one by one, and each role has its own unique token.

TDMQ for RocketMQ inherits these concepts. In TDMQ for RocketMQ, the `ACL_ACCESS_KEY` defined in the open-source client corresponds to the TDMQ token.

## Use Cases

You need to securely use TDMQ to produce/consume messages.

You need to set production/consumption permissions of different namespaces for different roles.

For example, your company has departments A and B, and department A's system produces transaction data and department B's system performs transaction data analysis and display. In line with the principle of least privilege, two roles can be configured to grant department A only the permission to produce messages to the transaction system namespace and grant department B only the permission to consume messages. This helps greatly avoid problems caused by unclear division of permissions, such as data disorder and dirty business data.

## Directions

### Creating a role

1. Log in to the [TDMQ console](#) and click [Role Management](#) on the left sidebar to enter the **Role Management** page.
2. On the **Role Management** page, select the target cluster and click **Create** to enter the **Create Role** page.
3. On the **Create Role** page, enter the role name and remarks:  
Role Name: it can contain up to 32 digits, letters, and delimiters (underscore or hyphen).  
Remarks (optional): enter remarks of up to 100 characters.
4. Click **Submit**.

### Create

Region: Shanghai

Role:

It can contain up to 32 digits, letters, or symbols ("\_" or "-").

Description:

It can contain up to 100 characters.

[Save](#) [Cancel](#)

## Granting permission to role

1. Find the newly created role in [Role Management](#) in the TDMQ console and copy the role token in the following methods:

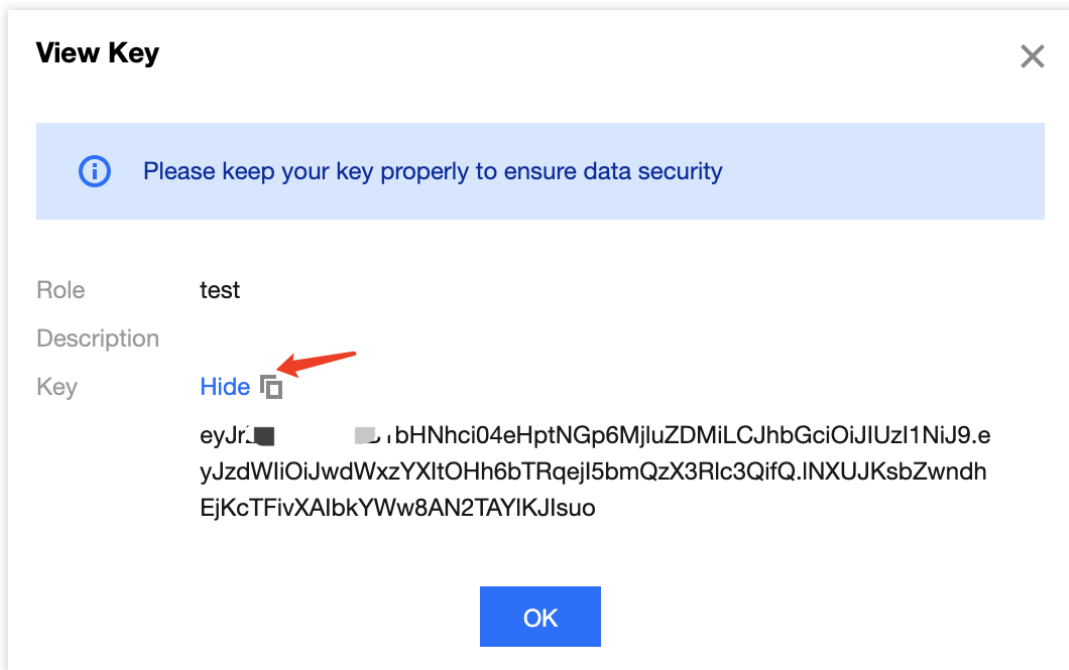
Option 1: Copy the token from the token column

Option 2: View and copy the token in the operation column

Click **Copy** in the token column.

Name	Key	Description	Creation Time	Last Updated	Operation
<input type="checkbox"/> test	<a href="#">Copy</a>		2021-11-25 16:47:32	2021-11-25 16:47:32	<a href="#">View Key</a> <a href="#">View Permission</a> <a href="#">Delete</a>

Click **View Token** in the **Operation** column and click **Copy** in the pop-up window.

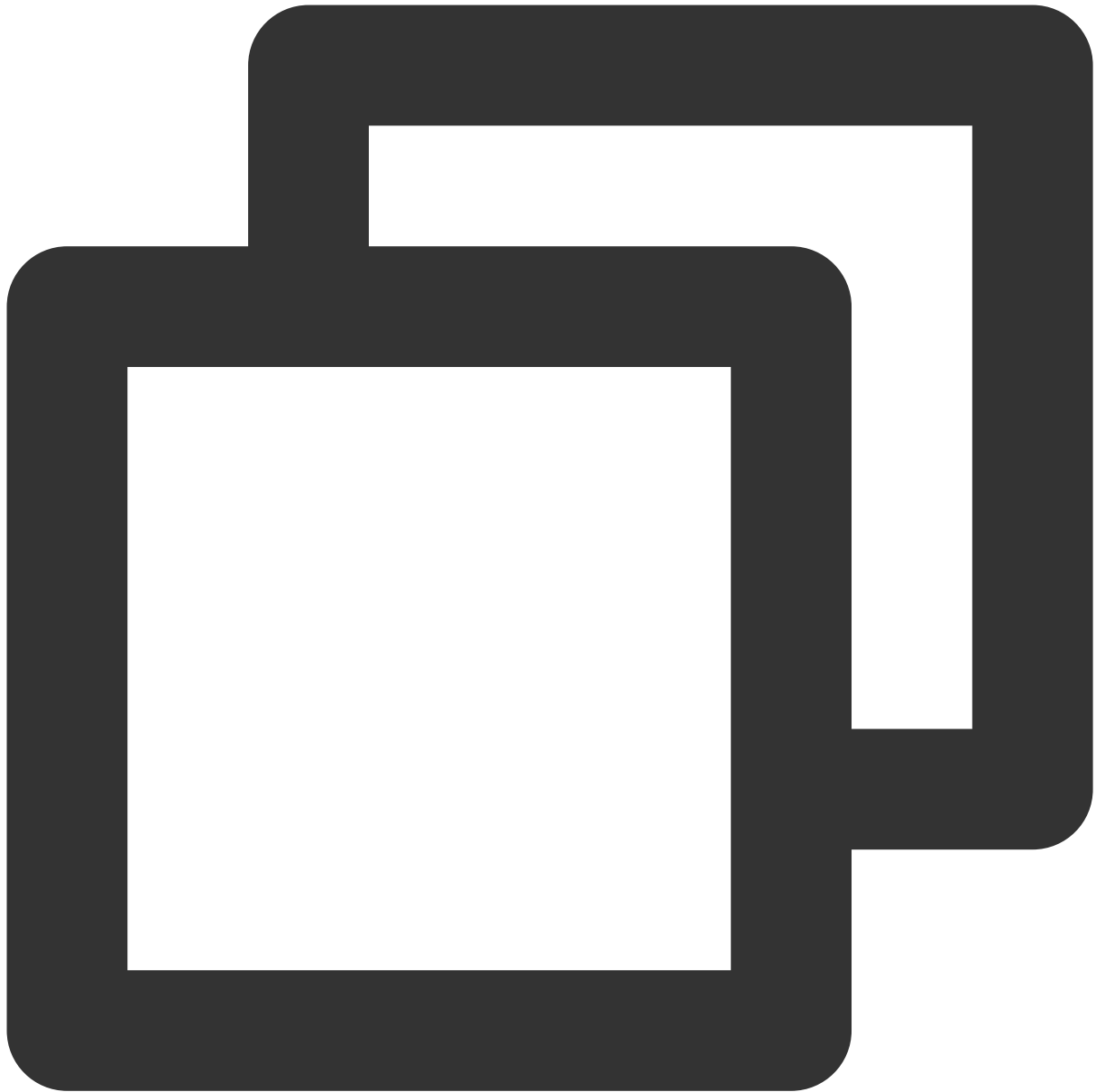
**Note**

Token leakage may lead to data leakage; therefore, you should keep your token confidential.

2. Add the copied role token to the client parameters. For directions on how to add the token parameter to the client code, see the [sample code](#) of RocketMQ.

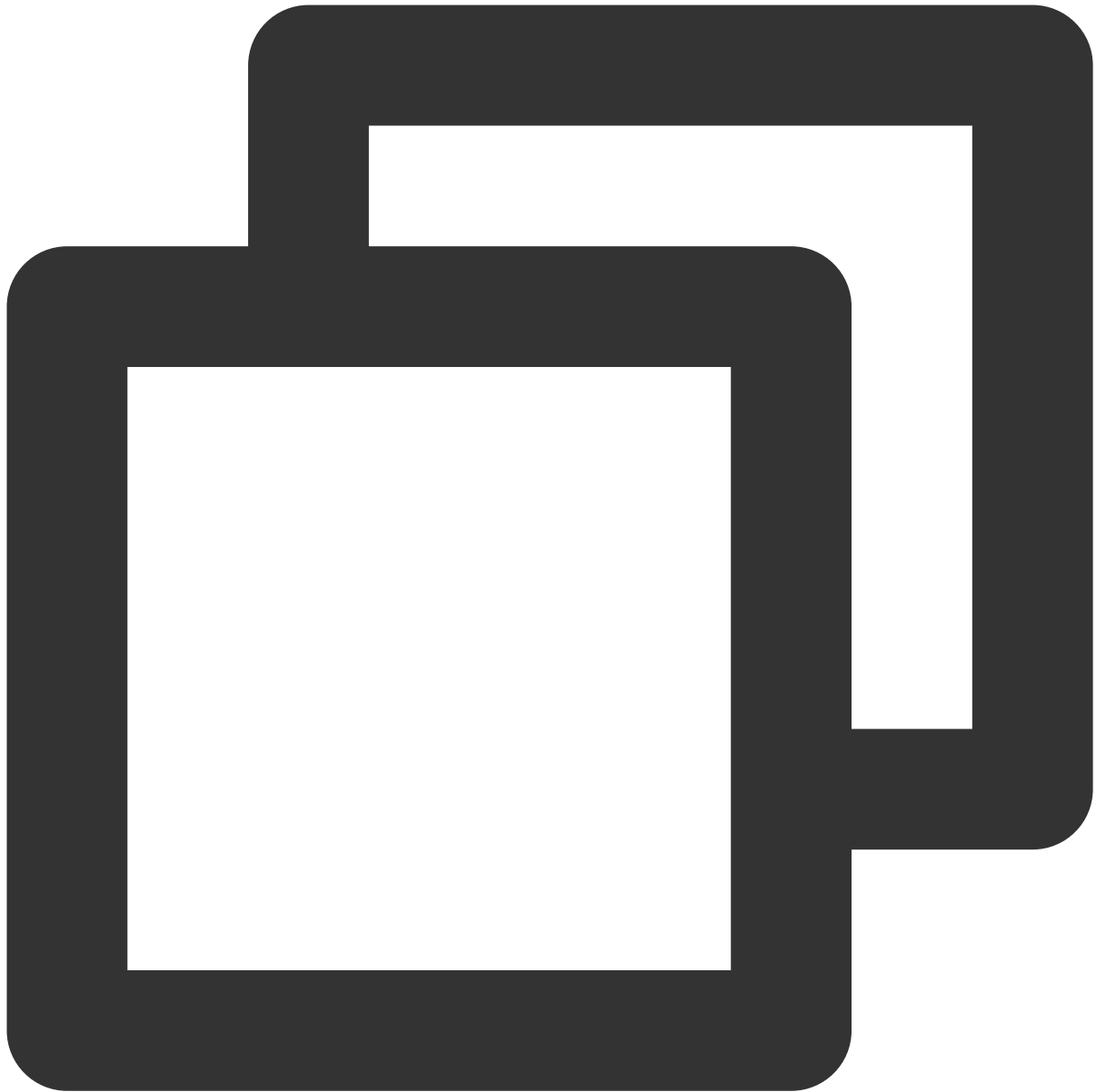
A recommended way is given below:

2.1 Declare two fields `ACL_SECRET_KEY` and `ACL_SECRET_ACCESS`. If you use various frameworks, we recommend that you read them from the configuration file.



```
private static final String ACL_ACCESS_KEY = "eyJr****";  
private static final String ACL_SECRET_KEY = "bigdata"; //Use the role name on the
```

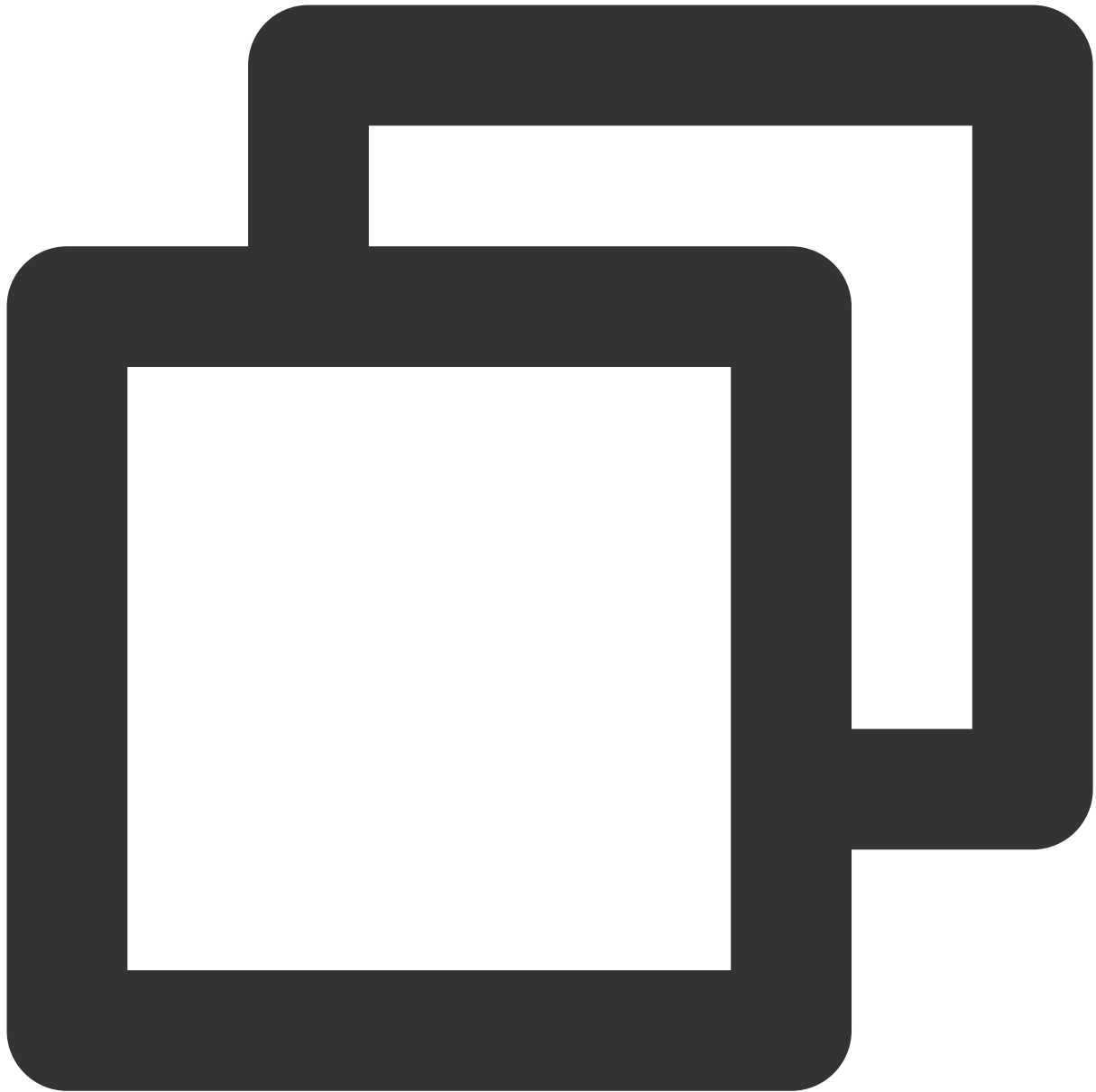
2.2 Declare a static function to load a `RPCHook` object of the RocketMQ client.



```
static RPCHook getAclRPCHook() {  
    return new AclClientRPCHook(new SessionCredentials(ACL_ACCESS_KEY, ACL_SECRET_K  
    }  
}
```

2.3 When creating RocketMQ's `producer` , `pushConsumer` , or `pullConsumer` , import the `RPCHook` object.

Below is the sample code for creating a `producer` :



```
DefaultMQProducer producer = new DefaultMQProducer("rocketmq-mw***|namespace", "Pro
```

3. After selecting the cluster with the previously set role in the TDMQ for RocketMQ console, switch to the **Namespace** page, select a namespace for which to configure production and consumption permissions, and click **Configure Permissions** in the **Operation** column.

Namespace Name	Message Retention Period ⓘ	Description	Oper
test rocketmq-8x3mzbkvmz8 test	1 day		Confi

4. Click **Add Role**, find the role just created in the drop-down list, select the required permission, and click **Save**.

### Create

Role

Unable to find a role? Please configure a role and key on the [Role Management](#) page.

Permission  Message production  
 Message consumption

For more permission type information, see [here](#)

**Save**

5. Check whether the permission has taken effect.

You can run the configured client to access the topic resources in the namespace and produce/consume messages according to the configured permission. Check whether a no permission error is reported, and if not, the permission has been configured successfully.

## Editing a permission

1. In **Namespace** in the TDMQ for RocketMQ console, find the target namespace and click **Configure Permission** in the **Operation** column to enter the permission configuration list.
2. In the permission configuration list, click **Edit** in the **Operation** column of the target role.
3. In the pop-up window, modify the permission information and click **Save**.

## Deleting a permission

### Note

Before deleting a permission, make sure that the current business no longer uses the role to produce/consume messages; otherwise, a client exception may occur due to the failure to produce/consume messages.

A role cannot be deleted if it has permissions configured in namespaces.

1. In **Namespace** in the TDMQ for RocketMQ console, find the target namespace and click **Configure Permission** in the **Operation** column to enter the permission configuration list.
2. In the permission configuration list, click **Delete** in the **Operation** column of the target role.

3. In the pop-up window, click **OK**.



# Access Management (CAM)

## Access Authorization for Root Account

Last updated : 2023-09-12 16:44:28

### Overview

As TDMQ for RocketMQ needs to access APIs of other Tencent Cloud products, you need to create service roles and grant those roles to TDMQ for RocketMQ.

### Prerequisites

You have signed up for a Tencent Cloud account. For more information, see [Signing Up](#).

#### Note

After you sign up, the system will create a root account for you by default, which is used to quickly access Tencent Cloud resources.

### Directions

1. Log in to the [TDMQ console](#), select **RocketMQ > Cluster** on the left sidebar, and click **Create Cluster** to enter the purchase page.
2. Configure the network on the purchase cluster page. After selecting a VPC, check **I authorize the "binding of the access point domain name of the newly purchased cluster to the above VPCs"**, and a prompt will pop up to require your authorization.

**Network Configuration**

VPC ⓘ

I authorize the "binding of the access point domain name of the newly purchased cluster to the above VPCs"

Public Network Access

Extra fees will be incurred after public network bandwidth is enabled, and you can disable this feature on the cluster management page. For details, see [here](#)

**Authorization is required for this feature** ✕

To use the "binding of the access point domain name of the newly purchased cluster to the above VPCs" feature, you need to allow **Tencent Distributed Message Queue** to access your authorized resources as a service role. Please click "Authorize Now" to grant permissions to the related service APIs of **Tencent Distributed Message Queue**.

[Authorize Now](#)[Cancel](#)

3. Click **Authorize** to enter the CAM console for authorization. Click **Grant**, and TDMQ for RocketMQ will be assigned a service role to access your other resources.

**Role Management****Service Authorization**

After you agree to grant permissions to **Tencent Distributed Message Queue**, a preset role will be created and relevant permissions will be granted to **Tencent Distributed Message Queue**

Role Name TDMQ\_QCSLinkedRoleInTDMQRocketMQVPCdomainbinding

Role Type Service-Linked Role

Description The current role is the TDMQ service linked role, which will access your other service resources within the scope of the permissions of the associated policy.

Authorized Policies Preset Policy QcloudAccessForTDMQLinkedRoleInTDMQRocketMQVPCdomainbinding ⓘ

[Grant](#)[Cancel](#)

4. After the authorization is completed, you can continue to create a TDMQ for RocketMQ cluster and use related services.

# Access Authorization for Sub-Accounts

## Granting Sub-Account Access to TDMQ for RocketMQ

Last updated : 2023-10-19 10:45:00

### Basic CAM Concepts

The root account authorizes sub-accounts by associating policies. The policy setting can be specific to the level of **\*\* [API, Resource, User/User Group, Allow/Deny, and Condition]\*\***.

#### Account system

**Root account:** It owns all the resources in Tencent Cloud and can access any of these resources.

**Sub-account:** It includes sub-users and collaborators.

**Sub-user:** It is created and fully owned by a root account.

**Collaborator:** It has the identity of a root account. After it is added as a collaborator of the current root account, it becomes one of the sub-accounts of the current root account and can switch back to its root account identity.

**Identity credential:** It includes login credentials and access certificates. **Login credential** refers to a user's login name and password. **Access certificate** refers to TencentCloud API keys (SecretId and SecretKey).

#### Resource and permission

**Resource:** It refers to an object operated in Tencent Cloud services, such as a CVM instance, a COS bucket, or a VPC instance.

**Permission:** It is used to allow or deny some user operations. By default, **the root account has full access to all its resources**, while **a sub-account does not have access to any resources under its root account**.

**Policy:** It is a syntax rule used to define and describe one or more permissions. The **root account** can authorize users or user groups by **associating them with policies**.

## Allowing Sub-Accounts to Use TDMQ for RocketMQ

To allow a sub-account to use TDMQ for RocketMQ, the root account needs to authorize the sub-account.

Log in to the [CAM console](#) as a root account, select the target sub-user in the user list, and click **Authorize** in the **Operation** column.

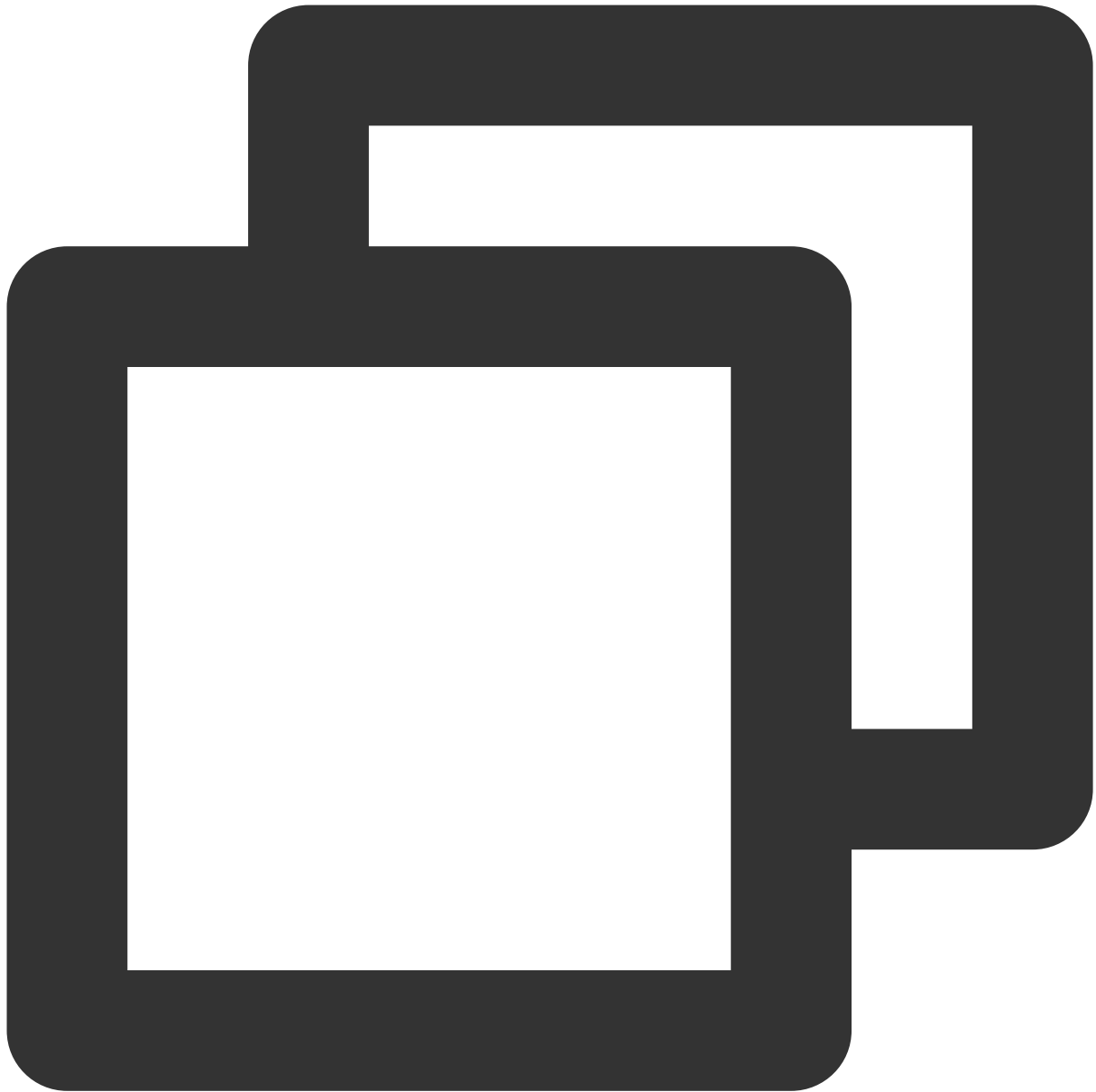
TDMQ for RocketMQ provides two preset policies for sub-accounts: `QcloudTrocketReadOnlyaccess` and `QcloudTrocketFullAccess`. The former only allows sub-accounts to view information in the console while the

latter allows sub-accounts to perform read or write operations in the console.

In addition to the preset policies, the root account can authorize sub-accounts to call the APIs of other Tencent Cloud products as needed. TDMQ for RocketMQ may require the API permissions of the following Tencent Cloud products:

Tencent Cloud Product	API Name	Description	Purpose in TDMQ for RocketMQ
Tencent Cloud Observability Platform (TCOP)	GetMonitorData	Queries the monitoring data of metrics.	Queries the monitoring data displayed in the console.
TCOP	DescribeDashboardMetricData	Queries the monitoring data of metrics.	Queries the monitoring data displayed in the console.
Tencent Cloud Tag	DescribeResourceTagsByResourceIds	Queries resource tags.	Queries the resource tags of a cluster.

To grant the above permissions to the sub-account, the root account needs to go to the [CAM console](#), and click **Create Custom Policy** on the **Policies** page. In the pop-up window, select **Create by Policy Syntax > Blank Template**, and enter the following policy syntax.



```
{
  "version": "2.0",
  "statement": [
    {
      "effect": "allow",
      "action": [
        "monitor:GetMonitorData",
        "monitor:DescribeDashboardMetricData",
        "tag:DescribeResourceTagsByResourceIds"
      ],
      "resource": [
```

```
        "*"
      ]
    }
  ]
}
```

After the policy is created, return to the policy list and associate it with the sub-account in the **Operation** column.

## Relevant Documentation

For more information, see the following documents:

[Granting Operation-Level Permissions to Sub-Accounts](#)

[Granting Resource-Level Permissions to Sub-Accounts](#)

[Granting Tag-Level Permissions to Sub-Accounts](#)

# Granting Operation-Level Permissions to Sub-Accounts

Last updated : 2023-09-12 16:58:34

## Overview

This document describes how to use the Tencent Cloud root account to authorize sub-accounts at the operation level. You can grant different read and write permissions to sub-accounts as needed.

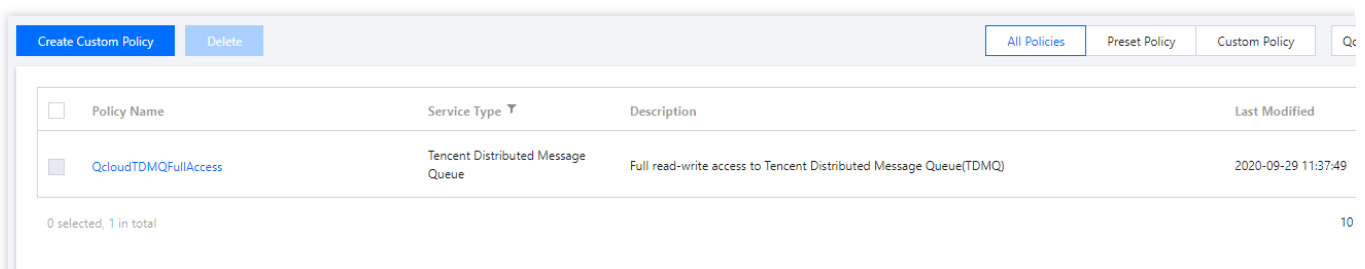
## Directions

### Full access permission

#### Note

After granting full access permissions to a sub-account, the sub-account will have **full read and write capabilities** to **all resources** under the root account.

1. Log in to the [CAM Console](#) with the root account.
2. In the left sidebar, click **Policies** to go to the policy management page.
3. Search for **QcloudTDMQFullAccess** on the right.



4. In the search results, click the **Associated Users/Groups** of **QcloudTDMQFullAccess** and select the sub-account to be authorized.

### Associate User/User Group/Role

Select Users (35 Total) (2) selected

Support multi-keyword search by user name/ID/SecretId/mobi

**User** Switch to User Group ...

<input type="checkbox"/>	[blurred]	User
<input checked="" type="checkbox"/>	[blurred]	User
<input checked="" type="checkbox"/>	[blurred]	User
<input type="checkbox"/>	[blurred]	User
<input type="checkbox"/>	[blurred]	User
<input type="checkbox"/>	[blurred]	User

Support for holding shift key down for multiple selection

Name	Type
[blurred]	User
[blurred]	User

OK
Cancel

5. Click **OK** to complete the authorization, which will be displayed in the **Policy List** of the user.

**Permission** | Service | Group (0) | Security ⓘ | API Key | Tag Policy

▼ **Permissions Policy**

ⓘ Associate a policy to get the action permissions that the policy contains. Disassociating a policy will result in losing the action permissions in the policy. A policy inherited from a use group can be disassociated only by removing the user group.

Associate Policy Disassociate Policy

Search for policy

<input type="checkbox"/>	Policy Name	Description	Association Type	Policy Type	Associa
<input type="checkbox"/>	[blurred]	[blurred]	Associated directly	Preset Policy	2023-08
<input type="checkbox"/>	QcloudTDMQFullAccess	Full read-write access to Tencent Distributed Messag...	Associated directly	Preset Policy	2023-08

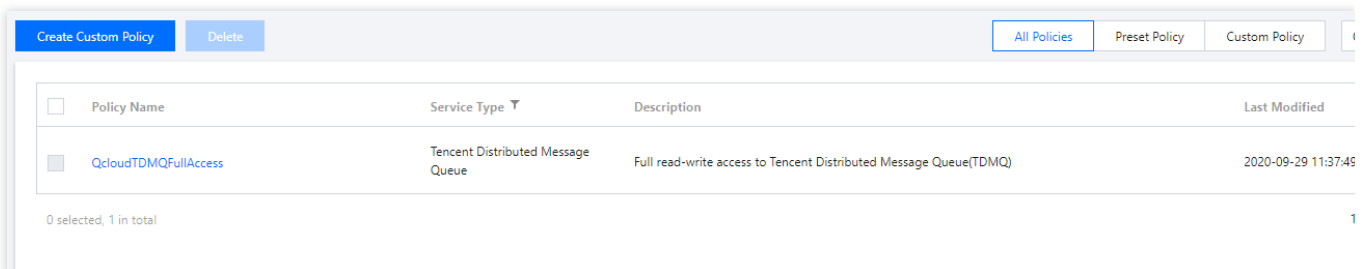
## Read-only permission

### Note

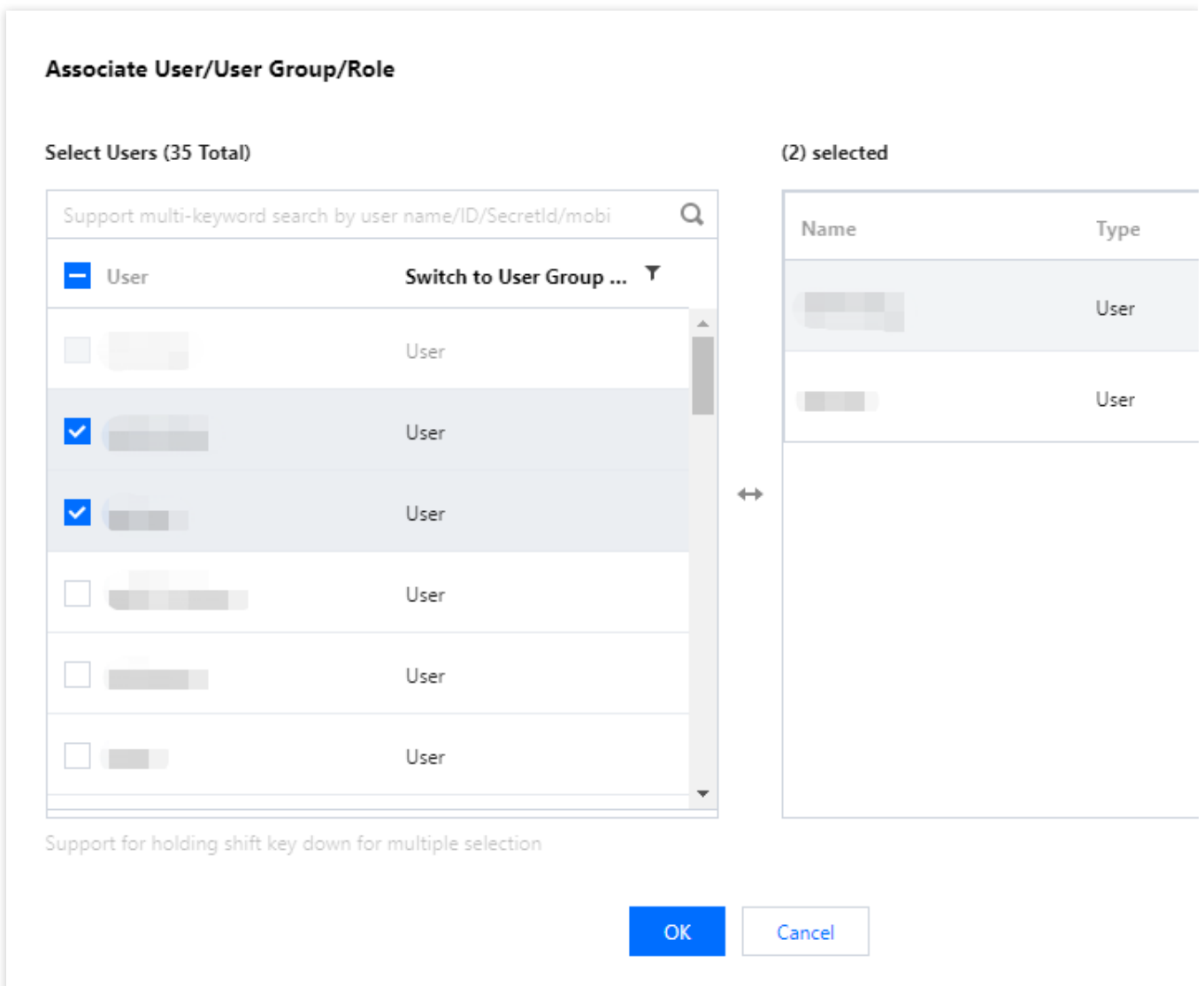
After granting the read-only permission to a sub-account, the sub-account will have **read-only capability to all resources** under the root account.



1. Log in to the [CAM Console](#) with the root account.
2. In the left sidebar, click **Policies** to go to the policy management page.
3. Search for **QcloudTDMQReadOnlyAccess** on the right.



4. In the search results, click the **Associated Users/Groups** of **QcloudTDMQReadOnlyAccess** and select the sub-account to be authorized.



5. Click **OK** to complete the authorization, which will be displayed in the **Policy List** of the user.

[Permission](#)
[Service](#)
[Group \(0\)](#)
[Security !](#)
[API Key](#)
[Tag Policy](#)

▼ **Permissions Policy**

ℹ Associate a policy to get the action permissions that the policy contains. Disassociating a policy will result in losing the action permissions in the policy. A policy inherited from a use group can be disassociated only by removing

[Associate Policy](#)
[Disassociate Policy](#)

<input type="checkbox"/>	Policy Name	Description	Association Type <span>▼</span>	Policy Type <span>▼</span>	Association T
<input type="checkbox"/>	[Redacted]	[Redacted]	Associated directly	Preset Policy	2023-08-17 1
<input type="checkbox"/>	QcloudTDMQFullAccess	Full read-write access to Tencent Distributed Messag...	Associated directly	Preset Policy	2023-08-17 1

## Other methods

[Resource-Level Authorization](#)

[Tag-Level Authorization](#)

# Granting Resource-Level Permissions to Sub-Accounts

Last updated : 2023-10-13 10:37:11

## Overview

This document describes how to use the root account to authorize sub-accounts at the resource level. After successful authorization, the sub-accounts will have the capability to control a certain resource.

## Prerequisites

You must have a Tencent Cloud root account and have activated the Cloud Access Management (CAM) service. Your root account must have at least one sub-account, and you have completed the authorization as instructed in [Access Authorization for Sub-Accounts](#).

You must have at least one TDMQ for RocketMQ cluster instance.

## Directions

By using the policy feature in the CAM console, you can grant a sub-account access to the TDMQ for RocketMQ resources owned by the root account. Taking cluster resource as an example, the following describes the detailed steps for **granting the sub-account access to TDMQ for RocketMQ resources**, which also apply to other types of resources.

### Step 1. Obtain the TDMQ for RocketMQ cluster ID

1. Log in to the [TDMQ for RocketMQ console](#) with **root account**, select an existing cluster instance, and click it to enter the details page.

Cluster ID/Name	Topics	Group Count	Cluster Status	Resource Tag	Cluster Description
rocketmq-test	Used: 1 Capacity: 150	Used: 2 Capacity: 1500	Running		

Total items: 1      20 / page

2. In **Basic Info**, the field **ID** indicates the ID of the current TDMQ for RocketMQ cluster.

Cluster / test

**Basic Info**   Cluster Monitoring   Namespace   Topic   Group

**Cluster Overview**

Topics	Currently Heaped Messages	API Calls This Month
1	-	0

**Basic Info**   [Edit](#)

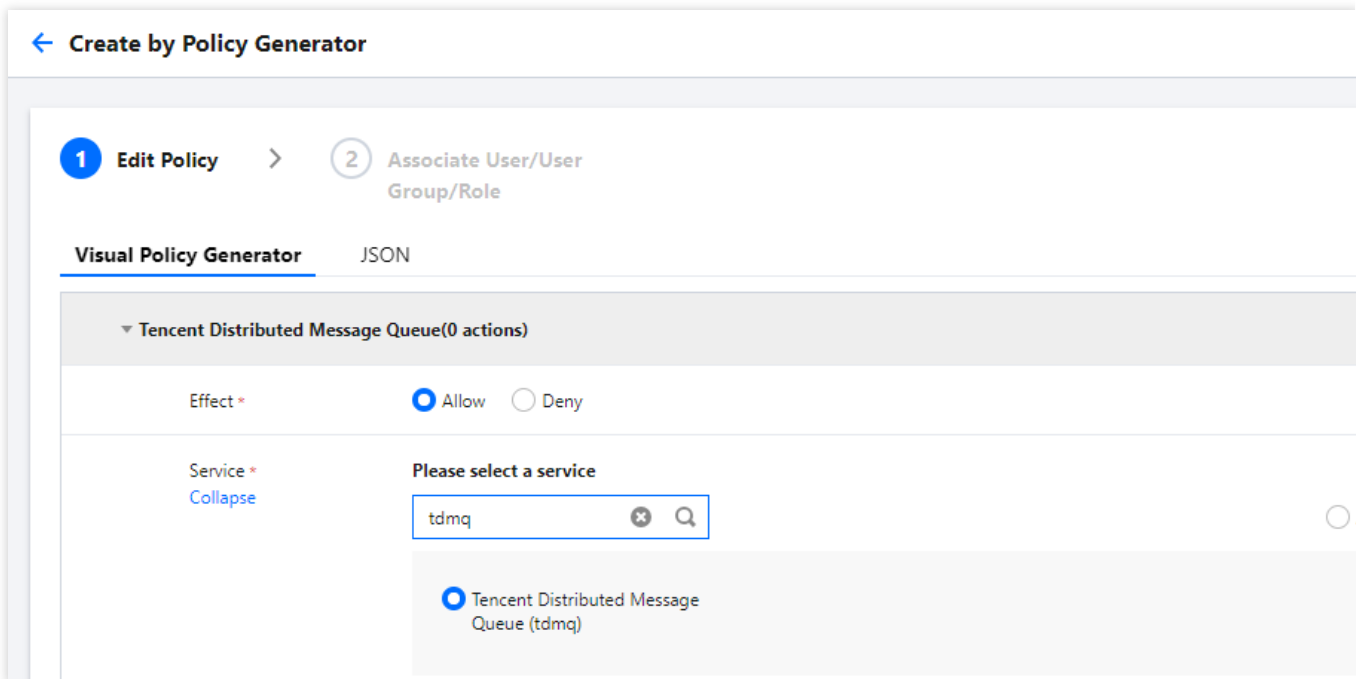
Name	test
ID	rocketmq-test-xxxxxx
Region	South China(Guangzhou)
Creation Time	2022-03-10 10:45:37
Description	-
Resource Tag	No tag

**Instance Configuration**

Max Cluster TPS	4000
Max Namespaces	10 (1 used/10)
Max Topics	150 (1 used/150)
Max Groups	1500 (2 used/1500)
Max Retention Period	3 days

## Step 2. Create a new authorization policy

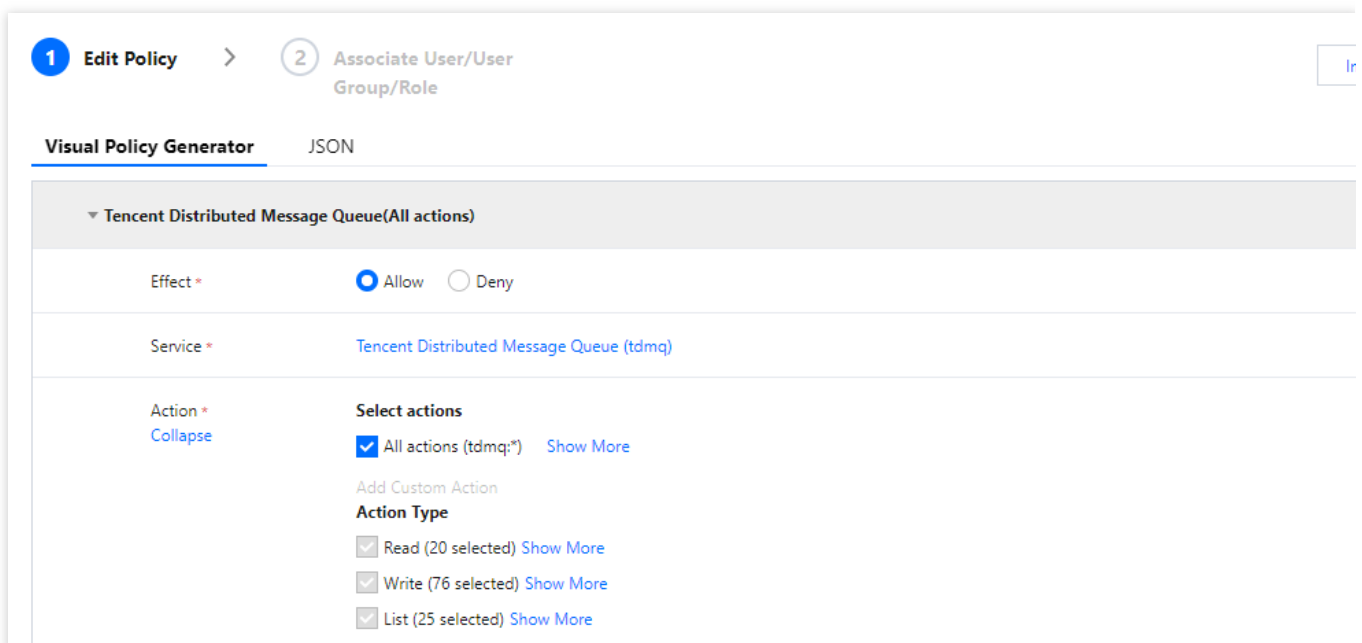
1. Log in to the [CAM console](#) and click [Policies](#) on the left sidebar.
2. Click **Create Custom Policy** > **Create by Policy Generator**.
3. In the visual policy generator, select **Allow** for **Effect**, enter "TDMQ" in **Service** to filter, and select **\*\*Tencent Distributed Message Queue (tdmq)\*\***.



4. Select **All actions** in **Action**, and you can also select the action type as needed.

#### Note

Currently, some APIs don't support resource authentication, which is as displayed in the console page. For the list of APIs that support resource-level authorization, see the list of APIs supporting resource-level authorization in the appendix.



5. In the **Resource** field, select **Specific resources**, find the **cluster** resource type, and you can select **Any resource of this type** on the right to authorize all cluster resources, or click **Add a six-segment resource description** to authorize specific cluster resources.

6. If you click **Add a six-segment resource description**, enter the **cluster ID** for **Resource** in the pop-up dialog box. For how to obtain the cluster ID, see [Step 1](#).

namespace	Specify a namespace six-segment resource description for CreateRocketMQGr <input type="checkbox"/> Any resource of this type <a href="#">Add a six-segment resource description</a> to restrict the access.	<b>Add a six-segment re</b>  Six-segment resource des Tencent Cloud resource o qcs:tdmq:uin/2000184  Service * <input type="text" value="td"/> Region * <input type="text" value="All"/> Account * <input type="text" value="uin"/> Resource Prefix * <input type="text" value="clu"/> Resource * <input type="text"/>
group	Specify a group six-segment resource description for DescribeRocketMQCons action(s). ⓘ <input type="checkbox"/> Any resource of this type <a href="#">Add a six-segment resource description</a> to restrict the access.	
exchange	Specify a exchange six-segment resource description for DeleteAMQPExchange <input type="checkbox"/> Any resource of this type <a href="#">Add a six-segment resource description</a> to restrict the access.	
environmentRoles	Specify a environmentRoles six-segment resource description for DescribeEnvi <input type="checkbox"/> Any resource of this type <a href="#">Add a six-segment resource description</a> to restrict the access.	
environmentRole	Specify a environmentRole six-segment resource description for CreateEnviron <input type="checkbox"/> Any resource of this type <a href="#">Add a six-segment resource description</a> to restrict the access.	
environmentId	Specify a environmentId six-segment resource description for DescribeEnviron <input type="checkbox"/> Any resource of this type <a href="#">Add a six-segment resource description</a> to restrict the access.	
environment	Specify a environment six-segment resource description for DescribeRocketMQ ⓘ <input type="checkbox"/> Any resource of this type <a href="#">Add a six-segment resource description</a> to restrict the access.	
dlq	Specify a dlq six-segment resource description for DescribeCmqDeadLetterSou <input type="checkbox"/> Any resource of this type <a href="#">Add a six-segment resource description</a> to restrict the access.	
consumer	Specify a consumer six-segment resource description for ResetRocketMQCons <input type="checkbox"/> Any resource of this type <a href="#">Add a six-segment resource description</a> to restrict the access.	
cmqtopic	Specify a cmqtopic six-segment resource description for DescribeCmqTopicDet <a href="#">Add a six-segment resource description</a> to restrict the access.	
cmqueue	Specify a cmqueue six-segment resource description for DescribeCmqQueue <input type="checkbox"/> Any resource of this type <a href="#">Add a six-segment resource description</a> to restrict the access.	
cluster	Specify a cluster six-segment resource description for DescribeAMQPCluster ar <input type="checkbox"/> Any resource of this type <a href="#">Add a six-segment resource description</a> to restrict the access.  <a href="#">Add a six-segment resource description</a> to restrict the access	
Condition	<input type="checkbox"/> Source IP ⓘ <a href="#">Add other conditions</a>	
<a href="#">+ Add Permissions</a>		
<input type="button" value="Next"/>	Characters: 161(up to 6,144)	<input type="button" value="OK"/>

7. Click **Next** and enter a policy name as needed.

8. Click **Select Users** or **Select User Groups** to select the users or user groups that need to be granted resource permissions.

**1** Edit Policy > **2** Associate User/User Group/Role

**Basic Info**

Policy Name \*  After the policy is created, its name cannot be modified.

Description

**Associate User/User Group/Role**

Authorized Users [Select Users](#)

Authorized User Groups [Select User Groups](#)

Grant Permission to Role [Select role](#)

[Previous](#) [Complete](#)

9. Click **Complete**. The sub-account with granted resource permissions will have the capability to access related resources.

## Other authorization methods

[Operation-Level Authorization](#)

[Tag-Level Authorization](#)

# Appendix

## List of APIs supporting resource-level authorization

TDMQ supports resource-level authorization. You can grant a specified sub-account the API permission of a specified resource. APIs supporting resource-level authorization include:

API Name	Description	Resource Type	Six-Segment Resource Example
ResetRocketMQConsumerOffset	Resets RocketMQ consumption offset	consumer	qcs::tdmq:\${region}:uin/\${uin}:
DescribeRocketMQClusters	Gets the list of RocketMQ clusters	cluster	qcs::tdmq:\${region}:uin/\${uin}:
DeleteRocketMQCluster	Deletes a RocketMQ cluster	cluster	qcs::tdmq:\${region}:uin/\${uin}:
DescribeRocketMQCluster	Gets the information of a RocketMQ cluster	cluster	qcs::tdmq:\${region}:uin/\${uin}:
CreateRocketMQNamespace	Creates a RocketMQ namespace	cluster	qcs::tdmq:\${region}:uin/\${uin}:
ModifyRocketMQNamespace	Updates a RocketMQ namespace	namespace	qcs::tdmq:\${region}:uin/\${uin}:
DeleteRocketMQNamespace	Deletes a RocketMQ namespace	namespace	qcs::tdmq:\${region}:uin/\${uin}:
CreateRocketMQGroup	Creates a RocketMQ consumer group	namespace	qcs::tdmq:\${region}:uin/\${uin}:
ModifyRocketMQGroup	Updates a RocketMQ	group	qcs::tdmq:\${region}:uin/\${uin}:



	consumer group		
DescribeRocketMQGroups	Gets the list of RocketMQ consumer groups	group	qcs::tdmq:\${region}:uin/\${uin}:
DeleteRocketMQGroup	Deletes a RocketMQ consumer group	group	qcs::tdmq:\${region}:uin/\${uin}:
CreateRocketMQTopic	Creates a RocketMQ topic	namespace	qcs::tdmq:\${region}:uin/\${uin}:
ModifyRocketMQTopic	Updates RocketMQ topic information	topic	qcs::tdmq:\${region}:uin/\${uin}:
DeleteRocketMQTopic	Deletes a RocketMQ topic	topic	qcs::tdmq:\${region}:uin/\${uin}:
DescribeRocketMQTopics	Gets the list of RocketMQ topics	topic	qcs::tdmq:\${region}:uin/\${uin}:
DescribeRocketMQTopicsByGroup	Gets the list of topics subscribed to a specified consumer group	topic	qcs::tdmq:\${region}:uin/\${uin}:
DescribeRocketMQConsumerConnections	Gets the current client connection status under a specified	group	qcs::tdmq:\${region}:uin/\${uin}:

	consumer group		
DescribeRocketMQConsumerConnectionDetail	Gets the details of online consumers	group	qcs::tdmq:\${region}:uin/\${uin}:
ModifyRocketMQCluster	Modifies RocketMQ cluster information	cluster	qcs::tdmq:\${region}:uin/\${uin}:

## List of APIs not supporting resource-level authorization

API Name	Description	Six-Segment Resource
CreateRocketMQCluster	Creates a RocketMQ cluster	*

# Granting Tag-Level Permissions to Sub-Accounts

Last updated : 2023-09-22 09:40:03

## Overview

This document describes how to use the root account to authorize sub-accounts at the tag level. After successful authorization, the sub-accounts will have the capability to control a certain resource under the authorized tag.

## Prerequisites

You must have a Tencent Cloud root account and have activated the Cloud Access Management (CAM) service. Your root account must have at least one sub-account, and you have completed the authorization as instructed in [Access Authorization for Sub-Accounts](#).

You must have at least one TDMQ for RocketMQ cluster instance.

You must have at least one **tag**, if you don't have one, you can go to the [Tag console](#) > **Tag List** to create a new one.

## Directions

By using the policy feature in the CAM console, you can grant a sub-account full access to the tagged TDMQ for RocketMQ resources owned by the root account through the tag authorization. The following describes the detailed steps for **granting the sub-account access to CKafka resources by tag**

### Step 1. Bind tags to resources

1. Log in to the [TDMQ for RocketMQ console](#) and enter the **Cluster** page.

**Cluster** Guangzhou 1 Exclusive Cluster Virtual Cluster TDMQ for RocketMQ User C

Create Cluster (1/50) Edit Resource Tag Search by keyword

<input type="checkbox"/>	Cluster ID/Name	Topics	Group Count	Cluster Status	Resource Tag	Cluster Descri
<input type="checkbox"/>	rocketmq-	Used: 1 Capacity: 150	Used: 2 Capacity: 1500	Running		

Total items: 1 20 / page

2. Select the target cluster, click **Edit Tag** in the upper left corner, and bind the resource tag to the instance.

**Edit Tag** ×

Tags are used to manage resources by category in different dimensions. If the existing tags don't meet your requirements, you can [manage tags](#).

1 resource(s) selected

tag\_44459 num33453 ×

+ Add

OK Cancel

## Step 2. Authorize by Tag

1. Log in to the [CAM console](#) and click **Policies** on the left sidebar.
2. Click **Create Custom Policy > Authorize by Tag**.
3. In the visual policy generator, enter "tdmq" in **Service** to filter, and select **Tencent Distributed Message Queue (tdmq)**. Then, select All actions in **Action**, and you can also select the action type as needed.

### Note

Currently, some APIs don't support tag authentication, which is as displayed in the console page.

← **Authorize by Tag**

1 **Edit Policy** > 2 **Associate User/User Group/Role**

**Visual Policy Generator** JSON

**Add Services and Operations** Add

▼ Tencent Distributed Message Queue(All actions)	
Service *	Tencent Distributed Message Queue (tdmq)
Action *	All actions (*)

**Select Tag (resource\_tag)** ⓘ

tag\_44459 num33453 ×

+ Add

If existing tags do not meet your requirements, [create one](#) in the console.

**Next** Characters: 3561(up to 6,144)

4. Click **Next** and enter a policy name as needed.

5. Click **Select Users** or **Select User Groups** to select the users or user groups that need to be granted resource permissions.

The screenshot shows the 'Associate User/User Group/Role' step in the 'Edit Policy' process. It includes a 'Basic Info' section with fields for 'Policy Name' (containing 'policyger') and 'Description'. Below this is the 'Associate User/User Group/Role' section, which contains three rows of options: 'Authorized Users' with a 'Select Users' link, 'Authorized User Groups' with a 'Select User Groups' link, and 'Grant Permission to Role' with a 'Select role' link. The first two rows are highlighted with a red box. At the bottom, there are 'Previous' and 'Complete' buttons.

6. Click **Complete**. The sub-account can control the resources under the specified tag according to the policy.

## Managing Resource Tags

You can also manage resource tags in a unified manner in the [Tag console](#). The detailed operations are as follows.

1. Log in to the [Tag console](#).
2. Select **Resource Tag** in the left navigation bar, select query conditions as needed, and select **Tencent Distributed Message Queue > Cluster** in **Resource type**.
3. Click **Query Resources**.
4. Select the required resources in the result and click **Edit Tag** to bind or unbind tags in batches.

### Query and Tagging

Region: \*

Resource type: \*

Tag:  :  [Delete](#)

[Add](#)

[Query Resources](#) [Reset](#) [More](#)

---

[Edit Tag](#) Selected: 0/1

<input type="checkbox"/>	Resource ID	Resource name	Service
All tags retrieved successfully			
<input type="checkbox"/>	rocketmq		Tencent Distributed Message Queue

Total items: 1 10 / page

## Other authorization methods

[Operation-Level Authorization](#)

[Resource-Level Authorization](#)

# Tag Management

## Managing Resource with Tag

Last updated : 2024-01-18 10:04:39

### Overview

**Tag** is a key-value pair provided by Tencent Cloud to identify a resource in the cloud. It can help you easily categorize and manage TDMQ for RocketMQ resources in many dimensions such as business, purpose, and owner.

**Note:**

Tencent Cloud will not use the tags you set, and they are only used for your management of TDMQ for RocketMQ resources.

### Use Limits

You need to pay attention to the following use limits of tags:

Limit	Description
Quantity	One Tencent Cloud resource can have up to 50 tags.
Tag key	You cannot place <code>qcloud</code> , <code>tencent</code> , or <code>project</code> at the beginning of a tag key as they are reserved by the system. A tag key can contain up to 255 digits, letters, and special symbols ( <code>+ = . @ -</code> ).
Tag value	It can contain up to 127 digits, letters, and special symbols ( <code>+ = . @ -</code> ) or be an empty string.

### Directions and Use Cases

**Use case**

A company has 6 TDMQ for RocketMQ clusters, with the department, business scope, and owner information as described below:

Cluster ID	Department	Business Scope	Owner
rocketmq-qzga74ov5gw1	Ecommerce	Marketing	John
rocketmq-qzga74ov5gw2	Ecommerce	Marketing	Harry



rocketmq-qzga74ov5gw3	Gaming	Game A	Jane
rocketmq-qzga74ov5gw4	Gaming	Game B	Harry
rocketmq-qzga74ov5gw5	Entertainment	Post-production	Harry
rocketmq-qzga74ov5gw6	Entertainment	Post-production	John

You can add the following three tags to the `rocketmq-qzga74ov5gw1` cluster:

Tag Key	Tag Value
dept	ecommerce
business	mkt
owner	zhangsan

Similarly, you can also set appropriate tags for other resources based on their department, business scope, and owner information.

## Setting tag in TDMQ for RocketMQ console

After designing the tag keys and values as detailed above, you can log in to the TDMQ for RocketMQ console to set tags.

1. Log in to the [TDMQ for RocketMQ console](#).
2. On the **Cluster Management** page, select the target region and cluster and click **Edit Resource Tag** at the top of the page.

Cluster ID/Name	Topics	Group Count	Cluster Description
<input checked="" type="checkbox"/> rocketmq-8x3mzbkvmz8 test	Used: 1 Capacity: 1000	Used: 0 Capacity: 10000	
<input checked="" type="checkbox"/> rocketmq-g833r57qd4w8 test	Used: 0 Capacity: 1000	Used: 0 Capacity: 10000	test

3. Set tags in the **Edit Tag** pop-up window.

For example, add three tags for the `rocketmq-qzga74ov5gw1` cluster.

## Edit Tags ✕

The tag is used to manage resources by category from different dimensions. If the existing tag does not meet your requirements, please go to [Manage Tags](#) ↗

1 resource selected

business ▼

mkt ▼ ✕

dept ▼

ecommerce ▼ ✕

owner ▼

zhangsan ▼ ✕

[+ Add](#)

OK
Cancel

**Note:**

If existing tags cannot meet your needs, go to [Tag Management](#) to create more.

4. Click **OK**, and you will be prompted that the tags have been modified successfully. You can view the tags bound to a cluster in its **Resource Tag** column.

	Cluster ID/Name	Topics	Group Count	Cluster Description
<input checked="" type="checkbox"/>	rocketmq-8x3mzbkvmmz8 test	Used: 1 Capacity: 1000	Used: 0 Capacity: 10000	

**Filtering resource by tag key**

You can filter out clusters bound to a specific tag in the following steps:

1. Select **Tag** in the search box at the top-right corner of the page.
2. In the window that pops up, select the tag you want to search for and click **OK**.

For example, if you select `Tag: owner:zhangsan`, you can filter out clusters bound to the tag key

owner:zhangsan .

<input checked="" type="checkbox"/> Cluster ID/Name	Topics	Group Count	Cluster Description
1 result found for "Tag:owner : zhangsan" <a href="#">Back to list</a>			
<input checked="" type="checkbox"/> rocketmq-8x3mzbkvmz8 test	Used: 1 Capacity: 1000	Used: 0 Capacity: 10000	

# Editing Tag

Last updated : 2024-01-18 10:06:03

## Overview

This document describes how to edit resource tags.

## Use Limits

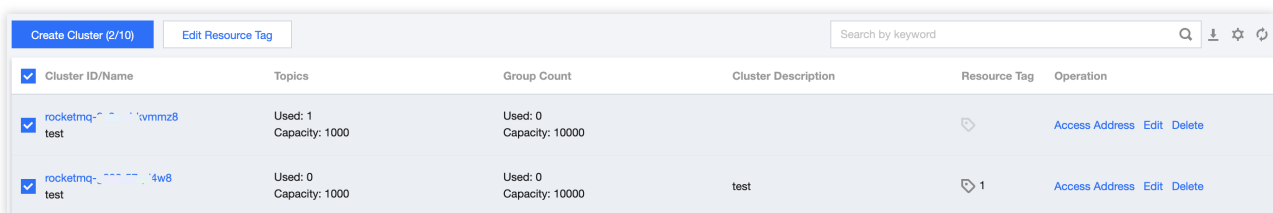
For the use limits of tags, see [Managing Resource with Tag - Use Limits](#).

## Prerequisites

You have logged in to the [TDMQ for RocketMQ console](#).

## Directions

1. On the **Cluster Management** page, select the target region and cluster and click **Edit Resource Tag** at the top of the page.



Cluster ID/Name	Topics	Group Count	Cluster Description	Resource Tag	Operation
<input checked="" type="checkbox"/> rocketmq-vmz8 test	Used: 1 Capacity: 1000	Used: 0 Capacity: 10000			<a href="#">Access Address</a> <a href="#">Edit</a> <a href="#">Delete</a>
<input checked="" type="checkbox"/> rocketmq-4w8 test	Used: 0 Capacity: 1000	Used: 0 Capacity: 10000	test	1	<a href="#">Access Address</a> <a href="#">Edit</a> <a href="#">Delete</a>

### Note:

You can batch edit tags for up to 20 resources at a time.

2. In the **Edit Tag** pop-up window, add, modify, or delete tags as needed.

## Use Cases

For directions on how to use tags, see [Managing Resource with Tag](#).

# Monitoring and Alarms

## Cluster Monitoring

Last updated : 2023-09-12 17:53:17

### Overview

TDMQ for RocketMQ supports cluster monitoring through production/consumption, storage, and consumer group metrics. You can analyze the cluster usage based on the monitoring data and handle potential risks promptly. You can also set alarm rules for monitoring metrics to receive alarm messages when metrics are abnormal. This helps you deal with risks in time and ensure the stable operations of your system.

### Monitored metrics

The monitoring metrics supported by TDMQ for RocketMQ are as follows:

Category	Unit	Metric
Production and consumption	MBytes	Cluster Production Traffic per Second
	Count/sec	Message Production Rate
	MBytes	Cluster Consumption Traffic per Second
	Count/sec	Message Consumption Speed
	Count	Heaped Messages
Billing (displayed for virtual cluster only)	Count/sec	Cluster Throttling Occurrences per Second
	Count/sec	Cluster Message Consumption API Calls per Second
	Count/sec	Cluster Message Production API Calls per Second
Storage (displayed for exclusive cluster only)	MBytes	Available Disk Space
	%	Disk Utilization

Node (displayed for exclusive cluster only)	%	Node Load (actual TPS of the current node/ TPS supported on the purchase page * 100%)
Consumer group	Count	Online Consumers
	Count/sec	Message Consumption Speed
	MBytes	Consumption Traffic per Second
	Count	Heaped Messages

## Viewing monitoring data

1. Log in to the [TDMQ for RocketMQ console](#).
2. Select **Cluster** on the left sidebar, select a region, and click the ID of the target cluster to enter the cluster details page.
3. At the top of the cluster details page, select the **Cluster Monitoring** tab to enter the monitoring page.
4. Select the target resource and set the time range to view the corresponding monitoring data.

# Group Monitoring

Last updated : 2023-10-19 10:52:06

## Overview

This document describes how to view the monitoring data of a created consumer group in the TDMQ for RocketMQ console.

## Monitoring Metrics

Monitoring Metric	Description
Message Consumption Speed (messages/sec)	The number of messages consumed by all consumers under the consumer group per second in the selected time range.
Consumption Traffic per Sec (MB)	The data size of messages consumed by all consumers under the consumer group per second in the selected time range.
Heaped Messages	The number of unconsumed messages heaped under the consumer group per second in the selected time range.
Online Consumers	The number of online consumers under the consumer group per second in the selected time range.
Dead Letters in the Consumer Group	The number of dead letter messages totaled by all consumers under the consumer group in the selected time range.

## Directions

1. Log in to the [TDMQ console](#), select a region, and click the ID of the target cluster to enter the **Basic Info** page of the cluster.
2. Select the **Group** tab to enter the consumer group list, and click the



icon in the **Monitoring** column of the target consumer group.

3. After setting the time range and time granularity, you can view the monitoring data of the corresponding consumer group.

4. On the monitoring page, you can switch the topics subscribed to by the consumer group to view the monitoring data, such as the consumption speed and traffic, of a specific topic.

The screenshot displays the Tencent Distributed Message Queue (TDMQ) monitoring interface for RocketMQ. On the left, a dark sidebar contains navigation options: Pulsar, RocketMQ (selected), and RabbitMQ. The main content area is titled 'Cluster / test' and shows a list of consumer groups under the namespace 'sdaa'. The groups listed are 'group-364733' and 'dasda', both with 'Online' status and bar chart icons. A 'Create (2/1500)' button is visible above the list. The right-hand panel provides detailed monitoring for the selected group 'group-364733'. It includes a 'Real Time' tab, a 'Period' dropdown set to '1 minute(s)', and several monitoring metrics, all of which currently show 'No data':

- Message Consumption Speed (messages/sec)
- Consumption Traffic per Second (MB)
- Heaped Messages
- Online Consumers
- Dead Letters in Consumer Group



# Topic Monitoring

Last updated : 2023-10-19 11:00:26

## Overview

This document describes how to view the monitoring data of a created topic in the TDMQ for RocketMQ console.

## Monitoring Metrics

Monitoring Metric	Description
Production Speed (messages/sec)	The number of messages produced by all producers under the topic per second in the selected time range.
Consumption Speed (messages/sec)	The number of messages consumed by all consumers under the topic per second in the selected time range.
Production Traffic (byte/sec)	The data size of messages produced by all producers under the topic per second in the selected time range.
Consumption Traffic (byte/sec)	The data size of messages consumed by all consumers under the topic per second in the selected time range.
Heaped Messages	The number of messages heaped under the topic per second in the selected time range.
Sending Throttling Occurrences (count/sec)	The number of times message production is throttled under the topic per second in the selected time range.
API Calls (count/sec)	The message production TPS of this topic, that is, the number of API calls for producing messages to this topic. It is calculated according to the billing rules. This metric is only displayed for virtual clusters.

## Directions

1. Log in to the [TDMQ console](#), select a region, and click the ID of the target cluster to enter the **Basic Info** page of the cluster.
2. Select the **Topic** tab and click the



icon in the **Monitoring** column of the target topic in the topic list.

3. Set the time range to view the monitoring data of the corresponding topic.

The screenshot displays the Tencent Cloud console interface for monitoring a RocketMQ topic. On the left, a navigation sidebar shows 'RocketMQ' expanded to 'Cluster'. The main content area is titled 'Cluster / test' and includes a 'Basic Info' tab and a 'Cluster Monitoring' tab. Under 'Cluster Monitoring', the 'Current Namespace' is set to 'sdaa'. A table lists topics, with 'dsada' selected and a monitoring icon (bar chart) visible in its 'Monitoring' column. Below the table, it indicates 'Total items: 1'. On the right, the monitoring dashboard for 'dsada' in namespace 'sdaa' is shown. It features a time range selector set to 'Real Time' (with options for 'Last 24 hours', 'Last 7 days', and 'Select Date') and a 'Period' of '1 minute(s)'. The dashboard displays several metrics with corresponding line graphs: 'Production Speed' (messages/sec) at 0, 'Consumption Speed' (messages/sec) with 'No data', 'Production Traffic' (Byte/sec) at 0, 'Consumption Traffic' (Byte/sec) with 'No data', 'Heaped Messages' with 'No data', 'Sending Throttling Occurrences' (times/sec) at 0, and 'Message Production API Calls per Second' (times/sec) at 0.

# Configuring Alarm

Last updated : 2023-05-16 10:54:38

## Overview

Tencent Cloud provides the TCOP service for all users by default; therefore, you do not need to manually activate it. TCOP will start collecting monitoring data only after a Tencent Cloud product is used.

TDMQ for RocketMQ allows you to monitor the resources created under your account, so that you can keep track of the status of your resources in real time. You can configure alarm rules for monitoring metrics. When a monitoring metric reaches the set alarm threshold, TCOP will notify you of exceptions in time via the configured notification channel.

## Directions

### Configuring an alarm policy

An alarm policy can determine whether an alarm notification should be sent based on the comparison between the monitoring metric and the given threshold in the selected time period. You can promptly take appropriate precautionary or remedial measures when the alarm is triggered by a TDMQ for RocketMQ status change. Properly configured alarm policies help improve the robustness and reliability of your applications.

#### Note

Be sure to configure alarms for your instance to prevent exceptions caused by traffic spikes or specification limits.

1. Log in to the [TCOP console](#).
2. On the left sidebar, select **Alarm Configuration > Alarm Policy** and click **Create**.
3. On the alarm configuration page, select a policy type and instance, and set the alarm rule and notification template.

**Policy Type:** Select **TDMQ/RocketMQ/Topic alarm**.

**Alarm Object:** select the TDMQ for RocketMQ resource for which to configure the alarm policy.

**Trigger Condition:** You can select **Select template** or **Configure manually**. The latter is selected by default. For more information on manual configuration, see the description below. For more information on how to create a template, see [Creating a trigger condition template](#).

#### Note

**Metric:** For example, if you select 1 minute as the statistical period for the "Heaped Messages" metric, then if the message retention volume exceeds the threshold for N consecutive data points, an alarm will be triggered.

**Alarm Frequency:** For example, "Alarm once every 30 minutes" means that there will be only one alarm triggered every 30 minutes if a metric exceeds the threshold in several consecutive statistical periods. Another alarm will be triggered only if the metric exceeds the threshold again in the next 30 minutes.

**Notification Template:** You can select an existing notification template or create one to set the alarm recipient objects and receiving channels.

4. Click **Complete**.

#### Note

For more information on alarms, see [Creating Alarm Policy](#).

### Creating a trigger condition template

1. Log in to the [TCOP console](#).
2. On the left sidebar, click **Trigger Condition Template** to enter the **Template** list page.
3. Click **Create** on the **Trigger Condition Template** page.
4. On the template creation page, configure the policy type.

**Policy Type:** Select **TDMQ-RocketMQ-topic alarm**.

**Apply preset trigger conditions:** Select this option and the system recommended alarm policy will be displayed.

5. After confirming that everything is correct, click **Save**.
6. Return to the **Create Alarm Policy** page, click **Refresh**, and the alarm policy template just configured will be displayed.

## Alarm Metric Dimension

Currently, TDMQ for RocketMQ metrics are divided into the following dimensions. You can select corresponding metrics to monitor and configure alarms based on your needs:

Metric Dimension	Metric Description
Cluster	Cluster-level data aggregation, such as the cluster's production/consumption speed and traffic, heaped message count, and traffic throttling times.
Storage (only for exclusive clusters)	The remaining available storage space of the current exclusive cluster and the proportion of storage used.
Node (only for exclusive clusters)	The load of each compute node in the current cluster
Topic	The production/consumption speed and traffic of the selected topic and its heaped message count.
Group	The number of online consumers (clients) of the selected consumer group, consumption speed and traffic, heaped message count, and dead letter count.
Consumer groups under a topic	The metrics of multiple groups are divided based on the topics subscribed to, so as to display the consumption speed/traffic and heaped message count in the groups of a specific topic.



# Message Query

## Querying Message

Last updated : 2023-03-28 10:15:36

TDMQ for RocketMQ records the complete flow of a message being sent from the producer to the TDMQ for RocketMQ server for consumption by the consumer, which is displayed as a message trace in the console. A message trace records the entire process of how a message flows, including the duration of each stage (accurate down to the microsecond), execution result, producer IP, and consumer IP.



## Overview

You can use the message query feature in the TDMQ for RocketMQ console to view the content, parameters, and trace of a specific message by time or by the message ID/message key displayed in the log. With this feature, you can do the following:

- View the specific content and parameters of the message.

- View the producer IP from which a message was sent, whether it was sent successfully, and the exact time when it arrived at the server.

- View whether the message was persistently stored.

View the consumers who consumed the message, whether it was consumed successfully, and the exact time when its consumption was acknowledged.

View the message queue's message processing latency to analyze the performance of the distributed system.

## Query Limits

You can query messages in the last 3 days.

## Prerequisites

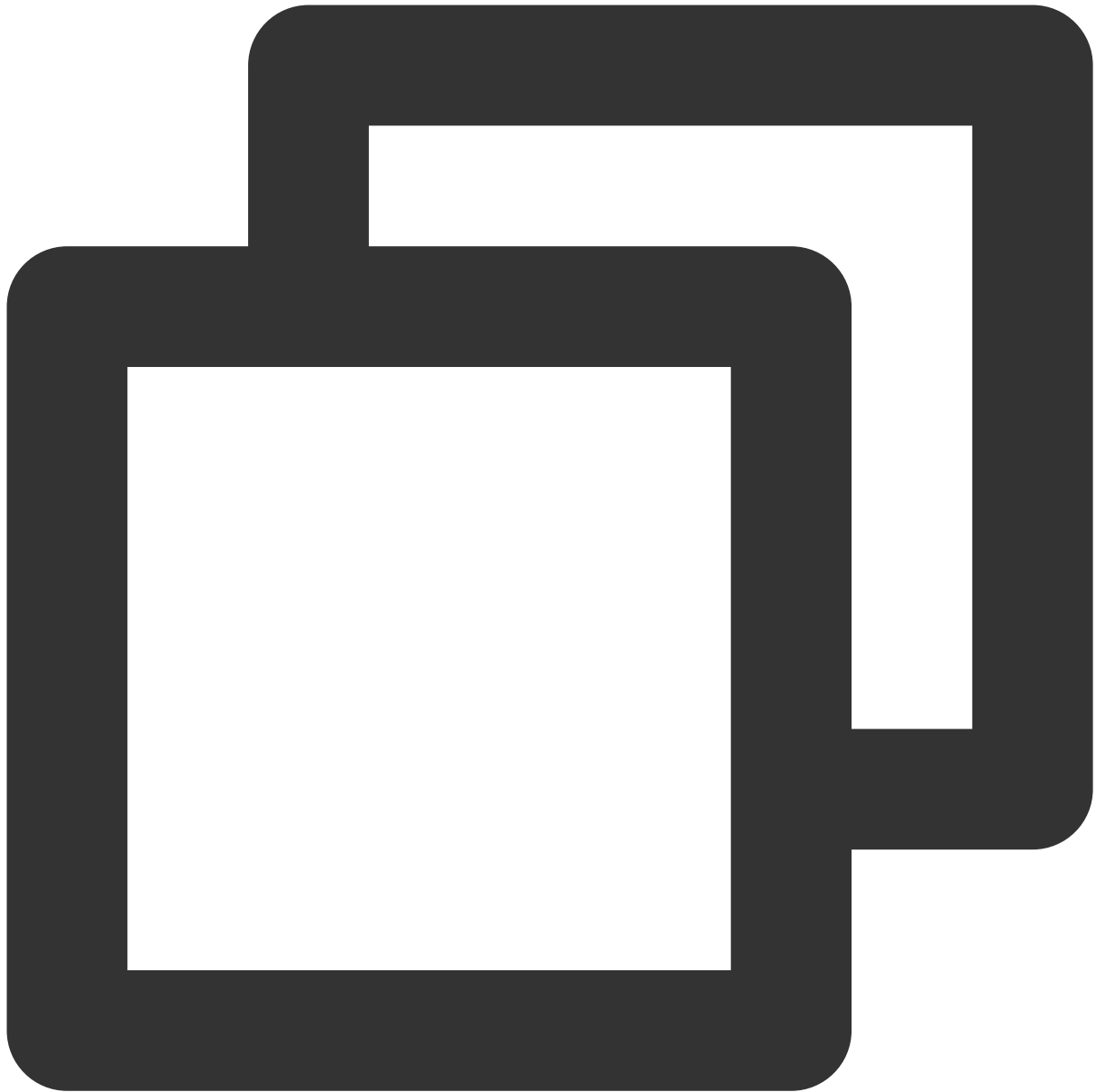
You have deployed the producer and consumer services as instructed in the [SDK Documentation](#), and they have produced and consumed messages in the last three days.

The TDMQ for RocketMQ virtual cluster service has been upgraded in some regions since August 8, 2022. The message trace feature of the old version of a virtual cluster is implemented by the server while that of the new version is implemented by the client. Therefore, if you use the new version of TDMQ for RocketMQ virtual cluster, you need to configure your client to enable the message trace feature. Below are sample settings:

Producer settings

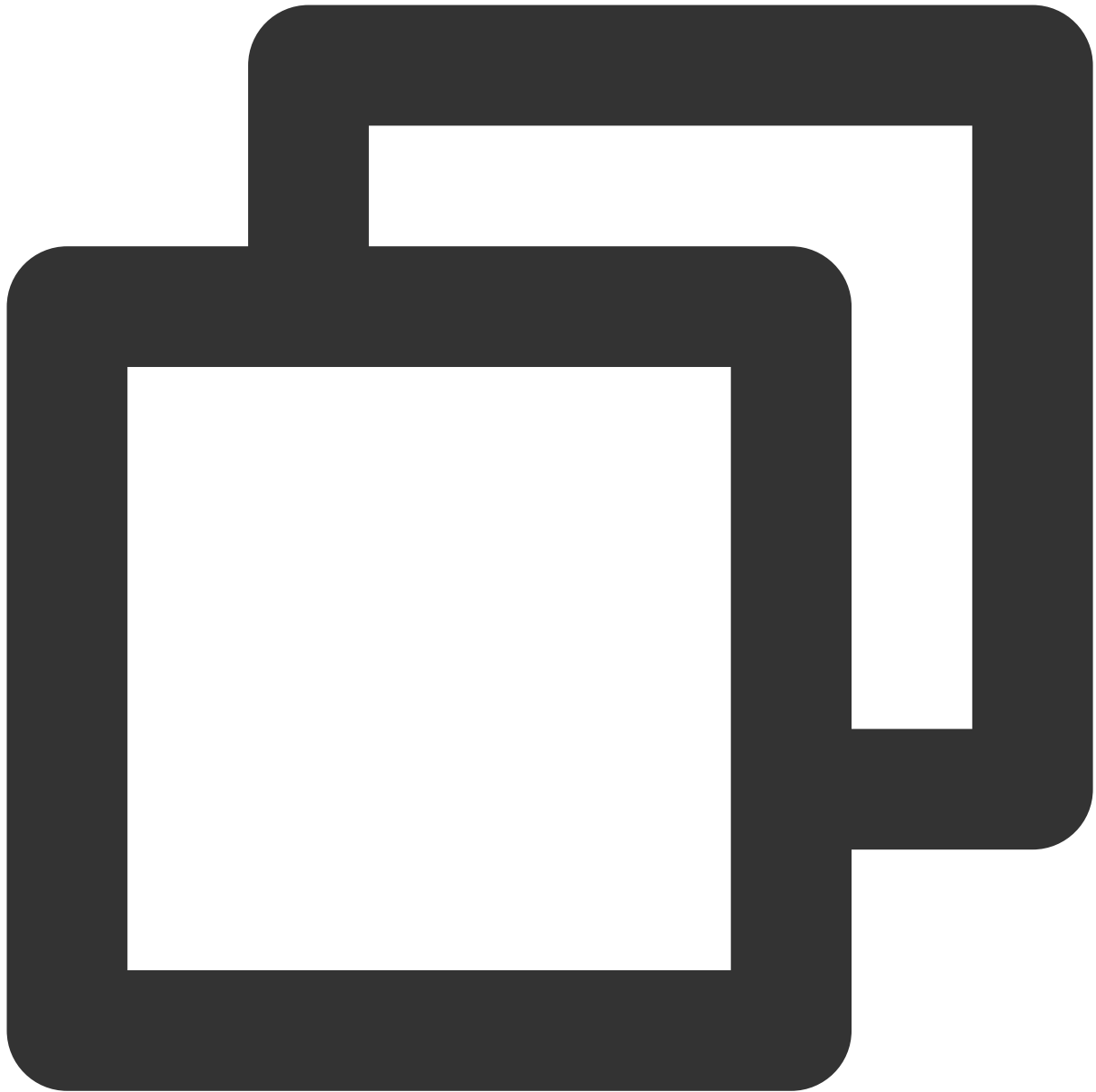
Push consumer settings

Pull consumer settings

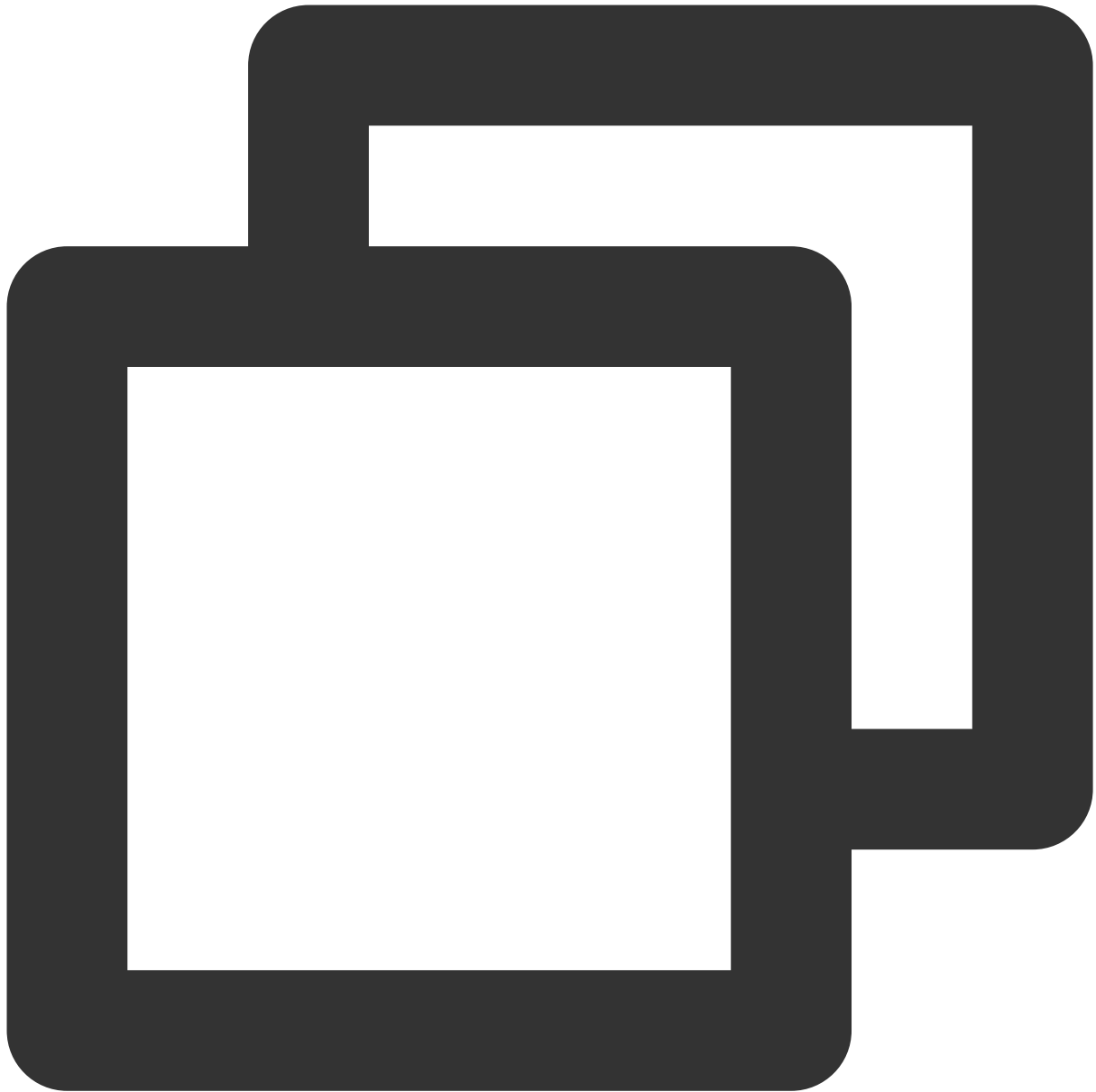


```
DefaultMQProducer producer = new DefaultMQProducer(namespace, groupName,  
    // ACL permission  
    new AclClientRPCHook(new SessionCredentials(AK, SK)), true, null);
```



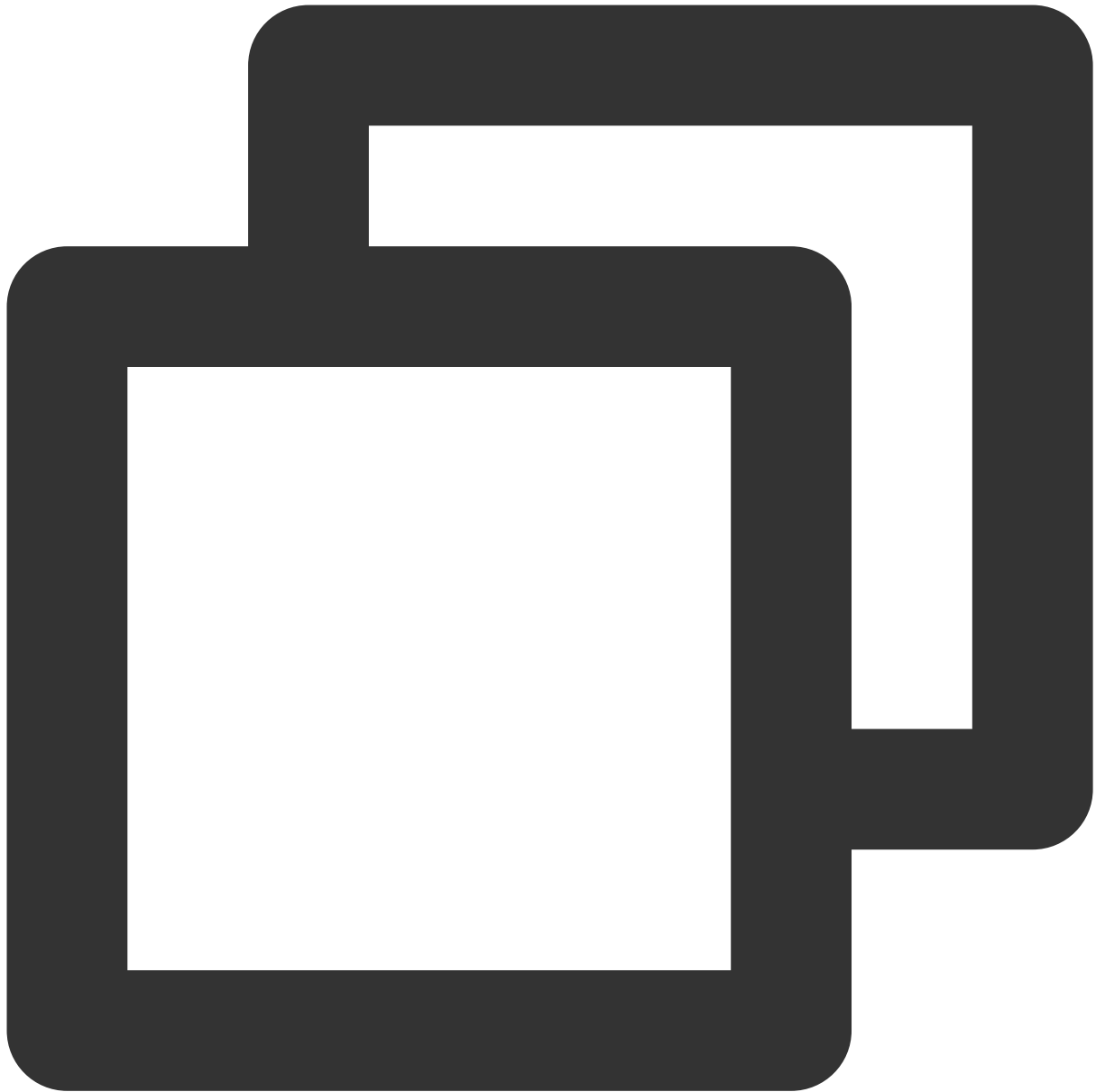


```
// Instantiate the consumer
DefaultMQPushConsumer pushConsumer = new DefaultMQPushConsumer(NAMESPACE, groupName,
    new AclClientRPCHook(new SessionCredentials(AK, SK)),
    new AllocateMessageQueueAveragely(), true, null);
```



```
DefaultLitePullConsumer pullConsumer = new DefaultLitePullConsumer(NAMESPACE, groupN
    new AclClientRPCHook(new SessionCredentials(AK, SK)));
// Set the NameServer address
pullConsumer.setNamesrvAddr(NAMESERVER);
pullConsumer.setConsumeFromWhere(ConsumeFromWhere.CONSUME_FROM_LAST_OFFSET);
pullConsumer.setAutoCommit(false);
pullConsumer.setEnableMsgTrace(true);
pullConsumer.setCustomizedTraceTopic(null);
```

If you use Spring Boot Starter 2.2.2 or later for access, see below for the specific code:



```
package com.lazycece.sbac.rocketmq.messagemodel;

import lombok.extern.slf4j.Slf4j;
import org.apache.rocketmq.spring.annotation.MessageModel;
import org.apache.rocketmq.spring.annotation.RocketMQMessageListener;
import org.apache.rocketmq.spring.core.RocketMQListener;
import org.springframework.stereotype.Component;

/**
 * @author lazycece
 * @date 2019/8/21
 */
```

```
*/
@Slf4j
@Component
public class MessageModelConsumer {

    @Component
    @RocketMQMessageListener(
        topic = "topic-message-model",
        consumerGroup = "message-model-consumer-group",
        enableMsgTrace = true,
        messageModel = MessageModel.CLUSTERING)
    public class ConsumerOne implements RocketMQListener<String> {
        @Override
        public void onMessage(String message) {
            log.info("ConsumerOne: {}", message);
        }
    }
}
}
```

## Directions

1. Log in to the [TDMQ for RocketMQ console](#) and click **Message Query** on the left sidebar.

2. On the **Message Query** page, select a region and enter the query conditions as prompted.

**Time Range:** Select the time range for query. You can select the last 30 minutes, last hour, last 6 hours, last 24 hours, or last 3 days, or set a custom time range. By default, the last 100 messages are displayed in chronological order.

**Current Cluster:** Select the cluster where the topic you want to query is located.

**Namespace:** Select the namespace where the topic you want to query is located.

**Topic:** Select the topic you want to query.

**Query Method:** Below are two supported query methods.

**By message ID:** A fast exact query method.

**By message key:** A fuzzy query method that is used when you have only set the message key.

3. Click **Query**, and the paginated results will be displayed in the list.



4. Click **View Details** in the **Operation** column of the target message to view its basic information, content (message body), and parameters.

In the **Consumption Status** module, you can view the consumer group that consumes this message and its consumption status. You can also perform the following operations in the **Operation** column:

**Send Again:** You can send the message to another client you specify. If the message has been successfully consumed, this operation may lead to repeated message consumption.

**Exception Diagnosis:** If an exception occurred during message consumption, you can view the exception diagnosis details.

**Notes:**

For the group in the broadcast consumption mode, if there is no online client under it, the information about consumption and diagnosis exceptions will not be displayed on the details page in the above figure; if you need to check the consumption status, you can go to the message trace page to view it .

5. Click **View Message Trace** in the **Operation** column or select the **Message Trace** tab on the details page to view the trace of the message. For more information, see [Message Trace Description](#).

## Consumption verification

After a message is found, you can click **Verify Consumption** in the **Operation** column to send the message to the specified client for verification. **This feature may lead to message repetition.**

**Note**

Currently, the consumption verification feature is only available to exclusive clusters. It only verifies the client consumption logic to make sure it is normal without affecting message receiving. Therefore, the verification causes no changes in any message information such as message consumption status.

## Message export

After querying a certain message, you can click **Export** in the operation column to view the message body, tag, key, production time, and consumption attributes.

You can also batch select and export messages on the current page. After the messages are exported locally, you can process them as needed, such as copying the message body or sorting them by time.

# Querying Dead Letter Message

Last updated : 2023-03-14 15:27:54

## Overview

A dead letter queue is a special type of message queue used to centrally process messages that cannot be consumed normally. If a message cannot be consumed after a specified number of retries, TDMQ for RocketMQ will determine that the message cannot be consumed under the current situation and deliver it to the dead letter queue.

In actual scenarios, messages may not be consumed due to service downtime or network disconnection. In this case, they will not be discarded immediately; instead, they will be persisted by the dead letter queue. After fixing the problem, you can create a consumer subscription to the dead letter queue to process such messages.

## Query Limit

You can only query messages in the last three days.

## Notes

After a message is delivered to the dead letter queue, it will not be consumed normally. You can query messages in the last 3 days. Therefore, we recommend you process dead letter messages within 3 days after generation; otherwise, they will be deleted.

All dead letter messages generated by all topics in a group are put into a dead letter queue and can be queried from there. The dead letter queue won't exist if there are no dead letter messages.

## Directions

1. Log in to the [TDMQ for RocketMQ console](#) and click **Message Query** on the left sidebar.
2. On the **Message Query** page, select a region and enter the query conditions as prompted.

**Time Range:** Select the time range for query, which can be the last 30 minutes, last hour, last 6 hours, last 24 hours, last 3 days, or a custom time range.

**Current Cluster:** Select the cluster where the dead letter message you want to query is located.

**Namespace:** Select the namespace where the dead letter message you want to query is located.

**Group:** Select the group where the dead letter message you want to query is located.

**Message ID:** It is optional.

If you don't enter the message ID, a **fuzzy query** will be performed, which will batch query all dead letter messages by group ID in the selected time range.

If you enter the message ID, an **exact query** will be performed, which will locate the message by group ID and message ID.

3. Click **Query**, and the paginated results will be displayed in the list.

4. You can select multiple dead letter messages and click **Batch Resend Messages** in the top-left corner to batch resend them to the retry queue of the original queue. You can also click **Resend Message** in the **Operation** column of a specific dead letter message to resend it. After the dead letter message is resent, it will be delivered to the retry queue of the original queue; however, it will not be deleted immediately from the dead letter queue; instead, it will be deleted at the end of the message lifecycle (3 days).

5. Click **View Details** in the **Operation** column of the target message to view its basic information, content (message body), and parameters.

6. Click **View Message Trace** in the **Operation** column or select the **Message Trace** tab on the details page to view the trace of the message. For more information, see [Message Trace Description](#).

You can see that after the dead letter message is redelivered, the consumption status changes to **Redelivered to retry queue**.



# Message Trace Description

Last updated : 2023-03-14 15:31:36

A message trace records the entire process of how a message flows, including the duration of each stage (accurate down to the microsecond), execution result, producer IP, and consumer IP.

## Prerequisites

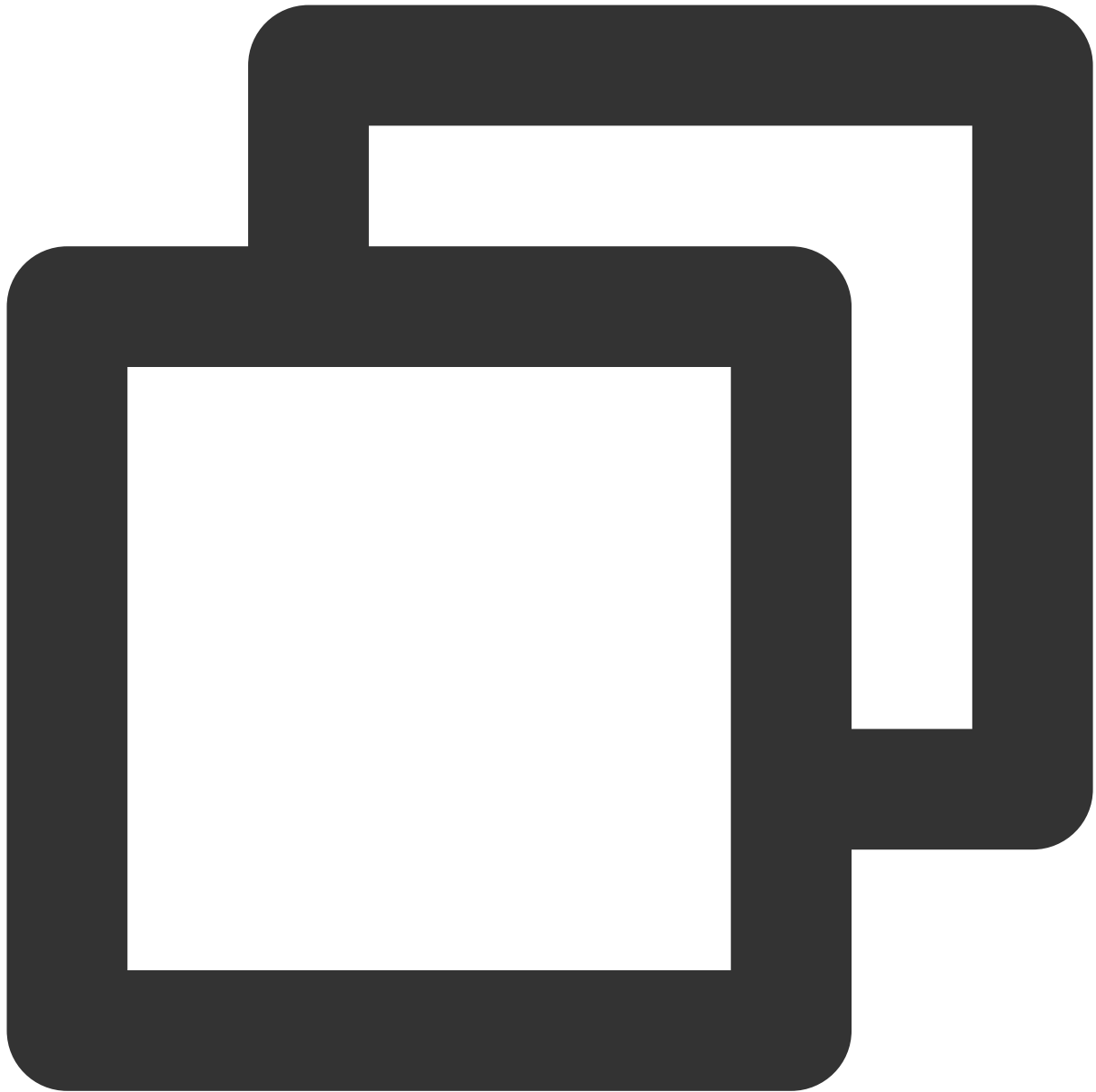
You have deployed the producer and consumer services as instructed in the [SDK Documentation](#), and they have produced and consumed messages in the last three days.

You need to enable and set the message trace feature on the client as follows:

Producer settings

Push consumer settings

Pull consumer settings



```
DefaultMQProducer producer = new DefaultMQProducer(namespace, groupName,  
    // ACL permission  
    new AclClientRPCHook(new SessionCredentials(AK, SK)), true, null);
```

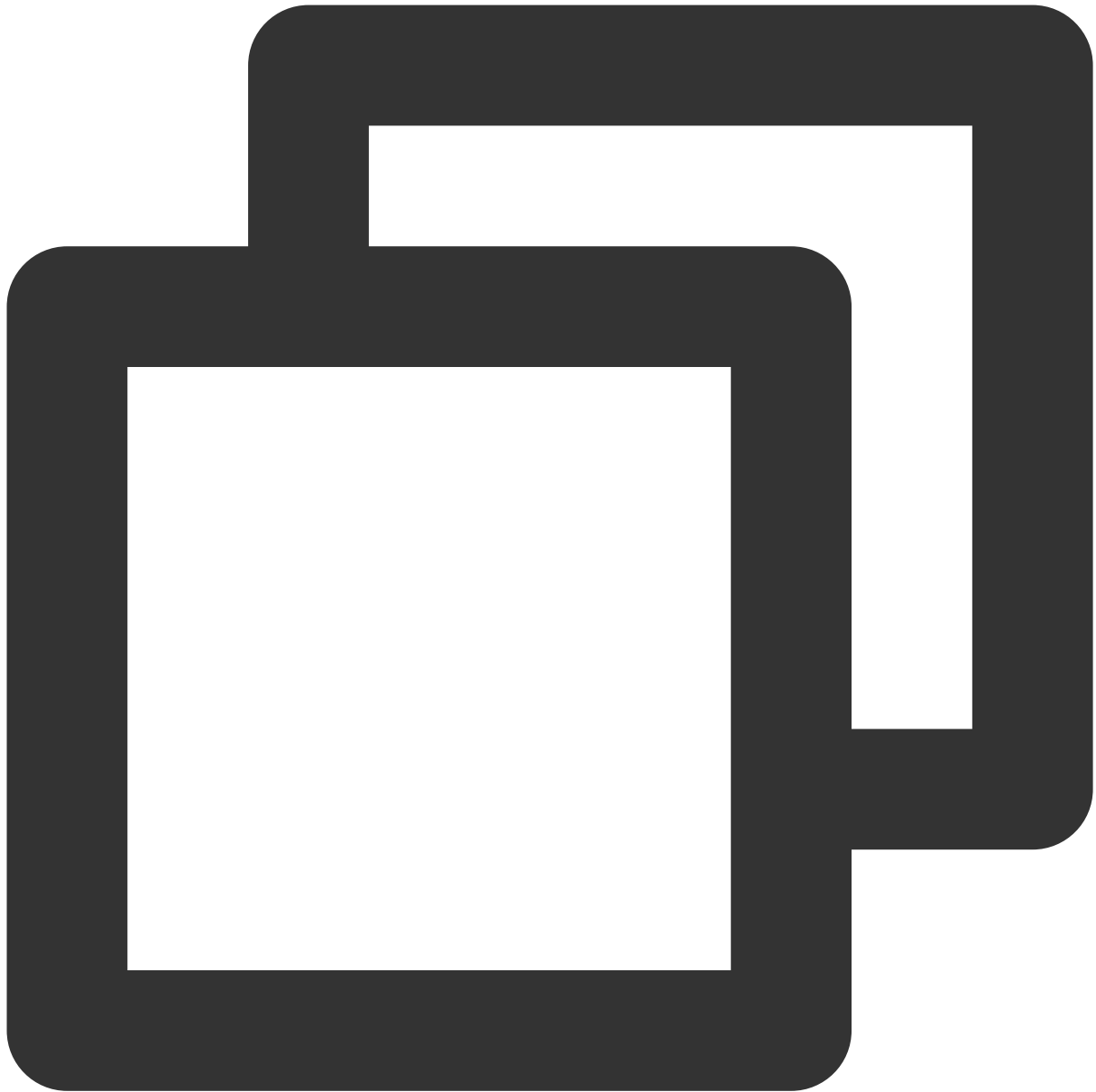


```
// Instantiate the consumer
DefaultMQPushConsumer pushConsumer = new DefaultMQPushConsumer(NAMESPACE, groupName,
    new AclClientRPCHook(new SessionCredentials(AK, SK)),
    new AllocateMessageQueueAveragely(), true, null);
```



```
DefaultLitePullConsumer pullConsumer = new DefaultLitePullConsumer(NAMESPACE, groupN
    new AclClientRPCHook(new SessionCredentials(AK, SK)));
// Set the NameServer address
pullConsumer.setNamesrvAddr(NAMESERVER);
pullConsumer.setConsumeFromWhere(ConsumeFromWhere.CONSUME_FROM_LAST_OFFSET);
pullConsumer.setAutoCommit(false);
pullConsumer.setEnableMsgTrace(true);
pullConsumer.setCustomizedTraceTopic(null);
```

If you use Spring Boot Starter 2.2.2 or later for access, see below for the specific code:



```
package com.lazycece.sbac.rocketmq.messagemodel;

import lombok.extern.slf4j.Slf4j;
import org.apache.rocketmq.spring.annotation.MessageModel;
import org.apache.rocketmq.spring.annotation.RocketMQMessageListener;
import org.apache.rocketmq.spring.core.RocketMQListener;
import org.springframework.stereotype.Component;

/**
 * @author lazycece
 * @date 2019/8/21
 */
```

```
*/
@Slf4j
@Component
public class MessageModelConsumer {

    @Component
    @RocketMQMessageListener(
        topic = "topic-message-model",
        consumerGroup = "message-model-consumer-group",
        enableMsgTrace = true,
        messageModel = MessageModel.CLUSTERING)
    public class ConsumerOne implements RocketMQListener<String> {
        @Override
        public void onMessage(String message) {
            log.info("ConsumerOne: {}", message);
        }
    }
}
```

## Directions

1. Log in to the [TDMQ for RocketMQ console](#) and click **Message Query** on the left sidebar.

2. On the **Message Query** page, select a region and enter the query conditions as prompted.

**Time Range:** Select the time range for query. You can select the last 6 hours, last 24 hours, or last 3 days, or set a custom time range.

**Current Cluster:** Select the cluster where the topic you want to query is located.

**Namespace:** Select the namespace where the topic you want to query is located.

**Topic:** Select the topic you want to query.

**Query Method:** Below are two supported query methods.

**By message ID:** A fast exact query method.

**By message key:** A fuzzy query method that is used when you have only set the message key.

3. Click **Query**, and the paginated results will be displayed in the list.

4. Click **View Message Trace** in the **Operation** column or select the **Message Trace** tab on the details page to view the trace of the message.

# Message trace query result description

A message trace query result consists of three parts: message production, message storage, and message consumption.

## Message production

Parameter	Description
Producer Address	Address and port of the producer.
Production Time	The time when the TDMQ for RocketMQ server acknowledged message receipt, accurate down to the millisecond.
Sending Duration	The time it took to send the message from the producer to the TDMQ for RocketMQ server, accurate down to the microsecond.
Production Status	Message production success or failure. If the status is <b>Failed</b> , it is generally because the header of the message was lost during sending, and the above fields may be empty.

## Message storage

Parameter	Description
Storage Time	The time when the message was persistently stored.
Storage Duration	The duration between when the message was persistently stored and when the TDMQ for RocketMQ server received the acknowledgment, accurate down to the millisecond.
Storage Status	Message storage success or failure. If the status is <b>Failed</b> , the message failed to be stored on the disk, which is possibly because the underlying disk was damaged or full. In this case, submit a ticket for assistance as soon as possible.

## Message consumption

Message consumption details are displayed in a list. TDMQ for RocketMQ supports two consumption modes: cluster consumption and broadcast consumption.

The information displayed in the list is as described below:

Parameter	Description
Consumer Group Name	Name of the consumer group.
Consumption Mode	The consumer group's consumption mode, which can be either cluster consumption or

	broadcast consumption. For more information, see <a href="#">Cluster Consumption and Broadcast Consumption</a> .
Number of Pushes	The number of times the TDMQ for RocketMQ server has delivered the message to consumers.
Last Pushed	The last time the TDMQ for RocketMQ server delivered the message to consumers.
Consumption Status	<p>Pushed yet unacknowledged: The TDMQ for RocketMQ server has delivered the message to consumers but has not received their acknowledgment.</p> <p>Acknowledged: Consumers acknowledged the consumption and the TDMQ for RocketMQ server has received the acknowledgment.</p> <p>Put to retry queue: Acknowledgment timed out. The server will deliver the message to consumers again as it did not receive their acknowledgment.</p> <p>Retried yet unacknowledged: The TDMQ for RocketMQ server has delivered the message to consumers again but still has not received their acknowledgment.</p> <p>Put to dead letter queue: The message has been put to the dead letter queue as it failed to be consumed after multiple retries.</p> <p>Note: If the consumption mode is broadcast, the consumption status can only be <b>Pushed</b>.</p>

You can view the message push details by clicking the right triangle on the left of the subscription name.

Parameter	Description
Push Sequence	The sequence number in which the TDMQ for RocketMQ server delivers the message to consumers.
Consumer Address	Address and port of the consumer receiving the message.
Push Time	The time when the TDMQ for RocketMQ server delivers the message to consumers.
Consumption Status	<p>Pushed yet unacknowledged: The TDMQ for RocketMQ server has delivered the message to consumers but has not received their acknowledgment.</p> <p>Acknowledged: Consumers acknowledged the consumption and the TDMQ for RocketMQ server has received the acknowledgment.</p> <p>Put to retry queue: Acknowledgment timed out. The server will deliver the message to consumers again as it did not receive their acknowledgment.</p> <p>Retried yet unacknowledged: The TDMQ for RocketMQ server has delivered the message to consumers again but still has not received their acknowledgment.</p> <p>Put to dead letter queue: The message has been put to the dead letter queue as it failed to be consumed after multiple retries.</p> <p>Redelivered to retry queue: On the dead letter queue resending page, the dead letter message has been redelivered to the retry queue of the original queue.</p>





# Migration to Cloud Metadata Migration

Last updated : 2023-05-16 11:01:37

## Overview

When using a TDMQ for RocketMQ cluster, you may need to migrate your existing business, for example, from a self-built or third-party RocketMQ cluster to the TDMQ for RocketMQ cluster.

This document describes how to migrate metadata from self-built open-source RocketMQ to TDMQ for RocketMQ.

### Note

Both dedicated clusters and virtual clusters support metadata migration, and the migration steps are the same.

## Prerequisites

You have created a [cluster](#) advance and a [namespace](#) in the console.

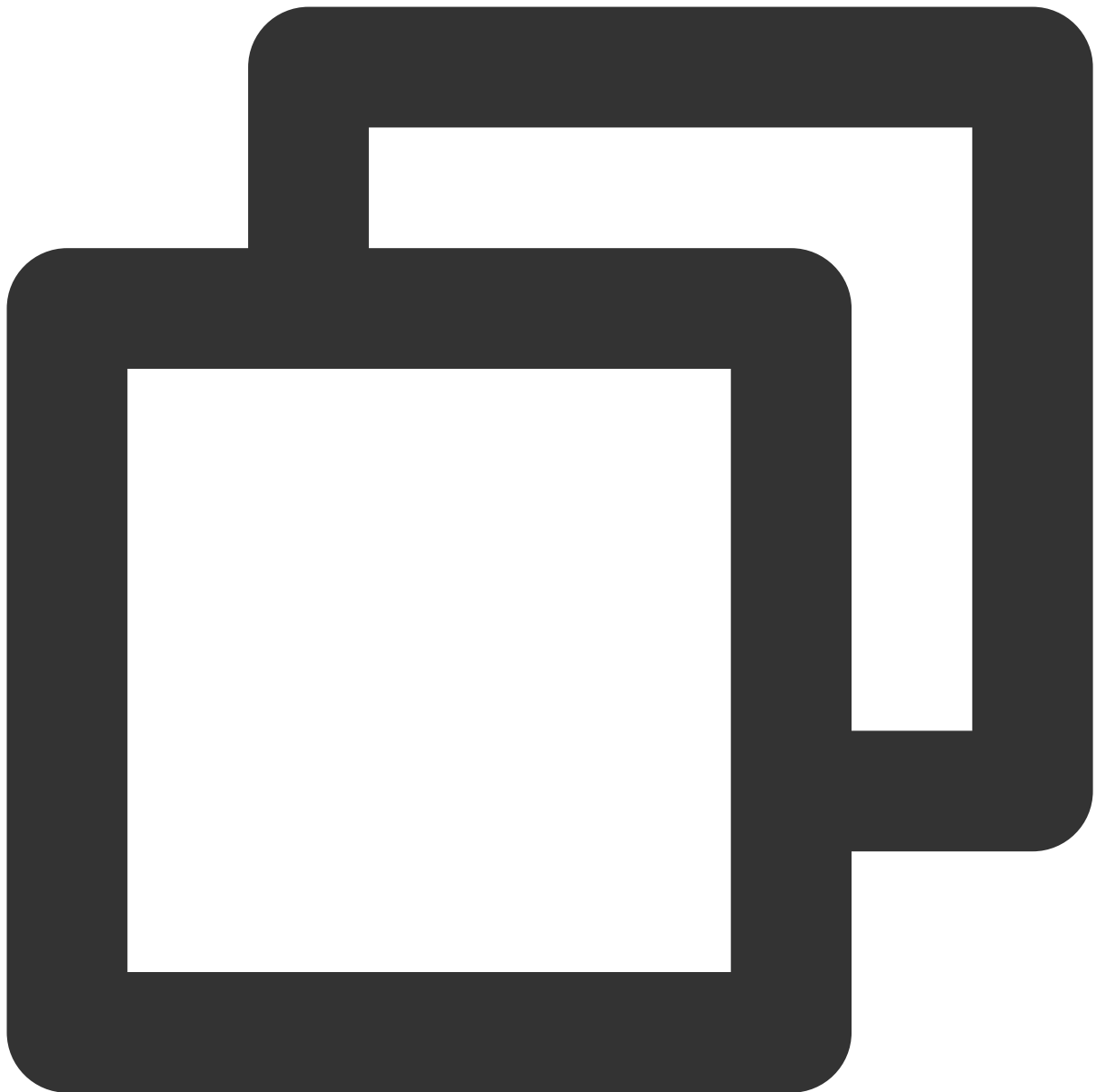
## Directions

### Step 1. Export the metadata file

If you are using self-built open-source RocketMQ, you can export metadata in the following two ways:

#### Option 1

If your RocketMQ server can access the public network, run the following script on your server directly (if there are multiple servers in your RocketMQ cluster, you can run the script on any server as long as the network is interconnected in the cluster).

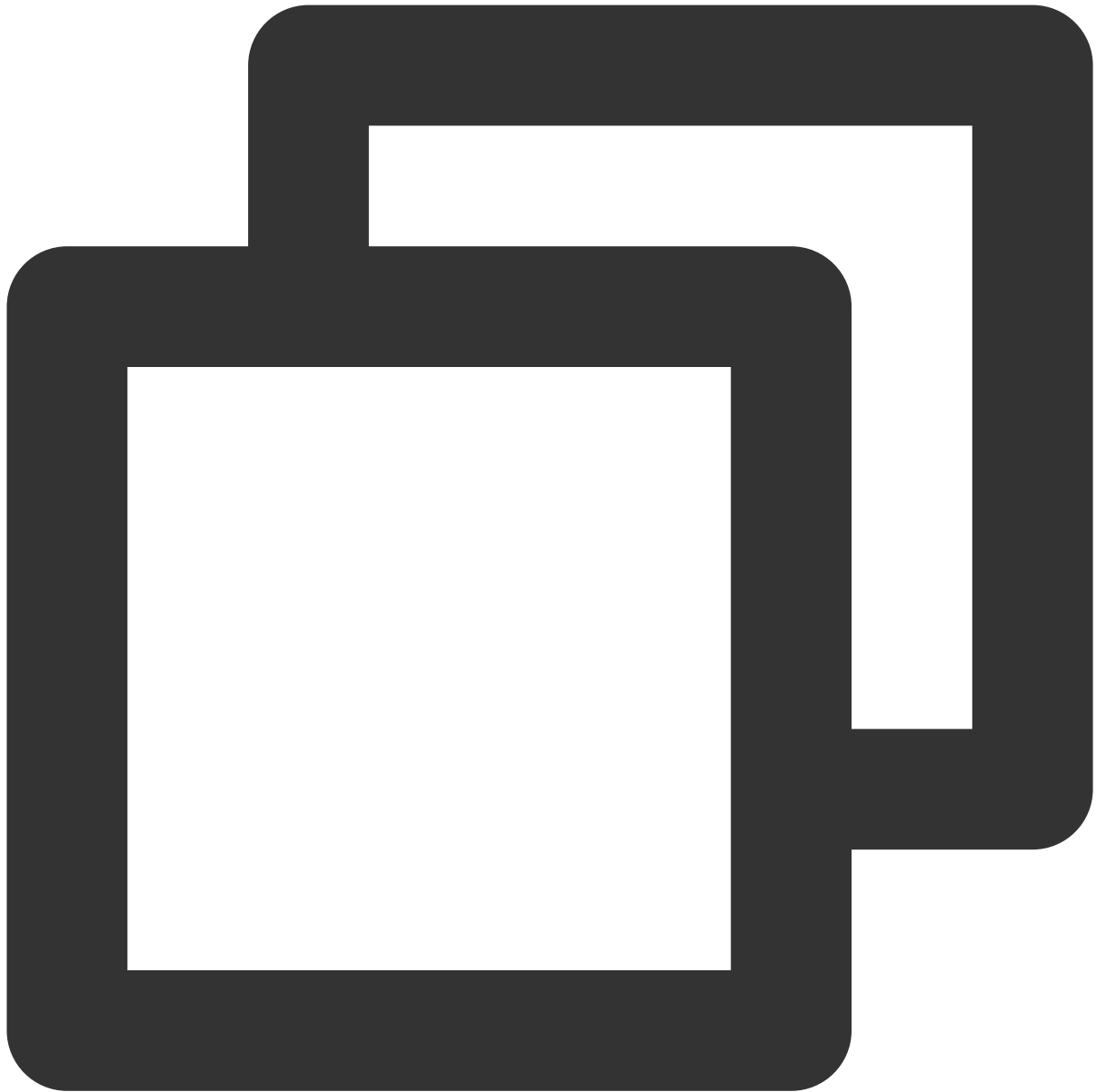


```
/bin/bash -c "$(curl -fsSL https://rocketmq-1306598660.cos.ap-guangzhou.myqcloud.co
```

## Option 2

If your RocketMQ server has no permission to access the public network, follow the steps below:

1. Download the [migration tool](#).
2. Upload the tool to your self-built RocketMQ cluster (if there are multiple servers in your RocketMQ cluster, you can run the tool on any server as long as the network is interconnected in the cluster).
3. Decompress the tool and enter the directory.



```
unzip rocketmq-migration.zip  
cd rocketmq-migration
```

4. Run the following command for migration.



```
./bin/export.sh
// Enter the open-source RocketMQ address, such as `localhost:9876`
Enter name server address list:localhost:9876
// Select a cluster to export, such as `DefaultCluster`
Choose a cluster to export:DefaultCluster
// Enter a directory for saving the exported metadata, which is `/tmp/rocketmq/conf`
Enter file path to export [default /tmp/rocketmq/export]:
```

## Step 2. Create a migration task

1. Log in to the [TDMQ for RocketMQ console](#) and enter the **Migration to Cloud** page, enter the migration task list page, and click **Create Task** to create a corresponding migration task.

2. Select the migration task type:

**Cluster migration:** This type of task migrates metadata from the self-built RocketMQ cluster to the TDMQ for RocketMQ cluster. The migration tool will parse the part before “%” of each topic name in the open-source dashboard as the namespace name by default, so that you can create multiple logically isolated namespaces. If no topic name can be parsed in the self-built cluster, a namespace named `default` will be generated automatically.

**Specified namespace import:** This type of task migrates metadata from the self-built RocketMQ cluster to a specified TDMQ for RocketMQ namespace. If there are no namespaces in topics in the self-built cluster, you can select specific topics and groups you want to import, and specify the TDMQ for RocketMQ namespaces to which they are imported to distinguish between businesses or environments.

3. Upload the metadata file obtained in step 1 and select the topics and groups you want to import.

#### Note

Up to 1,000 topics and 1,000 groups can be imported in a single task. Excess data will fail to be imported.

### Step 3. Check the task status

After the task is successfully created, enter the task list to view the task status. If there is too much data, the task needs to load for a while. Click **View Details** to view the specific running status of the task.

If the task status is **Some failed** or **All failed**, you can filter causes of the failures in the **Task Status** column.

# Metadata Migration from Huawei Cloud

Last updated : 2023-05-16 10:56:42

## Overview

When using a TDMQ for RocketMQ cluster, you may need to migrate your existing business, for example, from a self-built or third-party RocketMQ cluster to the TDMQ for RocketMQ cluster. This document describes how to migrate metadata from Huawei Cloud RocketMQ to TDMQ for RocketMQ.

### Note

Both dedicated clusters and virtual clusters support metadata migration, and the migration steps are the same.

## Prerequisites

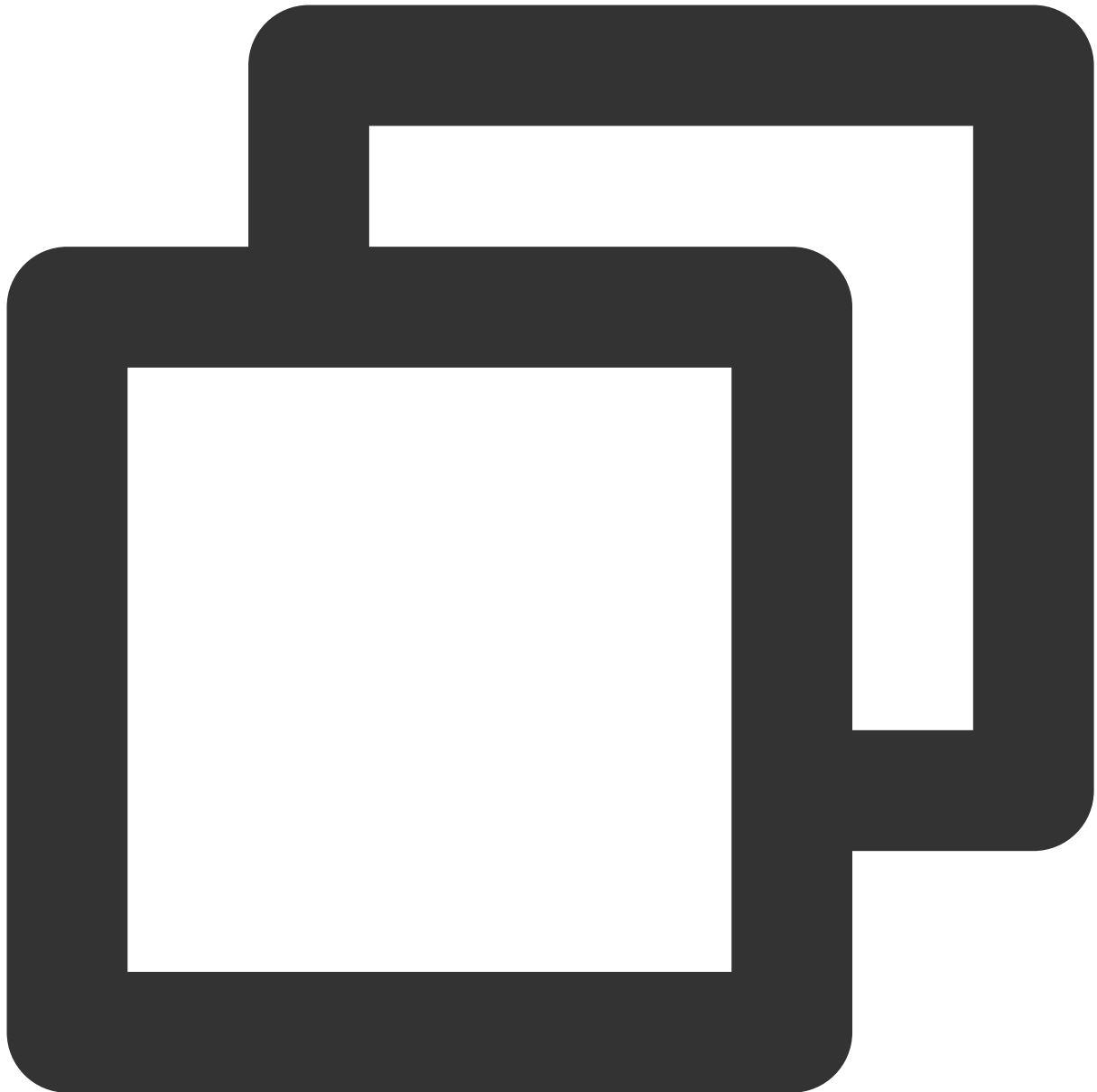
You have created a [cluster]

advance(<https://www.tencentcloud.com/document/product/1113/43122!85fe5099f6807dada65d274810933389>) and a [namespace](#) in the console.

## Directions

### Step 1. Export the metadata file

1. To export metadata on the Huawei Cloud console, you need to obtain the relevant credentials of your Huawei Cloud account. Enter the **My Credentials** page in the console, select **API Credentials**, select the region where the RocketMQ cluster resides, and copy the corresponding project ID (such as `07e5d7ab260026xxxxxa3c7347b`) and the region it belongs to (such as `cn-north-1`).
2. Go to the RocketMQ page and get the `instanceId` of the instance in Huawei Cloud.
3. Execute the following migration command on any Linux environment machine with public network connection enabled. Replace the API credential and instance ID in the command line with those obtained in the above steps.



```
./rocketmq-export hw -a YHTxxxxxxxxCGNCZFGT -s bIGoRaeJRvxxxxx7pB38SD -r cn-east-3 -
```

If your machine cannot be connected to the public network, you can click to download [rocketmq-export](#). The relevant script commands are as follows:





```
Export rocketmq metadata
```

```
Usage:
```

```
rocketmq-export hw [flags]
```

```
Flags:
```

```
-a, --accessKey string      (required) Access Key  
-h, --help                  help for hw  
-i, --instanceId string    (required) Instance ID (default "a21db77c-9e8f-4f24-a3a  
-o, --outputFile string    Output File name (default "/tmp/export-data.json")  
-p, --projectId string     (required) Project ID (default "08535f7e4680f25f2fc9c01
```

```
-r, --region string      (required) Region (default "cn-east-3")
-s, --secretKey string  (required) Secret Access Key
```

4. Export and save the metadata file as prompted. You can also set the environment variable

```
CONF_LOG_LEVEL=debug
```

 before the command to enable the debug mode for troubleshooting.

## Step 2. Create a migration task

1. Log in to the [TDMQ for RocketMQ console](#) and enter the **Migration to Cloud** page, enter the migration task list page, and click **Create Task** to create a corresponding migration task.

2. Select the migration task type:

**Cluster migration:** This type of task migrates metadata from the self-built RocketMQ cluster to the TDMQ for RocketMQ cluster. The migration tool will parse the part before “%” of each topic name in the open-source dashboard as the namespace name by default, so that you can create multiple logically isolated namespaces. If no topic name can be parsed in the self-built cluster, a namespace named `default` will be generated automatically.

**Specified namespace import:** This type of task migrates metadata from the self-built RocketMQ cluster to a specified TDMQ for RocketMQ namespace. If there are no namespaces in topics in the self-built cluster, you can select specific topics and groups you want to import, and specify the TDMQ for RocketMQ namespaces to which they are imported to distinguish between businesses or environments.

3. Upload the metadata file obtained in step 1 and select the topics and groups you want to import.

### Note

Up to 1,000 topics and 1,000 groups can be imported in a single task. Excess data will fail to be imported.

## Step 3. Check the task status

After the task is successfully created, enter the task list to view the task status. If there is too much data, the task needs to load for a while. Click **View Details** to view the specific running status of the task.

If the task status is **Some failed** or **All failed**, you can filter causes of the failures in the **Task Status** column.

# Metadata Migration from ONS 4.0

Last updated : 2023-04-12 11:30:31

## Overview

When using a TDMQ for RocketMQ cluster, you may need to migrate your existing business, for example, from a self-built or third-party RocketMQ cluster to the TDMQ for RocketMQ cluster.

This document describes how to migrate metadata from AlibabaMQ for Apache RocketMQ (formerly ONS) to TDMQ for RocketMQ.

### Note

Both dedicated clusters and virtual clusters support metadata migration, and the migration steps are the same.

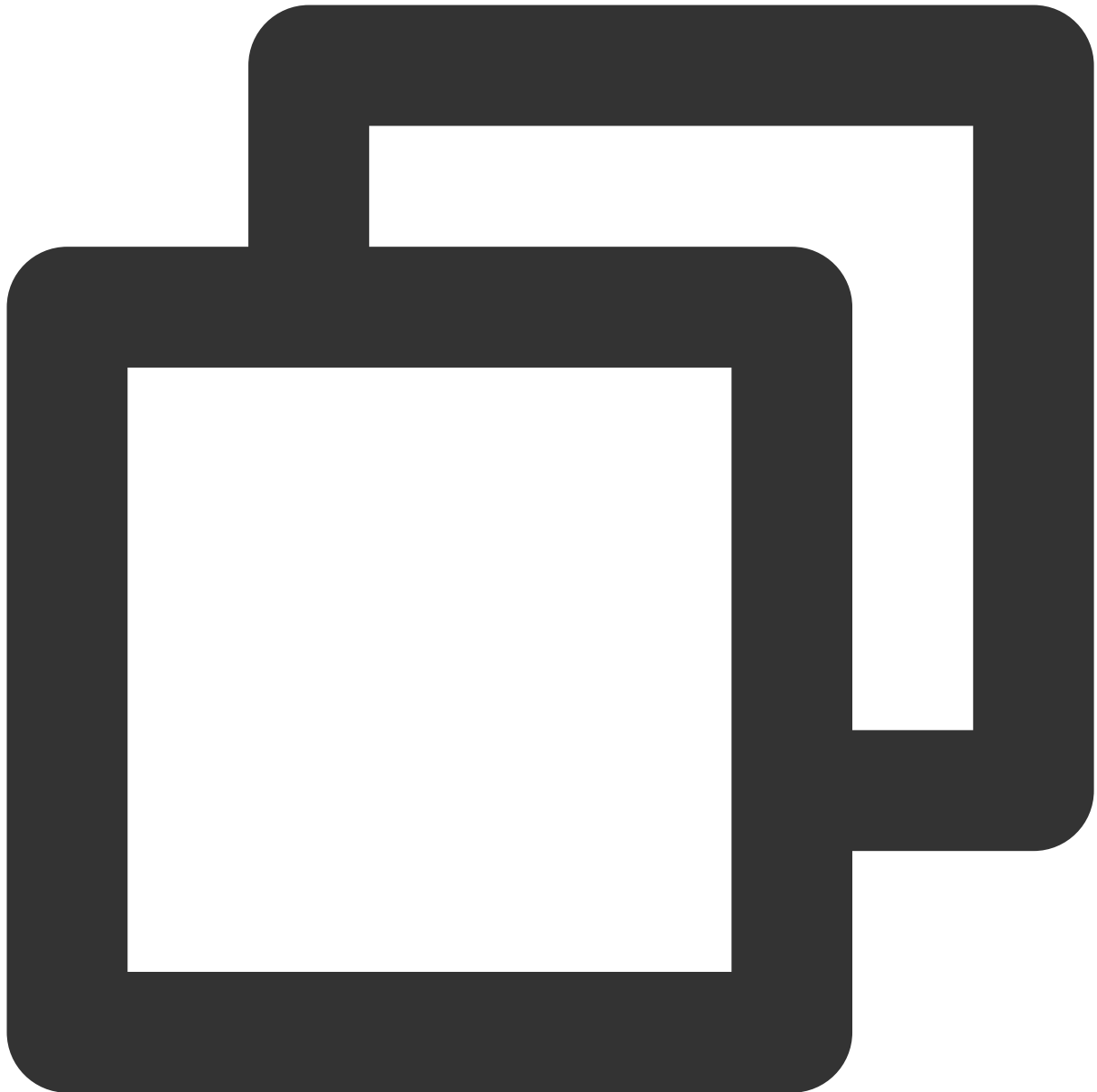
## Prerequisites

You have created a [cluster](#) advance and a [namespace](#) in the console.

## Directions

### Step 1. Export the metadata file

1. To export the metadata in the AlibabaMQ for Apache RocketMQ console, you need to obtain the AccessKey ID and AccessKey Secret of the Alibaba Cloud root account. For details, see [Documentation](#).
2. Run the following migration command on any server in Linux environment that is connected to the public network.



```
CONF_ACCESSKEY_ID=LTAD***k59ppyN CONF_ACCESSKEY_SECRET=Xx8e86***L2lgI38Z /bin/bas
```

### Notes

Set the values of `CONF_ACCESSKEY_ID` and `CONF_ACCESSKEY_SECRET` to the `AccessKey ID` and `AccessKey Secret` you obtained, respectively.

3. Export and save the metadata file as prompted. You can also set the environment variable `CONF_LOG_LEVEL=debug` before the command to enable the debug mode for troubleshooting.

## Step 2. Create a migration task

1. Log in to the [TDMQ for RocketMQ console](#), go to the **Migration to Cloud** page, enter the migration task list page, and click **Create Task** to create a corresponding migration task.

2. Select the migration task type:

**Cluster migration:** This type of task migrates metadata from the self-built RocketMQ cluster to the TDMQ for RocketMQ cluster. The migration tool will parse the part before “%” of each topic name in the open-source dashboard as the namespace name by default, so that you can create multiple logically isolated namespaces. If no topic name can be parsed in the self-built cluster, a namespace named `default` will be generated automatically.

**Specified namespace import:** This type of task migrates metadata from the self-built RocketMQ cluster to a specified TDMQ for RocketMQ namespace. If there are no namespaces in topics in the self-built cluster, you can select specific topics and groups you want to import, and specify the TDMQ for RocketMQ namespaces to which they are imported to distinguish between businesses or environments.

3. Upload the metadata file obtained in step 1 and select the topics and groups you want to import.

#### Notes

Up to 1,000 topics and 1,000 groups can be imported in a single task. Excess data will fail to be imported.

### Step 3. Check the task status

After the task is successfully created, you can go to the task list to check its status. If there is too much data, it may take a period of time to load the task. Click **View Details**, and you can check the task status.

If the task status is “Some failed” or “All failed”, you can filter causes of the failures in the **Task Status** column.

# Message Service Data Flow Migration

Last updated : 2024-01-18 10:07:12

## Overview

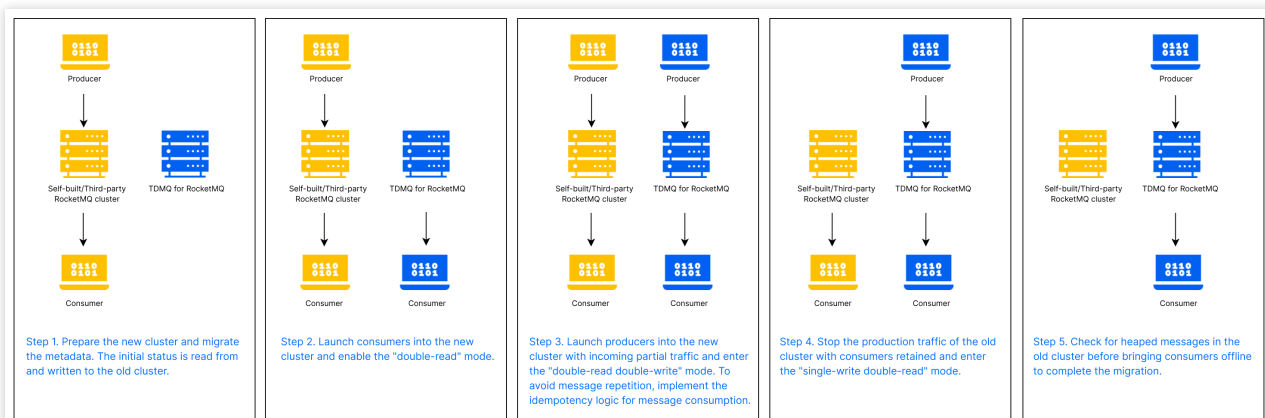
In a [metadata migration task](#), you can sync the metadata of a self-built RocketMQ cluster to TDMQ for RocketMQ. After the sync, you need to modify the access information of the producer and consumer clusters in order to migrate them from the self-built cluster to TDMQ for RocketMQ for message sending and receiving.

### Note:

Currently, the message migration service migrates only message production and consumption linkages but not message data in the old RocketMQ cluster. It applies only to exclusive clusters as the destination and will be supported for virtual clusters after the beta test ends.

## Migration Directions

This document describes the **double-read double-write** and **batch release** schemes of the message migration service. During migration, the producer and consumer clusters can produce or consume messages in the old RocketMQ and new TDMQ for RocketMQ clusters in parallel. Data will not be heaped because of the migration, so the business can transition smoothly.



The detailed directions are as follows:

1. Create a TDMQ for RocketMQ cluster, migrate the metadata, and get the required client information in the console, such as the access point of the new cluster, `AccessKey`, and `SecretKey`.
2. Modify the access information of certain nodes in the consumer cluster to connect corresponding consumers to the new TDMQ for RocketMQ cluster. They will consume messages in the new cluster, while the rest will continue to

consume messages in the old cluster.

3. Modify the access information of certain nodes in the producer cluster to connect corresponding producers to the new TDMQ for RocketMQ cluster. They will send messages to the new cluster, while the rest will continue to send messages to the old cluster. To avoid message repetition or loss, implement the idempotency logic for message consumption in advance.

4. Connect the remaining producers to the new TDMQ for RocketMQ cluster. Then, all messages will be sent to the new cluster.

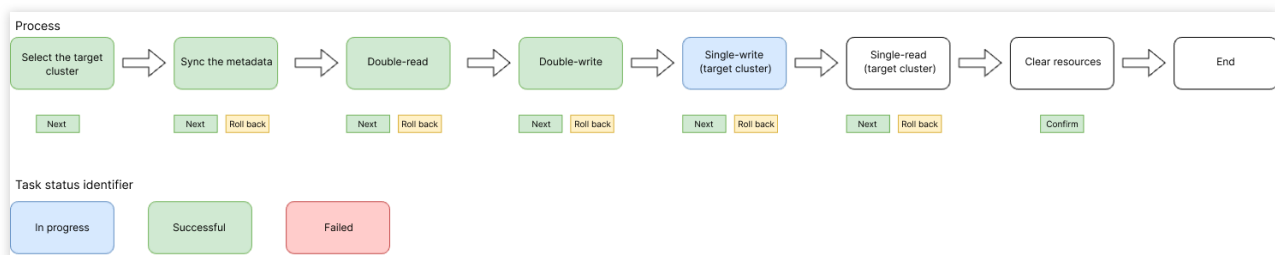
5. Check whether there are heaped messages that are not consumed in the old RocketMQ cluster, and if not, connect the remaining consumers to the new TDMQ for RocketMQ cluster. At this point, the migration is completed.

### Note:

If you don't follow the above steps strictly, for example, if you switch producers first and then consumers, message loss may occur.

Before switching the remaining consumers, make sure that all messages in the old RocketMQ cluster have been consumed; otherwise, some messages may not be consumed. You can view the number of heaped messages in the old cluster to check whether consumption has completed.

## Migration process diagram



## Possible Issues

### Sequence

Message sequence cannot be guaranteed due to cluster switch.

### Message repetition

Message repetition occurs only in extreme cases. For example, if a consumer consumes a message but doesn't send an acknowledgment to the server (the old RocketMQ cluster), the message will be put into the retry queue, causing repeated consumption. Implementing the idempotency logic can avoid this issue.

### Consumption delay

During the read switch, partitions are reallocated with rebalancing between the queue and consumer client. This may cause a short consumption delay, but you don't need to handle it because it won't persist after the switch.



# Development Guide

## Message Types

### General Message

Last updated : 2024-01-18 10:07:46

General message is a basic message type, where a message is delivered to the specified topic by the producer and then consumed by the consumer subscribed to the topic. As general messages are not sequential in a topic, you can use multiple partitions to improve the message production/consumption efficiency, and they deliver the best performance when the throughput is huge.

General message is different from scheduled, delayed, sequential, and transactional message. The topics corresponding to these types of messages cannot be mixed and can only be used to send and receive messages of the same type. For example, a general message topic can only be used to send and receive general messages but not other types of messages.

# Scheduled Message and Delayed Message

Last updated : 2023-09-12 16:09:41

This document describes the concepts and usage of scheduled message and delayed message in TDMQ for RocketMQ.

## Relevant Concepts

**Scheduled message:** After a message is sent to the server, the business may want the consumer to receive it at a later time point rather than immediately. This type of message is called "scheduled message".

**Delayed message:** After a message is sent to the server, the business may want the consumer to receive it after a period of time rather than immediately. This type of message is called "delayed message".

Actually, delayed message can be regarded as a special type of scheduled message, which is essentially the same thing.

## Directions

Apache RocketMQ does not provide an API for you to freely set the delay time. In order to ensure compatibility with the open-source RocketMQ client, TDMQ for RocketMQ has designed a method to specify the message sending time by adding the property key-value pair to the message. You only need to add the `__STARTDELIVERTIME` property value to the `property` of the message that needs to be sent at a scheduled time (within 40 days). For delayed messages, you can first calculate the time point for scheduled sending and then send them as scheduled messages. A code sample is given below to show how to use scheduled and delayed messages in TDMQ for RocketMQ. You can also [view the complete sample code >>](#)

### Scheduled message

To send a scheduled message, simply write a standard millisecond timestamp to the `__STARTDELIVERTIME` property before sending it.

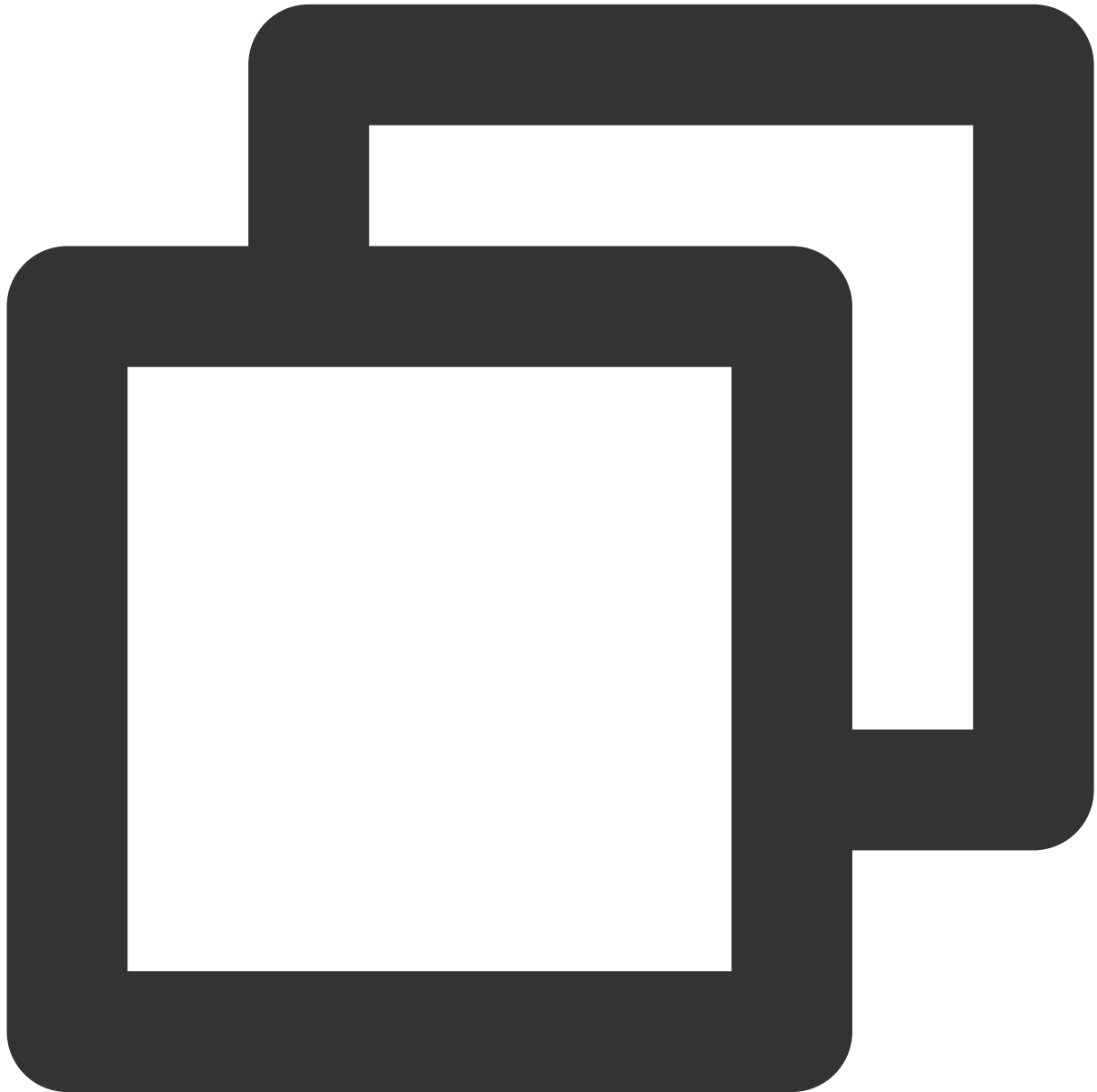


```
Message msg = new Message("test-topic", ("message content").getBytes(StandardCharsets.UTF_8));
// Set the message to be sent at 00:00:00 on 2021-10-01
try {
    long timeStamp = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").parse("10/1/2021 00:00:00");
    // Set `__STARTDELIVERTIME` into the property of `msg`
    msg.putUserProperty("__STARTDELIVERTIME", String.valueOf(timeStamp));
    SendResult result = producer.send(msg);
    System.out.println("Send delay message: " + result);
} catch (ParseException e) {
    // TODO: Add the method for handling the timestamp parsing failure
    e.printStackTrace();
}
```

```
}
```

## Delayed message

For a delayed message, its scheduled sending time point is first calculated by `System.currentTimeMillis()` + `delayTime`, and then it is sent as a scheduled message.



```
Message msg = new Message("test-topic", ("message content").getBytes(StandardCharsets.UTF_8));  
  
// Set the message to be sent after 10 seconds  
long delayTime = System.currentTimeMillis() + 10000;
```

```
// Set `__STARTDELIVERTIME` into the property of `msg`  
msg.putUserProperty("__STARTDELIVERTIME", String.valueOf(delayTime));  
  
SendResult result = producer.send(msg);  
System.out.println("Send delay message: " + result);
```

## Use Limits

When using delayed messages, make sure that the time on the client is in sync with the time on the server (UTC+8 Beijing time in all regions); otherwise, there will be a time difference.

There is a precision deviation of about 1 second for scheduled and delayed messages.

The maximum time range for scheduled and delayed messages are both 40 days.

When using scheduled messages, you need to set a time point after the current time; otherwise, the message will be sent to the consumer immediately.

# Sequential Message

Last updated : 2023-09-12 16:32:32

Sequential message is an advanced message type provided by TDMQ for RocketMQ. For a specified topic, messages are published and consumed in strict accordance with the principle of First-In-First-Out (FIFO), that is, messages sent first are consumed first, and messages sent later are consumed later.

Sequential messages are suitable for scenarios that have strict requirements on the sequence of message sending and consumption.

## Use Cases

The comparison between sequential message and general message is as follows:

Message Type	Consumption Sequence	Performance	Applicable Scenarios
General message	No sequence	High	Huge-throughput scenarios with no requirements for production and consumption sequence
Sequential message	All messages in the specific topic follow the FIFO rule	Average	Average-throughput scenarios that require publishing and consuming all messages in strict accordance with the FIFO rule

Sequential messages are often used in the following business scenarios:

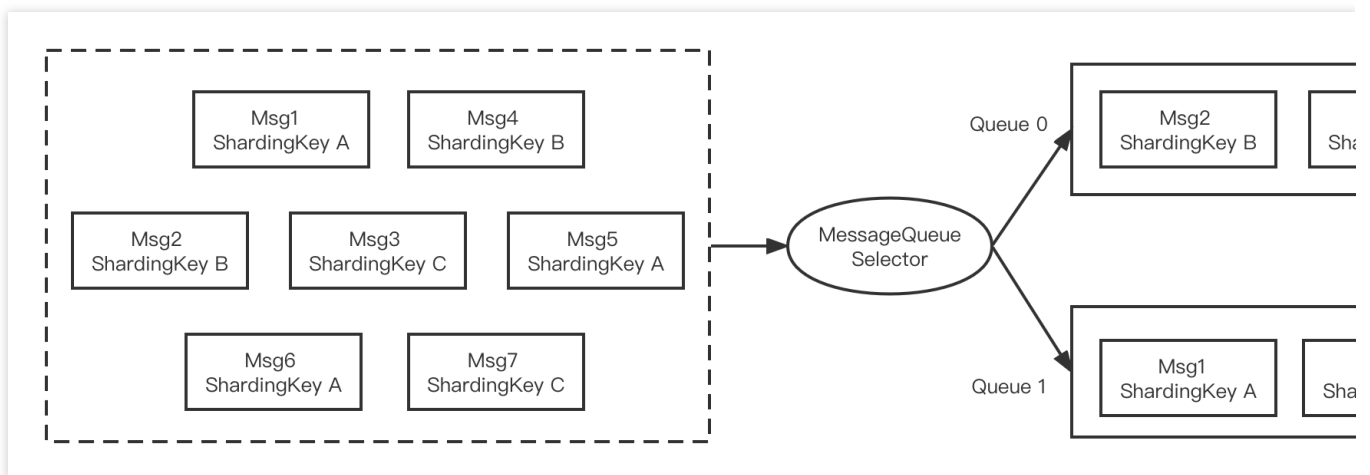
**Order creation:** In some ecommerce systems, an order's creation, payment, refund, and logistics messages must be produced or consumed in strict sequence, No the order status will be messed up during consumption, which will affect the normal operation of the business. Therefore, the messages of this order must be produced and consumed in a certain sequence in the client and message queue. At the same time, the messages are sequentially dependent, and the processing of the next message must be dependent on the processing result of the preceding message.

**Log sync:** In the scenario of sequential event processing or real-time incremental data sync, sequential messages can also play a greater role. For example, it is necessary to ensure that database operations are in sequence when MySQL binlogs are synced.

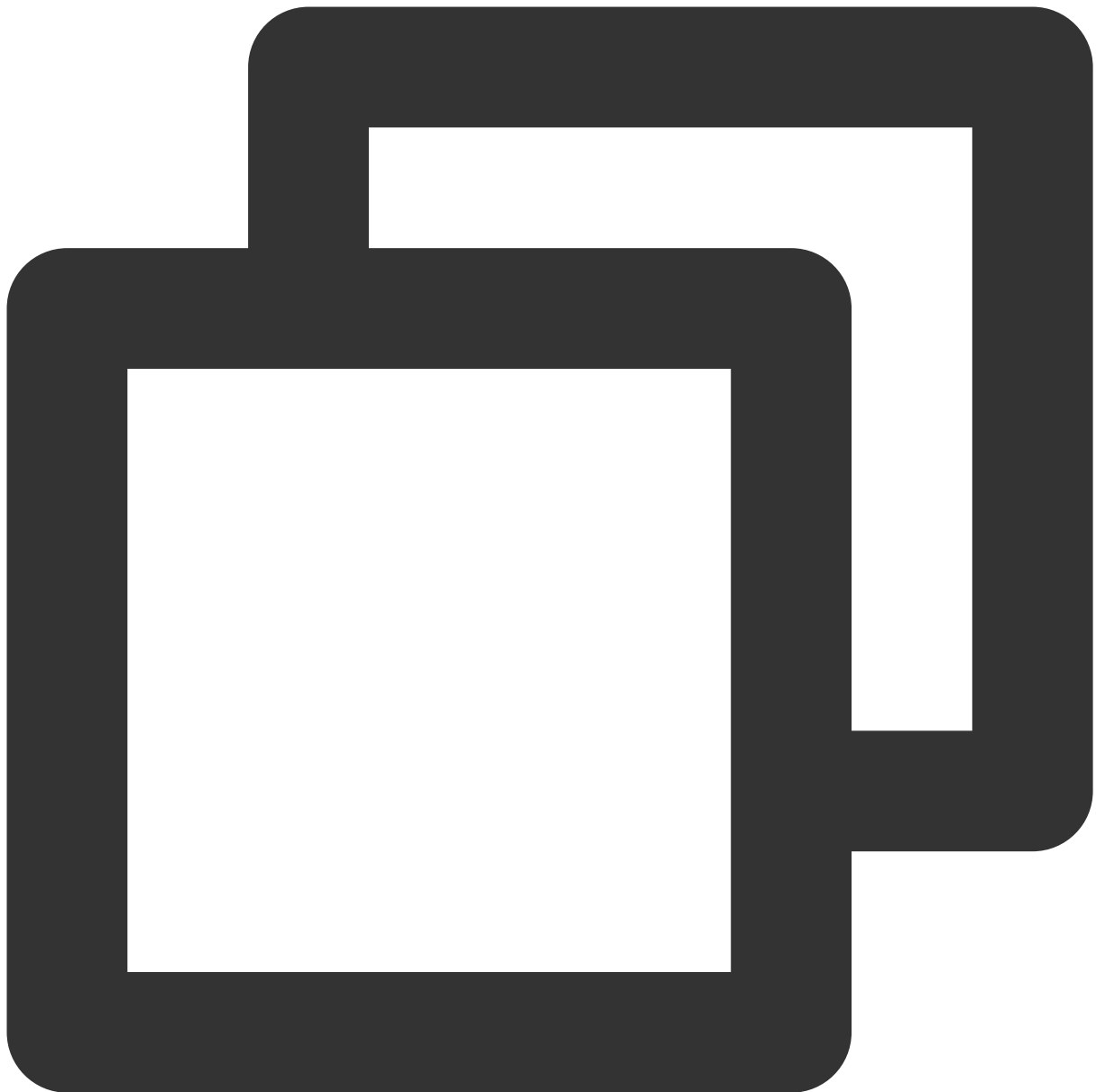
**Financial scenarios:** In some matchmaking transaction scenarios like certain securities transactions, the first bidder is given priority in the case of the same bidding price, so it is necessary to produce and consume sequential messages in a FIFO manner.

## How It Works

In TDMQ for RocketMQ, the principle of sequential messages is shown in the figure below. You can partition messages according to a certain standard (such as ShardingKey in the figure), and messages of the same ShardingKey will be assigned to the same queue and consumed in sequence.



The code of sequential message is as shown below:



```
public class Producer {
    public static void main(String[] args) throws UnsupportedEncodingException {
        try {
            DefaultMQProducer producer = new DefaultMQProducer("please_rename_uniquely");
            producer.start();

            String[] tags = new String[] {"TagA", "TagB", "TagC", "TagD", "TagE"};
            for (int i = 0; i < 100; i++) {
                int orderId = i % 10;
                Message msg =
                    new Message("TopicTest", tags[i % tags.length], "KEY" + i,
```



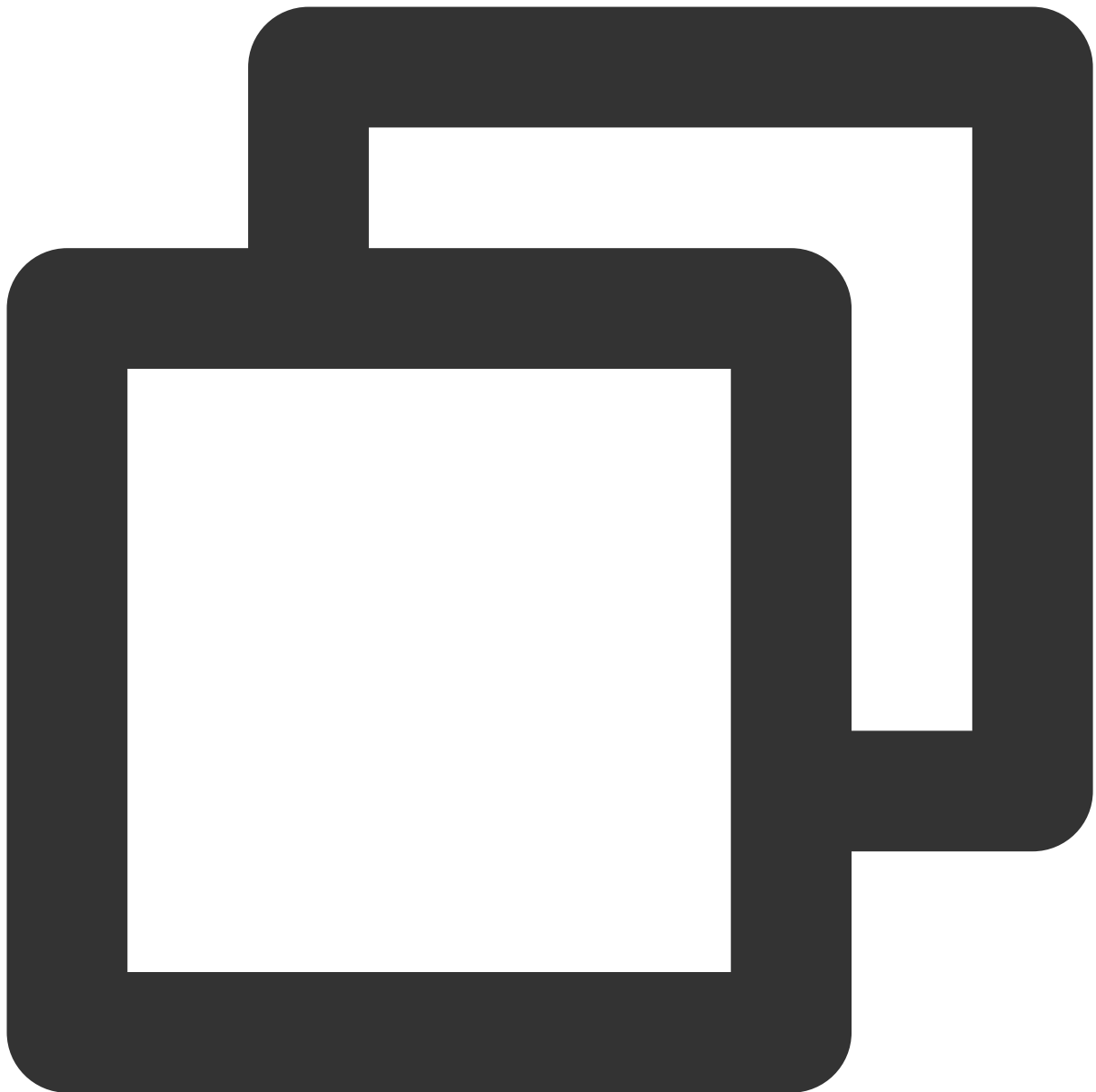
```
        ("Hello RocketMQ " + i).getBytes(RemotingHelper.DEFAULT_CHARSET);
        SendResult sendResult = producer.send(msg, new MessageQueueSelector<Object>() {
            @Override
            public MessageQueue select(List<MessageQueue> mqs, Message msg,
                Integer id = (Integer) arg;
                int index = id % mqs.size();
                return mqs.get(index);
            }
        }, orderId);

        System.out.printf("%s%n", sendResult);
    }

    producer.shutdown();
} catch (MQClientException | RemotingException | MQBrokerException | InterruptedException) {
    e.printStackTrace();
}
}
```

The main difference here is that the `SendResult send(Message msg, MessageQueueSelector selector, Object arg)` method is called, `MessageQueueSelector` is the queue selector, and `arg` is a Java object, which can be passed in as the classification standard of the message sending partition.

The `MessageQueueSelector` API is as follows:



```
public interface MessageQueueSelector {  
    MessageQueue select(final List<MessageQueue> mqs, final Message msg, final Object arg)  
}
```

Among them, `mqs` is the queue that can be sent, `msg` is the message, `arg` is the object passed in the above `send` API, and the queue to which the message needs to be sent is returned. In the above sample, `orderId` is used as the partition classification standard, and the remainder of all queue numbers is used to send messages with the same `orderId` to the same queue.

In the production environment, we recommend that you select the most fine-grained partition key for splitting. For example, when the order ID and user ID are used as the partition key keywords, the messages of the same end user will be processed in sequence, while those of different users will not.

**Note**

In order to ensure the high availability of messages, TDMQ for RocketMQ currently doesn't support "globally sequential messages" in a single queue (if you have already created globally sequential messages, you can use them normally); if you want to ensure global sequence, you can use consistent ShardingKey to do so.

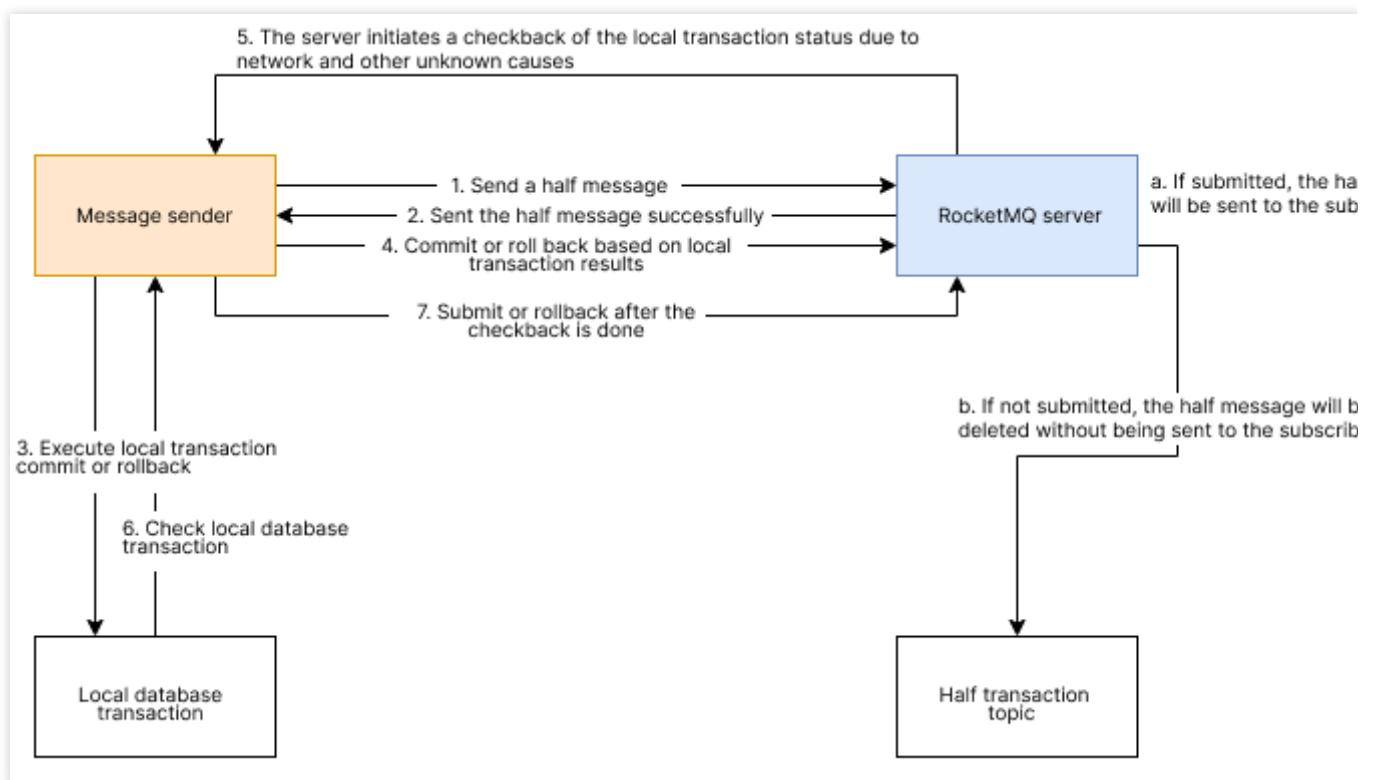
# Transactional Message

Last updated : 2023-04-14 16:59:12

This document describes the concept, technical principle, use cases, and usage methods of transactional messages in the TDMQ for RocketMQ.

## Description

The transactional message solves the atomicity problem of local transaction execution and message sending, ensuring the eventual consistency between them. It provides users with the distributed transaction feature similar to X/Open XA, so users can achieve the eventual consistency of the distributed transaction in TDMQ for RocketMQ.



1. The producer sends a message to RocketMQ (1).
2. After receiving the message, the server stores the message in the half message topic (2).
3. Local transaction execution is done (3).
4. The producer actively sends the transaction execution result to RocketMQ (4).
5. If the local transaction execution result has not been returned after a certain period of time, RocketMQ will execute the checkback logic (5).

6. After receiving the message checkback, the producer needs to check the final result of the local transaction execution of the corresponding message and give feedback (6, 7). There are three transaction execution status:
- TransactionStatus.COMMIT: Commits the transaction, and consumers can consume the message.
  - TransactionStatus.ROLLBACK: Rolls back the transaction, and the message is discarded without being consumed by consumers.
  - TransactionStatus.UN\_KNOW: Unknown status, indicating the waiting of another checkback.
7. When the transaction is successfully executed, RocketMQ submits the transactional message to the real topic for consumption by consumers (a).

## Use Cases

The transaction messages of TDMQ for RocketMQ can be used to process transactions, which can greatly improve processing efficiency and performance. A billing system often has a long transaction linkage with a significant chance of error or timeout. TDMQ's automated repush and abundant message retention features can be used to provide transaction compensation, and the eventual consistency of payment tips notifications and transaction pushes can also be achieved through TDMQ.

# Message Filtering

Last updated : 2023-10-19 11:02:16

This document describes the message filtering feature of TDMQ for RocketMQ and its use cases and usage instructions.

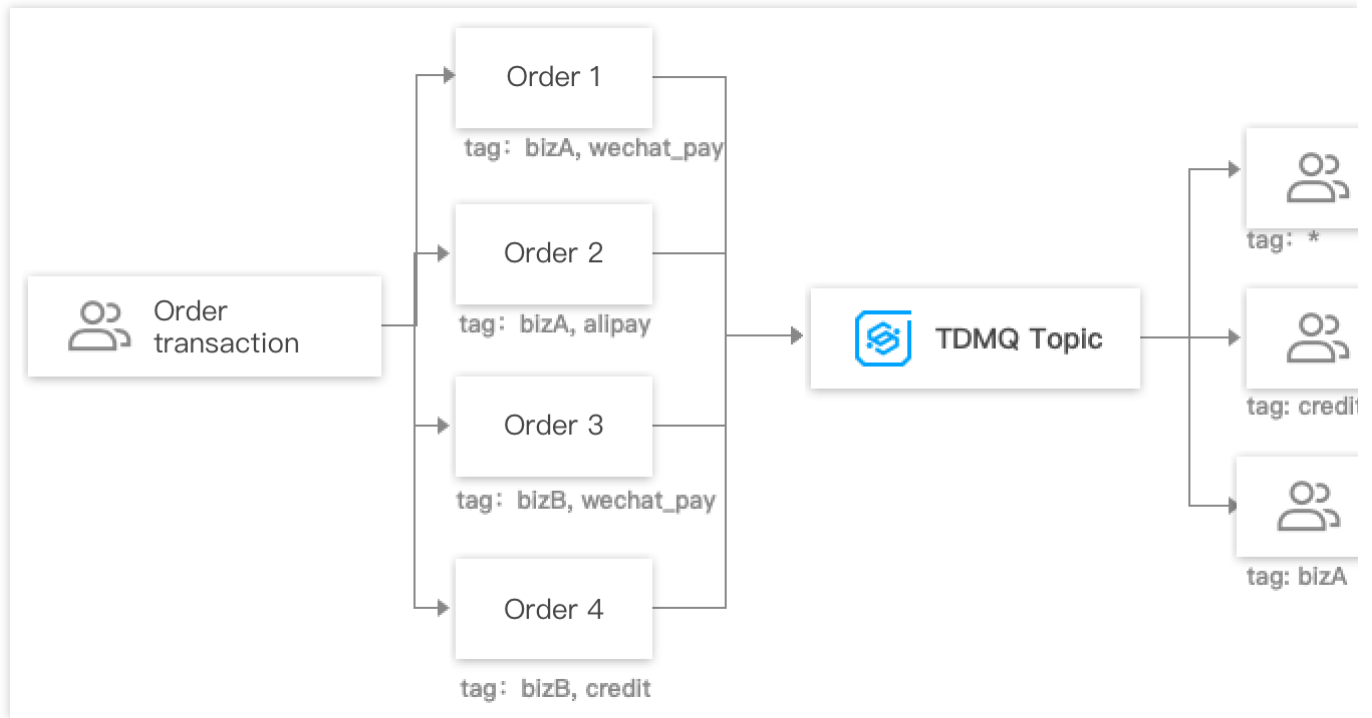
## Feature Overview

Message filtering means filtering messages by the message attribute. The message producer can configure message attributes to group messages before sending them to a topic, and the consumer that subscribes to the topic can filter messages based on their attributes so that only eligible messages are delivered to the consumer for consumption. If a consumer sets no filter conditions when subscribing to a topic, no matter whether filter attributes are set during message sending, all messages in the topic will be delivered to the consumer for consumption.

## Use Cases

Generally, messages with the same business attributes are stored in the same topic. For example, when an order transaction topic contains messages of order placements, payments, and deliveries, and if you want to consume only one type of transaction messages in your business, you can filter them on the client, but this will waste bandwidth resources.

To solve this problem, TDMQ supports message filtering on the broker. You can set one or more tags during message production and subscribe to specified tags during consumption.

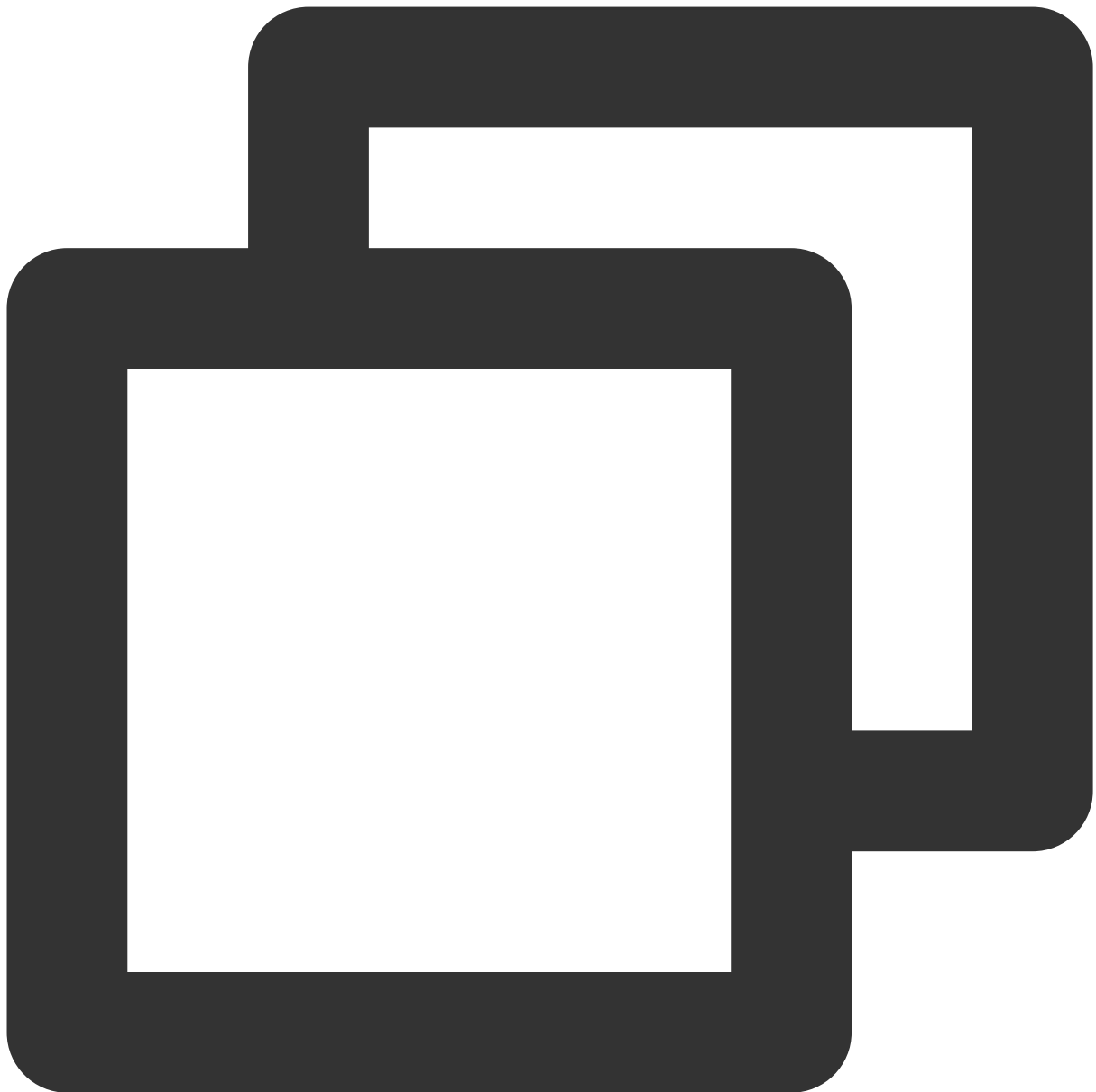


## Usage Instructions

### Filtering by tag

### Sending messages

You must specify tags for each message when sending it.

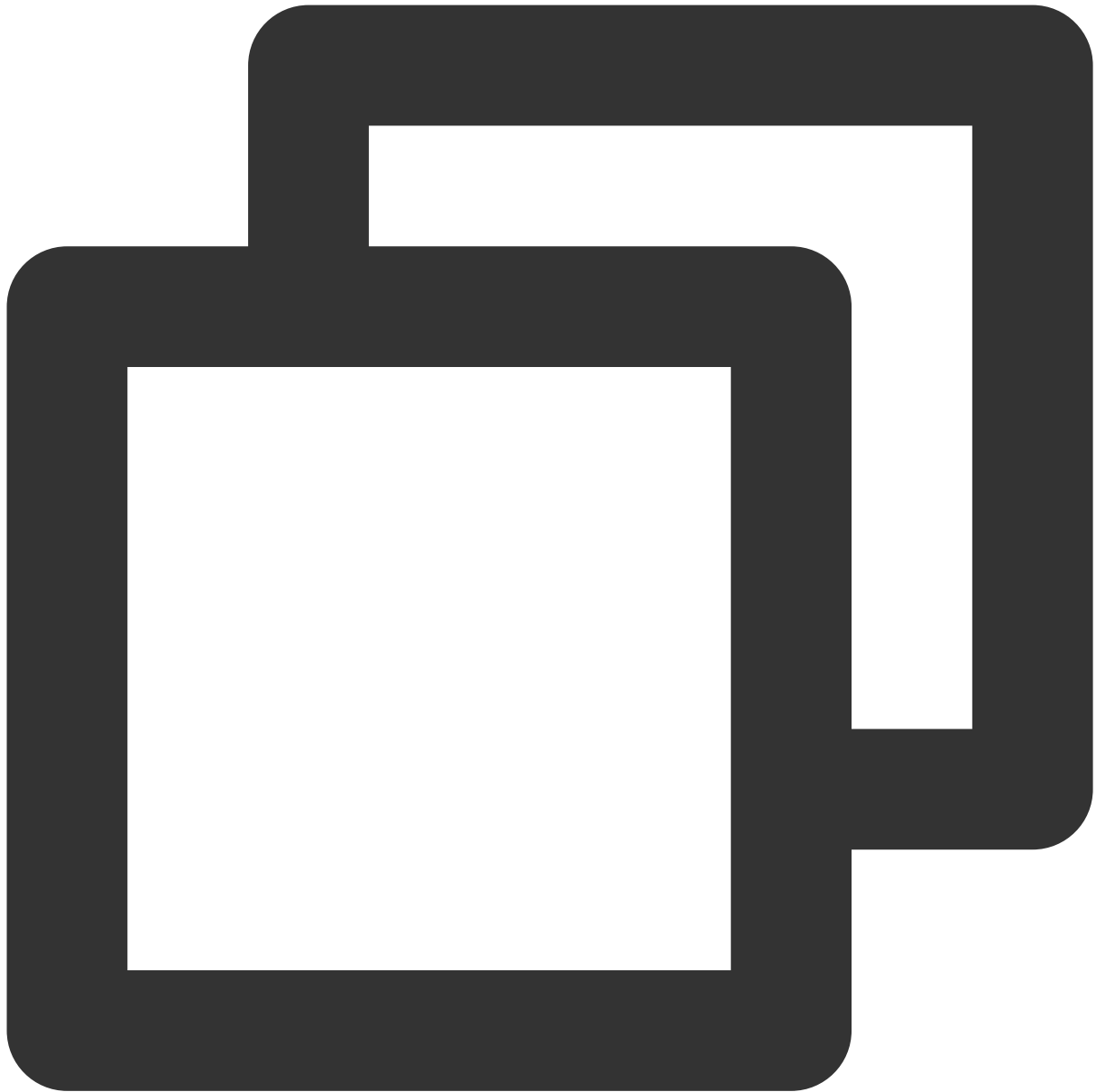


```
Message msg = new Message("TOPIC", "TagA", "Hello world".getBytes());
```

### Subscribing to messages

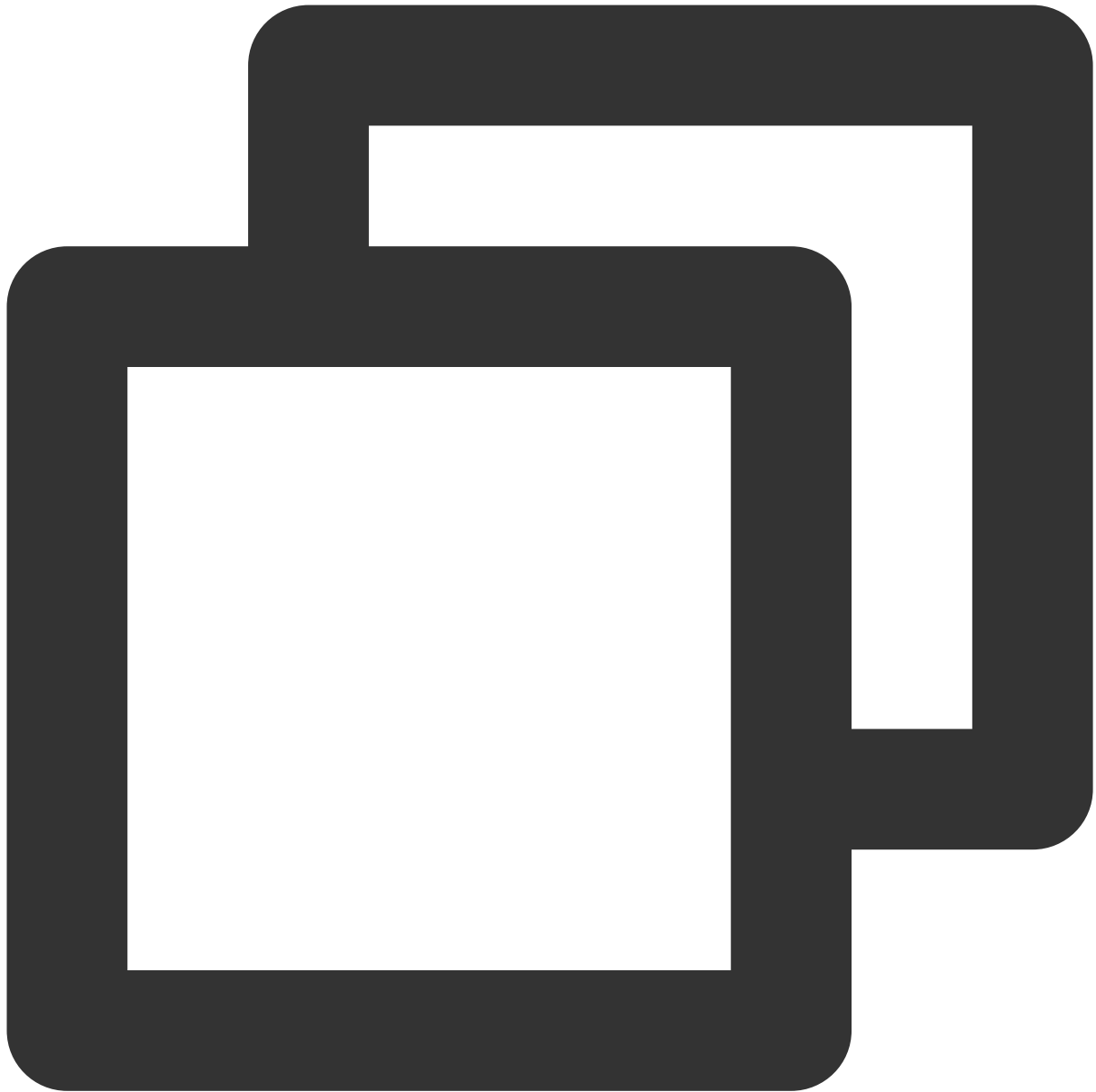
Subscribing to all tags: If a consumer wants to subscribe to all types of messages under a topic, an asterisk (\*) can be used to represent all tags.





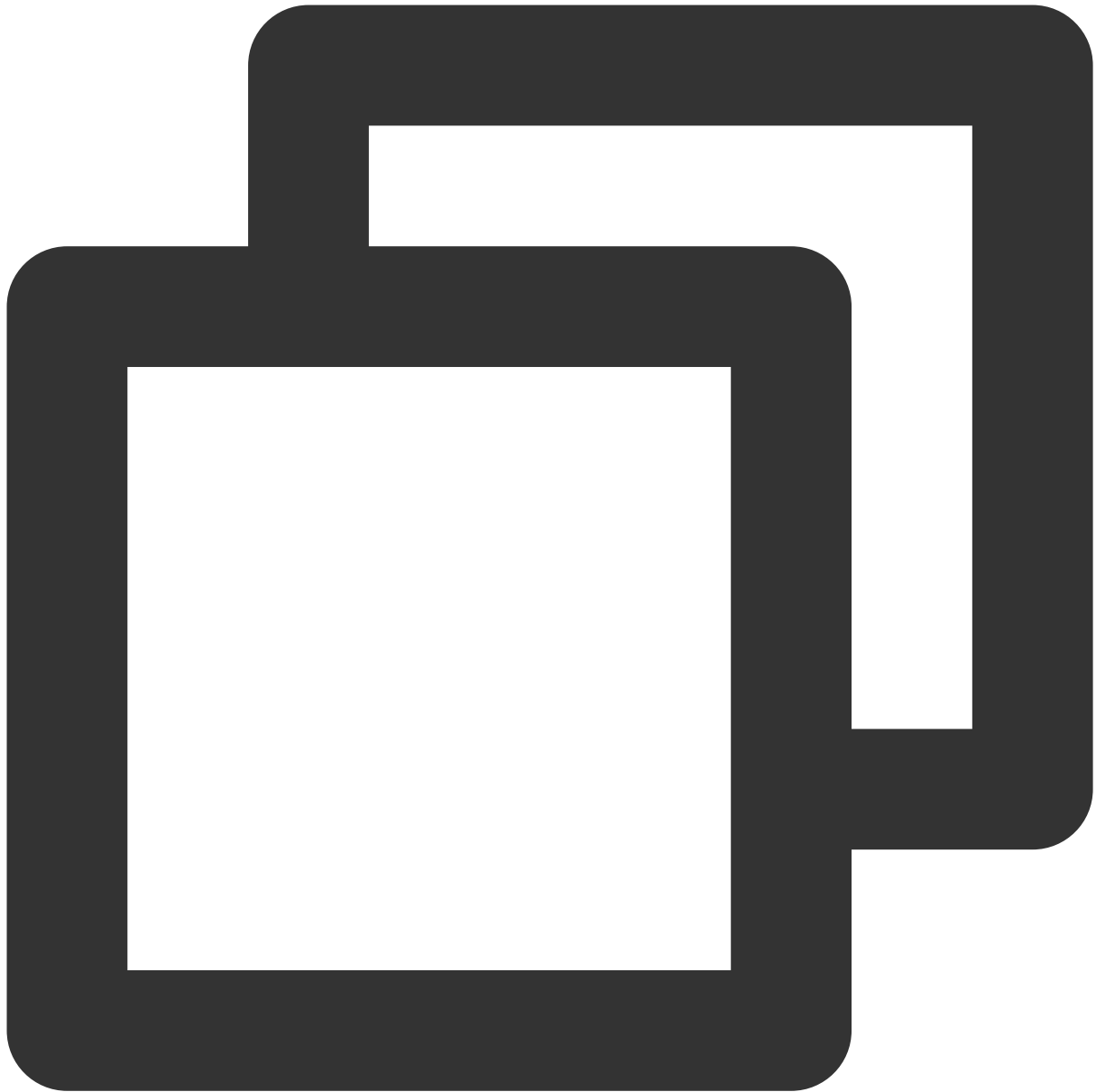
```
consumer.subscribe("TOPIC", "*", new MessageListener() {  
    public Action consume(Message message, ConsumeContext context) {  
        System.out.println(message.getMsgID());  
        return Action.CommitMessage;  
    }  
});
```

Subscribing to one tag: If a consumer wants to subscribe to a certain type of messages under a topic, the tag should be specified clearly.



```
consumer.subscribe("TOPIC", "TagA", new MessageListener() {  
    public Action consume(Message message, ConsumeContext context) {  
        System.out.println(message.getMsgID());  
        return Action.CommitMessage;  
    }  
});
```

Subscribing to multiple tags: If a consumer wants to subscribe to multiple types of messages under a topic, two vertical bars (||) should be added between the two tags for separation.

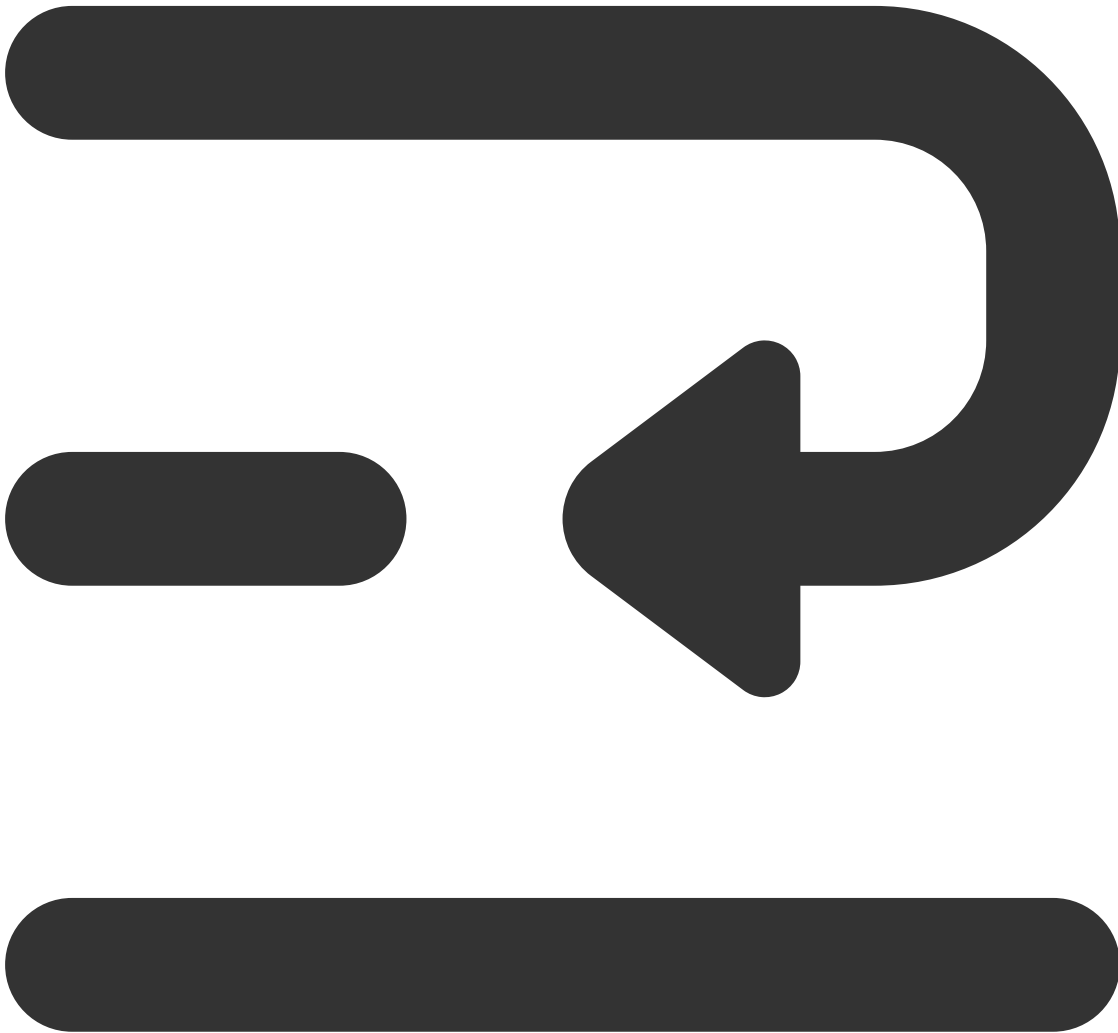


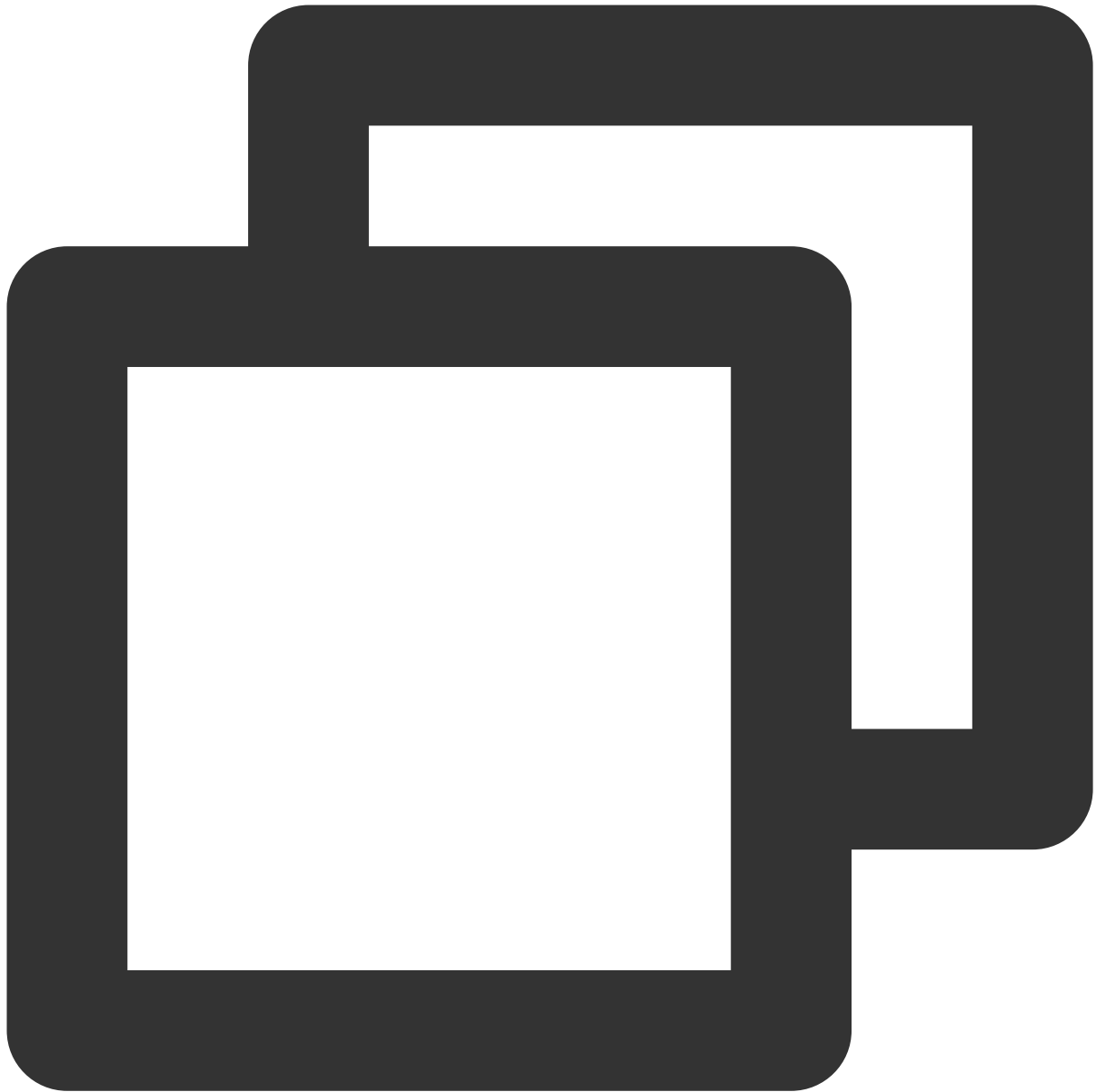
```
consumer.subscribe("TOPIC", "TagA||TagB", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        System.out.println(message.getMsgID());
        return Action.CommitMessage;
    }
});
```

## Filtering by SQL

## Sending messages

The message sending code here is basically the same as the code for sending simple messages. Here, a message is allowed to carry multiple user-defined attributes when constructing the message body.

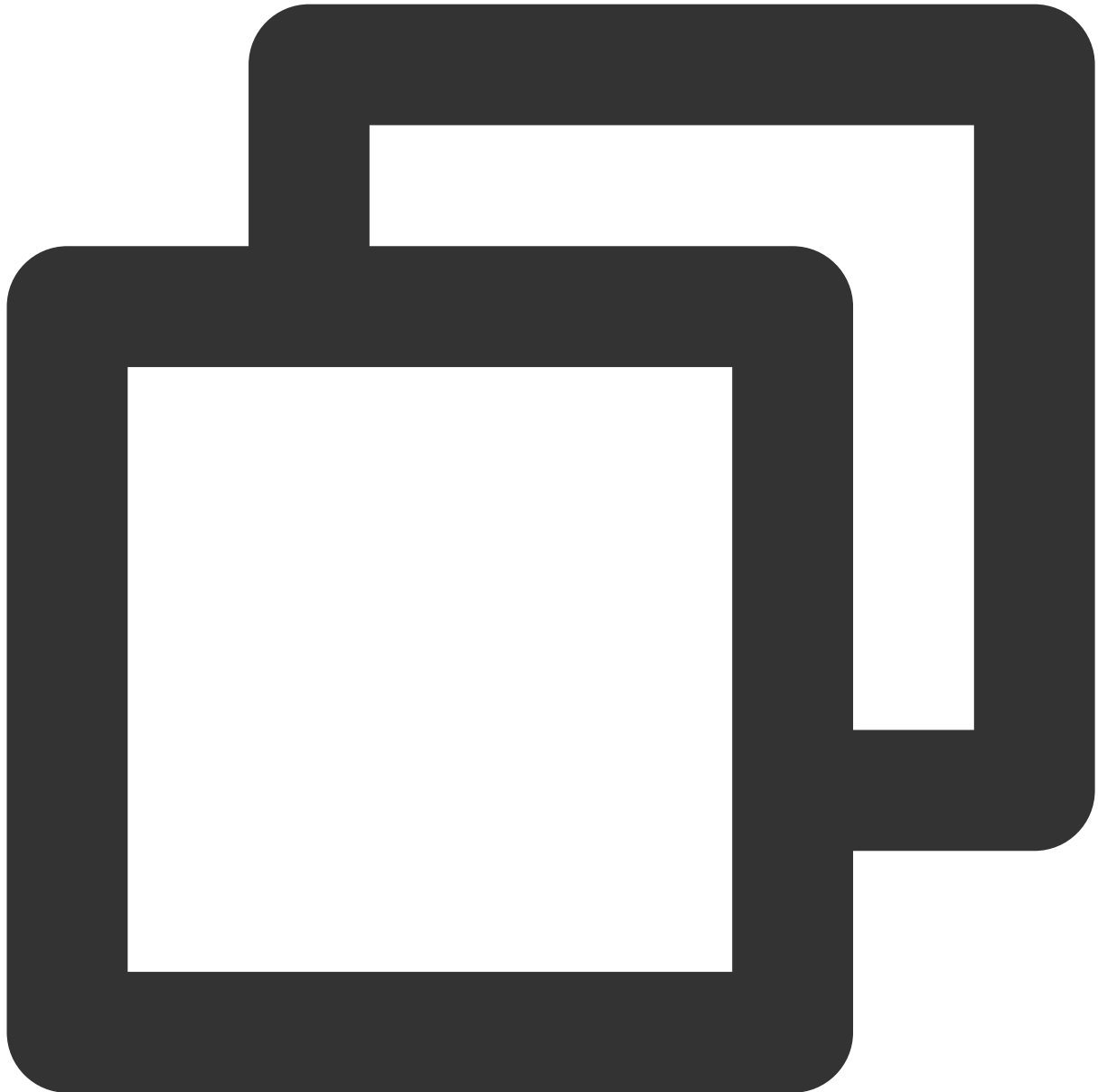




```
int totalMessagesToSend = 5;
for (int i = 0; i < totalMessagesToSend; i++) {
    Message msg = new Message(TOPIC_NAME, "Hello RocketMQ.".getBytes(StandardCharset
    msg.putUserProperty("key1", "value1");
    // Send the message
    SendResult sendResult = producer.send(message);
    System.out.println("sendResult = " + sendResult);
}
```

### Subscribing to messages

The message consumption code here is basically the same as the code for consuming simple messages. However, a message needs to be carried with the corresponding SQL expression when being subscribed to.



```
// Subscribe to all messages
pushConsumer.subscribe(TOPIC_NAME, MessageSelector.bySql("True"));

// Subscribe to single-key SQL expression when subscribing to a topic
//pushConsumer.subscribe(TOPIC_NAME,    MessageSelector.bySql("key1 IS NOT NULL AND

// Subscribe to multiple attributes
//pushConsumer.subscribe(TOPIC_NAME,    MessageSelector.bySql("key1 IS NOT NULL AND
```

```
// Register a callback implementation class to process messages pulled from the bro
pushConsumer.registerMessageListener((MessageListenerConcurrently) (msgs, context)
    // Message processing logic
    System.out.printf("%s Receive New Messages: %s %n", Thread.currentThread()
    // Mark the message as being successfully consumed and return the consump
    return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
});
// Start the consumer instance
pushConsumer.start();
```

**Note**

Above is a brief introduction to message publishing and subscription. For more information, see

[TencentCloud/rocketmq-demo](#) or [Apache RocketMQ documentation](#).

# Consumption Mode

Last updated : 2023-09-13 11:40:40

This document describes the features and use cases of clustering consumption and broadcasting consumption in TDMQ for RocketMQ.

## Feature Overview

Cluster consumption: if the cluster consumption mode is used, any message only needs to be processed by any consumer in the cluster.

Broadcast consumption: if the broadcast consumption mode is used, each message will be pushed to all registered consumers in the cluster to ensure that the message is consumed by each consumer at least once.

## Use Cases

Clustering consumption is suitable for scenarios where each message only needs to be processed once.

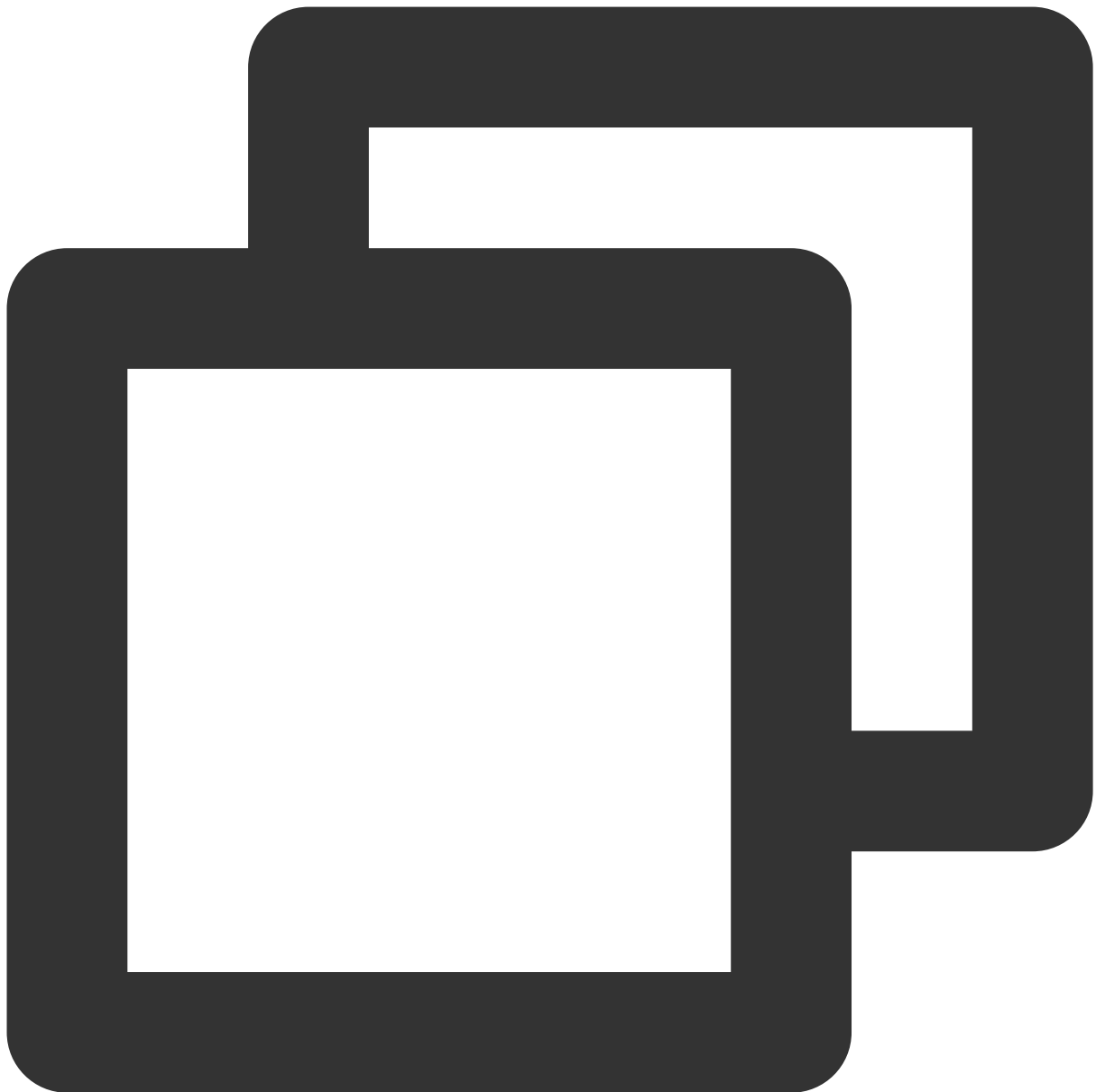
Broadcasting consumption is suitable for scenarios where each message needs to be processed by each consumer in the cluster.

## Sample Codes

### Cluster subscription

All consumers identified by the same group ID will evenly share messages for consumption. For example, if a topic has nine messages, and a group ID identifies three consumer instances, then each instance will consume only three messages evenly in the clustering consumption mode.





```
// Set the cluster subscription mode (which is the default mode if you don't specify  
properties.put(PropertyKeyConst.MessageModel, PropertyValueConst.CLUSTERING);
```

### **Broadcast subscription**

A message will be consumed once by all consumers identified by the same group ID. In the broadcasting consumption mode, for example, if a topic has nine messages and a group ID identifies three consumer instances, each instance will consume nine messages.



```
// Set the broadcast subscription mode
properties.put(PropertyKeyConst.MessageModel, PropertyValueConst.BROADCASTING);
```

**Note**

You need to ensure that all consumer instances under the same group ID have the same subscription relationships.

# Message Retry

Last updated : 2023-09-13 11:41:02

This document describes the message retry mechanisms and their usages in TDMQ for RocketMQ.

## Feature Overview

When a message is consumed for the first time by a consumer and fails to get a normal response, or when it is requested by users to deliver again in the server, TDMQ for RocketMQ will automatically retry delivering this message through the message retry mechanism until it is consumed successfully. When the number of retries reaches the specified value but the message is still not consumed successfully, retry will stop, and the message will be delivered to the dead letter queue.

After the message enters the dead letter queue, TDMQ for RocketMQ can no longer process it automatically. At this point, human intervention is generally required. You can write a dedicated client to subscribe to the dead letter queue to process such messages.

### Note

The broker will automatically retry in the cluster consumption mode but not the broadcast consumption mode.

The following results are considered as consumption failure, and the message will be retried accordingly:

1. The consumer returns `ConsumeConcurrentlyStatus.RECONSUME_LATER` .
2. The consumer returns null.
3. The consumer actively/passively throws an exception.

## Number of Retries

When a message needs to be retried in TDMQ for RocketMQ, set the "messageDelayLevel" parameter as follows to configure the number of retries and retry intervals:



```
messageDelayLevel=1s 5s 10s 30s 1m 2m 3m 4m 5m 6m 7m 8m 9m 10m 20m 30m 1h 2h
```

The number of retries and retry intervals have the following relationships:

Retry No.	Time Interval Since Last Retry	Retry No.	Time Interval Since Last Retry
1	1 second	10	6 minutes
2	5 seconds	11	7 minutes
3	10 seconds	12	8 minutes

---

4	30 seconds	13	9 minutes
5	1 minute	14	10 minutes
6	2 minutes	15	20 minutes
7	3 minutes	16	30 minutes
8	4 minutes	17	1 hour
9	5 minutes	18	2 hours

## Instructions

If you need to adjust the number of retries by yourself, you can set the parameters of the consumer.



```
pushConsumer.setMaxReconsumeTimes(3);
```

# Dead Letter Queue

Last updated : 2023-09-12 16:42:15

This document describes the dead letter queues and their usages in TDMQ for RocketMQ.

## Feature Overview

When a message is consumed for the first time by a consumer and fails to get a normal response, or when it is requested by users to deliver again in the server, TDMQ for RocketMQ will automatically retry delivering this message through the message retry mechanism until it is consumed successfully. When the number of retries reaches the specified value but the message is still not consumed successfully, retry will stop, and the message will be delivered to the dead letter queue.

After the message enters the dead letter queue, TDMQ for RocketMQ can no longer process it automatically. At this point, human intervention is generally required. You can write a dedicated client to subscribe to the dead letter queue to process such messages.

## Notes

Messages in the dead letter queue must be processed manually or by new code logic, whereas messages in the retry queue can be consumed automatically.

Messages in the dead letter queue are only valid for three days by default and are deleted after that.

The dead letter queue starts with %DLQ%, which corresponds to the consumer group one by one. Therefore, a dead letter queue contains all the dead letter messages corresponding to the group ID, no matter which topic the message belongs to.

# SDK Documentation

## Access over TCP

## Spring Boot Starter

# Sending and Receiving General Messages

Last updated : 2023-10-19 11:04:14

## 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 or prepared 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](#).

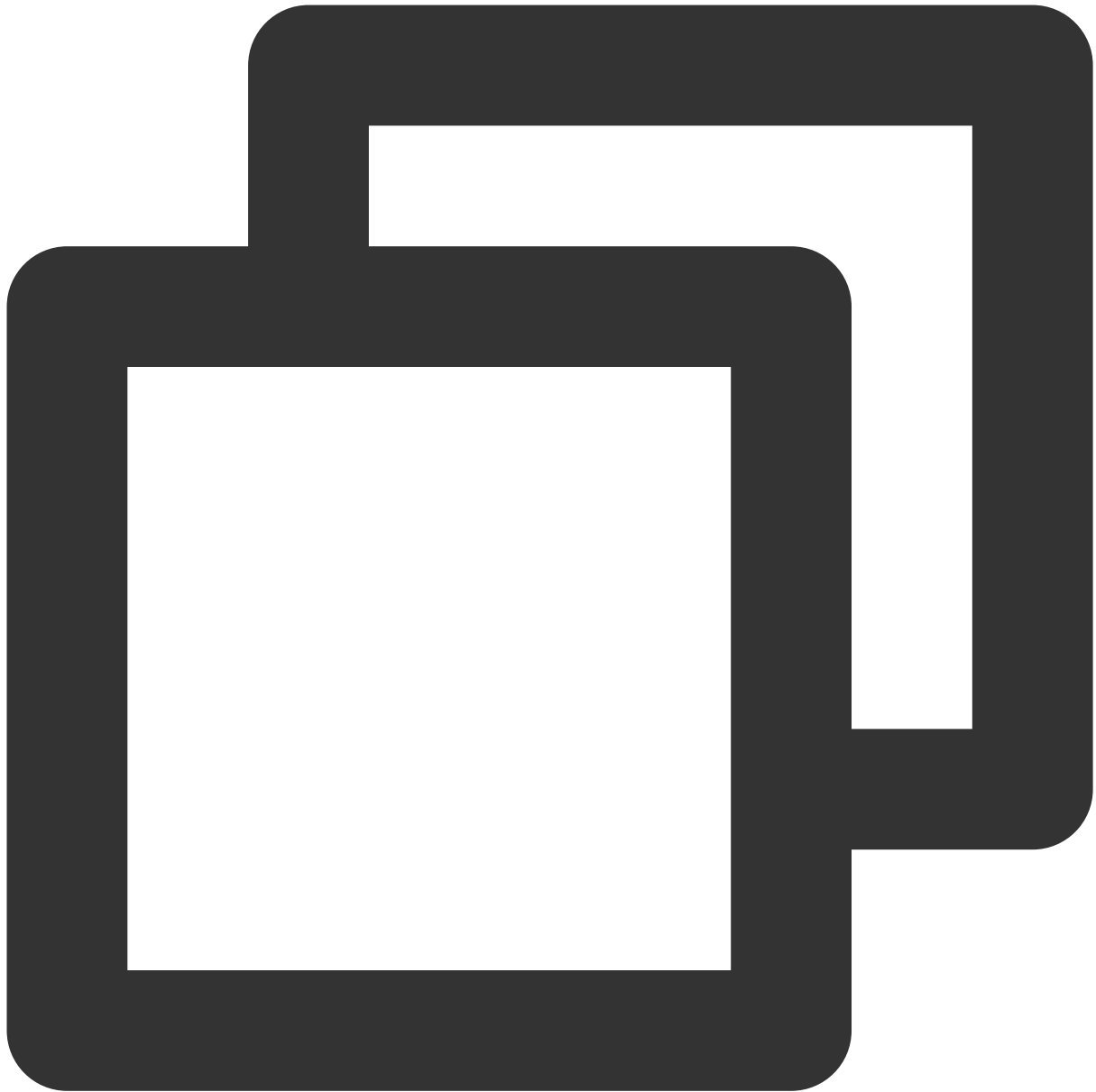
You have [downloaded the demo](#) or obtained the demo in [TencentCloud/rocketmq-demo](#) in GitHub.

## Directions

### Step 1. Add dependencies

Add dependencies to the `pom.xml` file.

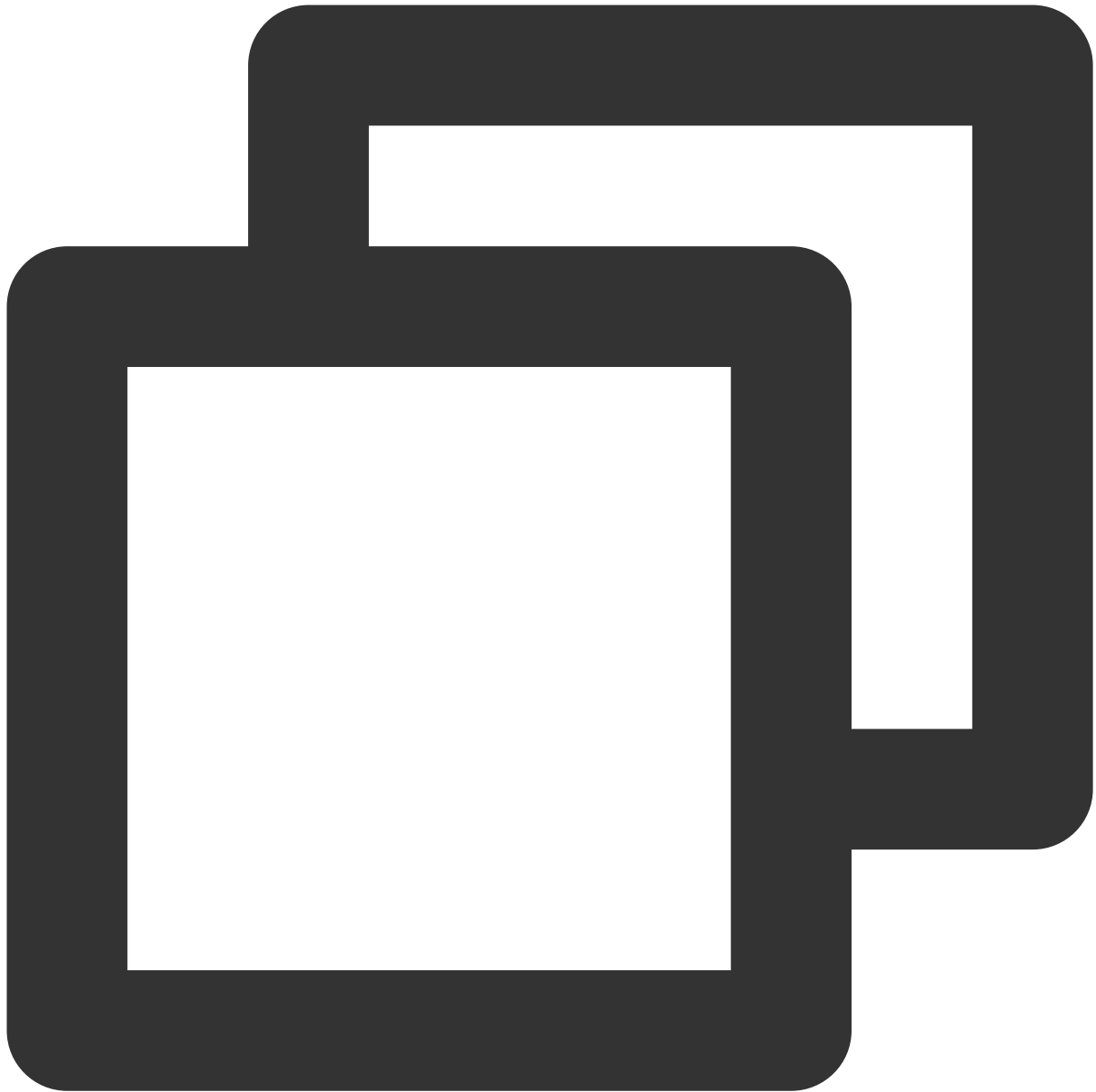




```
<dependency>
  <groupId>org.apache.rocketmq</groupId>
  <artifactId>rocketmq-spring-boot-starter</artifactId>
  <version>2.2.2</version>
</dependency>
```

## Step 2. Prepare configurations

Add configuration information to the configuration file.

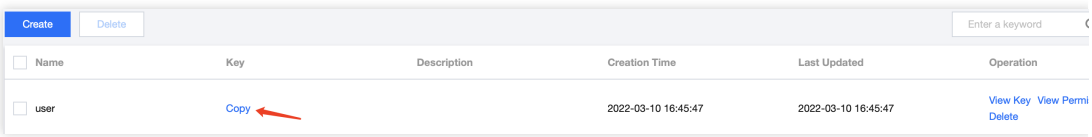


```
server:
  port: 8082

# RocketMQ configuration information
rocketmq:
  # Service access address of TDMQ for RocketMQ
  name-server: rocketmq-xxx.rocketmq.ap-bj.public.tencenttdmq.com:9876
  # Producer configurations
  producer:
    # Producer group name
    group: group111
```

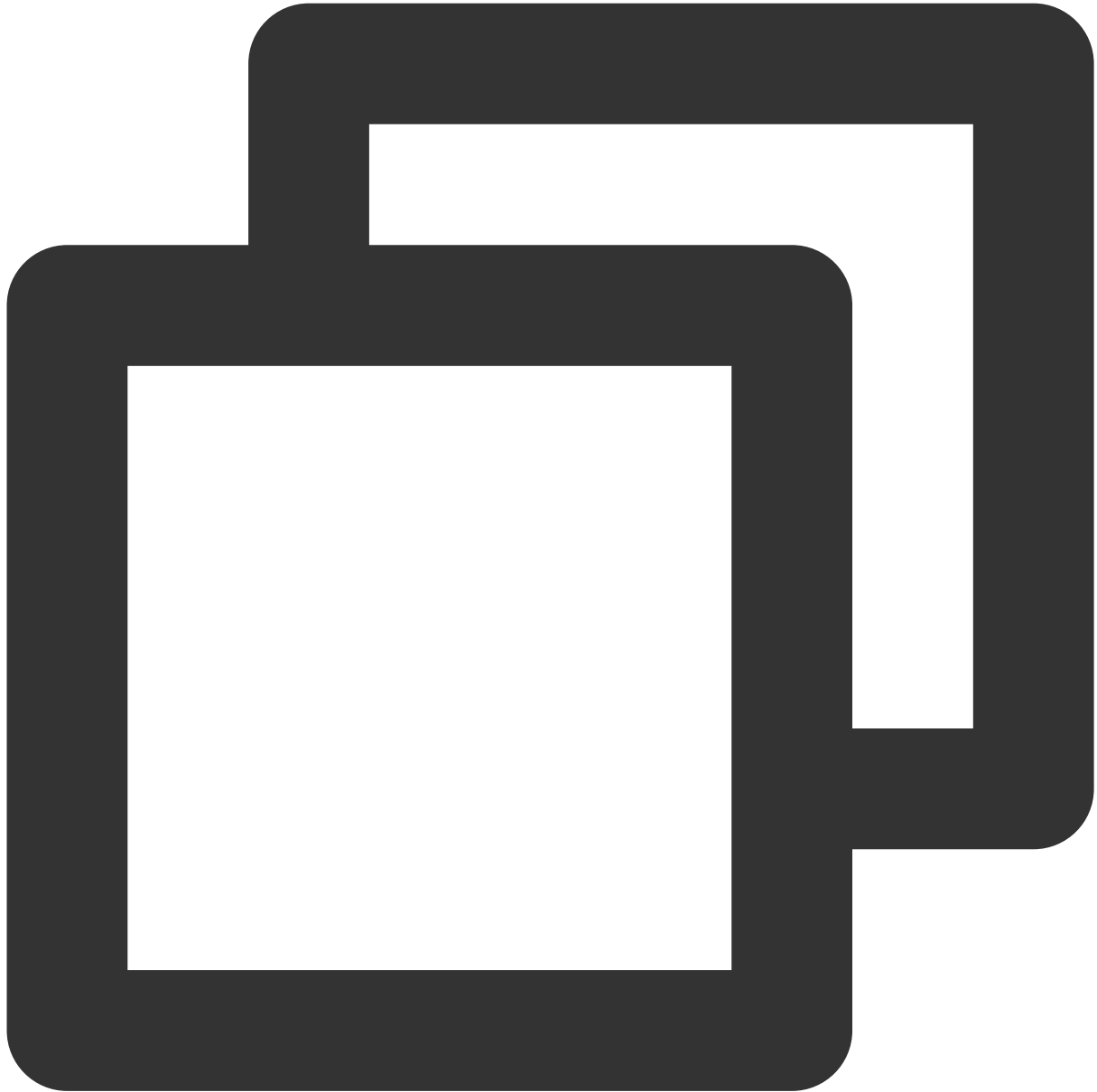
```
# Role token
access-key: eyJrZXlJZC....
# Name of the authorized role
secret-key: admin
# Common configurations for the consumer
consumer:
  # Role token
  access-key: eyJrZXlJZC....
  # Name of the authorized role
  secret-key: admin

# Custom configurations based on business needs
namespace: rocketmq-xxx|namespace1
producer1:
  topic: testdev1
consumer1:
  group: group111
  topic: testdev1
  subExpression: TAG1
consumer2:
  group: group222
  topic: testdev1
  subExpression: TAG2
```

Parameter	Description
name-server	Cluster access address, which can be obtained from <b>Access Address</b> in the <b>Operation</b> column <b>Cluster</b> page in the console. The namespace access address can be obtained under the <b>Names</b> on the <b>Cluster</b> page.
group	Consumer group name, which can be copied under the <b>Group</b> tab on the <b>Cluster</b> page in the co
secret-key	Role name, which can be copied on the <a href="#">Role Management</a> page.
access-key	Role token, which can be copied in the <b>Token</b> column on the <a href="#">Role Management</a> page. 
namespace	Namespace name, which can be copied under the <b>Namespace</b> tab on the <b>Cluster</b> page in the c
topic	Topic name, which can be copied under the <b>Topic</b> tab on the <b>Cluster</b> page in the console.
subExpression	A parameter used to set the message tag.

### Step 3. Send messages

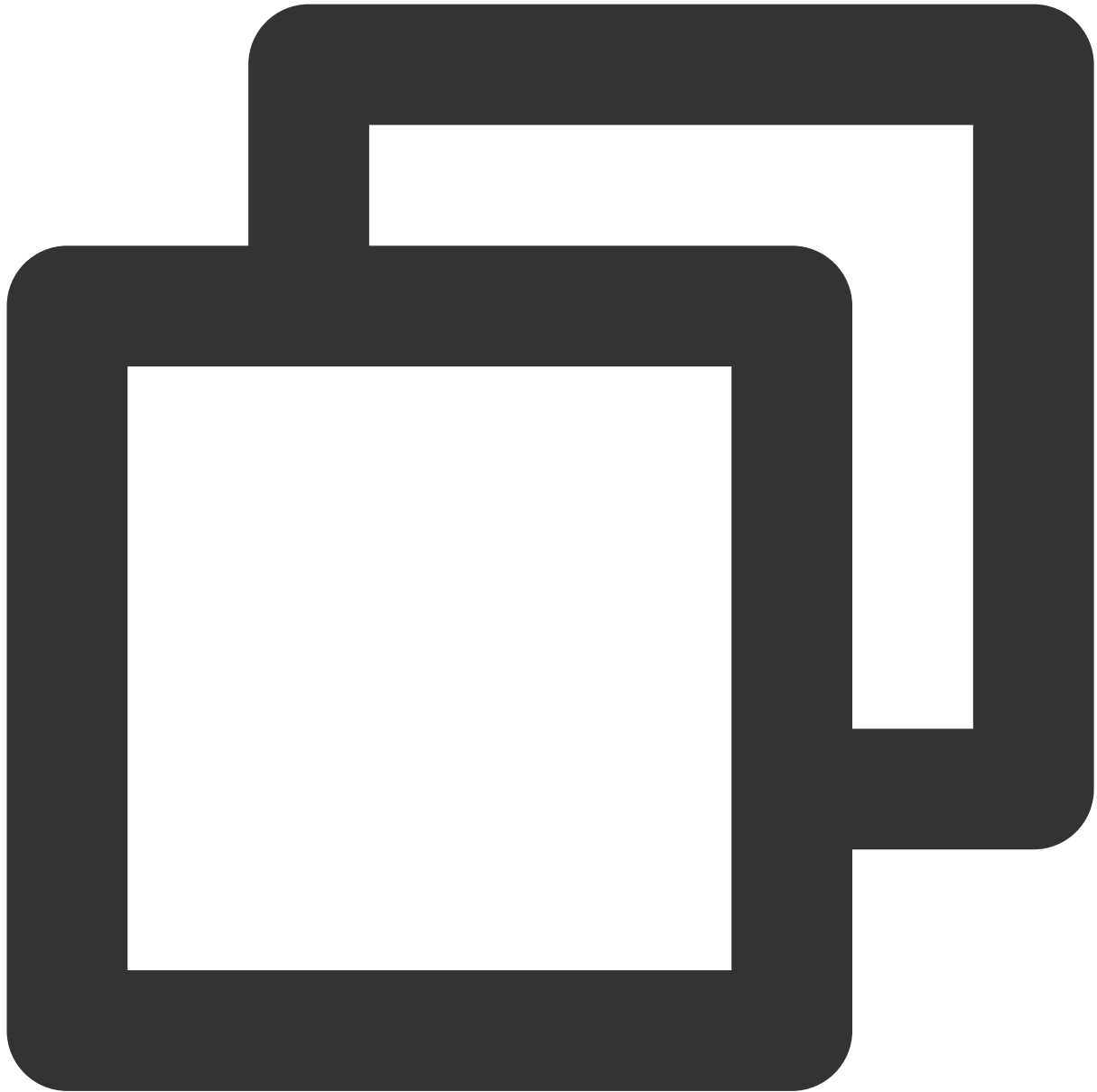
1. Inject `RocketMQTemplate` into the class that needs to send messages.



```
@Value("${rocketmq.namespace}%${rocketmq.producer1.topic}")
private String topic; // Full topic name, which needs to be concatenated.

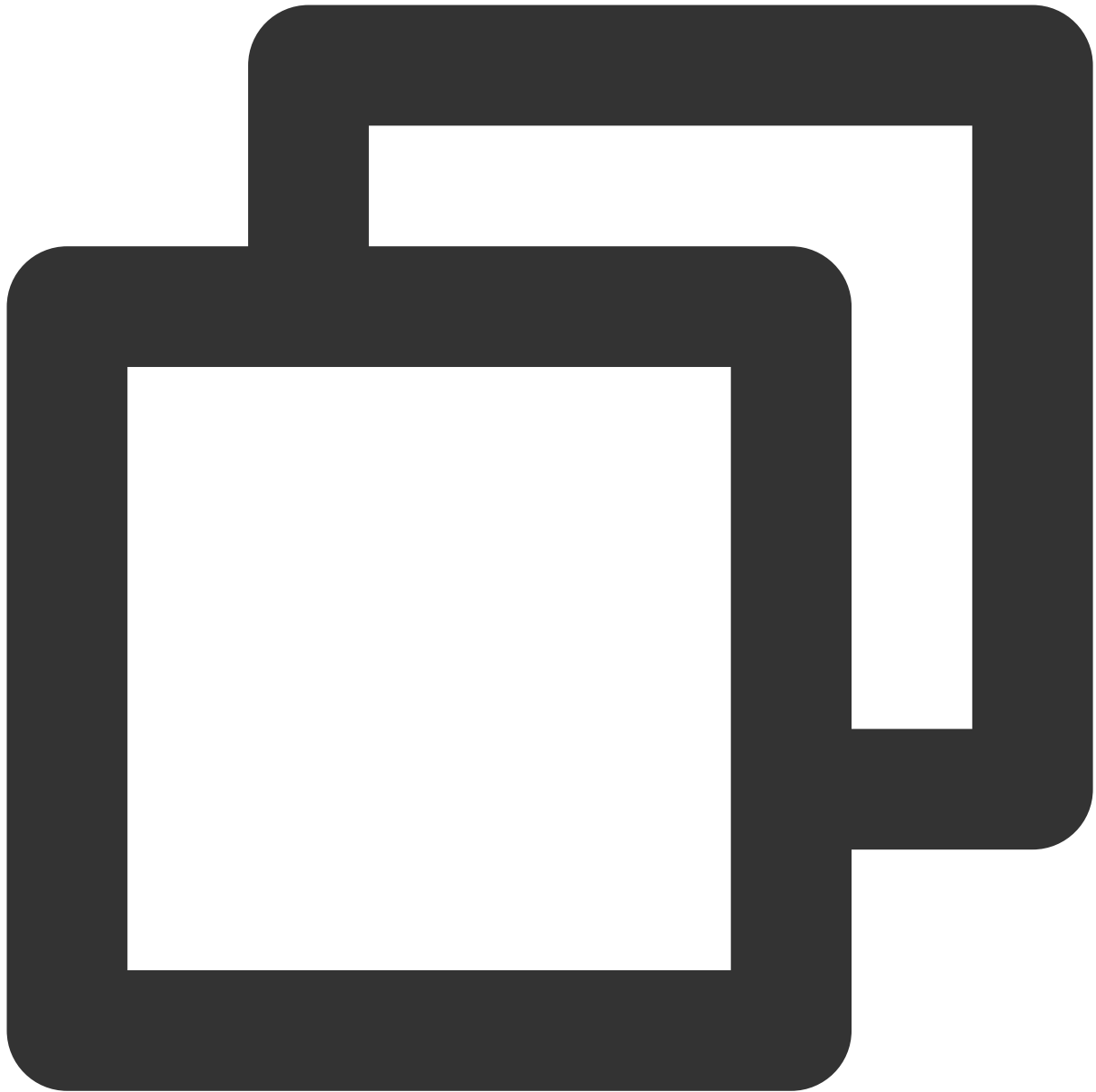
@Autowired
private RocketMQTemplate rocketMQTemplate;
```

2. Send messages. The message body can be a custom object or a message object that is contained in the package `org.springframework.messaging`.



```
SendResult sendResult = rocketMQTemplate.syncSend(destination, message);  
/*-----*/  
rocketMQTemplate.syncSend(destination, MessageBuilder.withPayload(message).build())
```

3. Below is a complete sample.



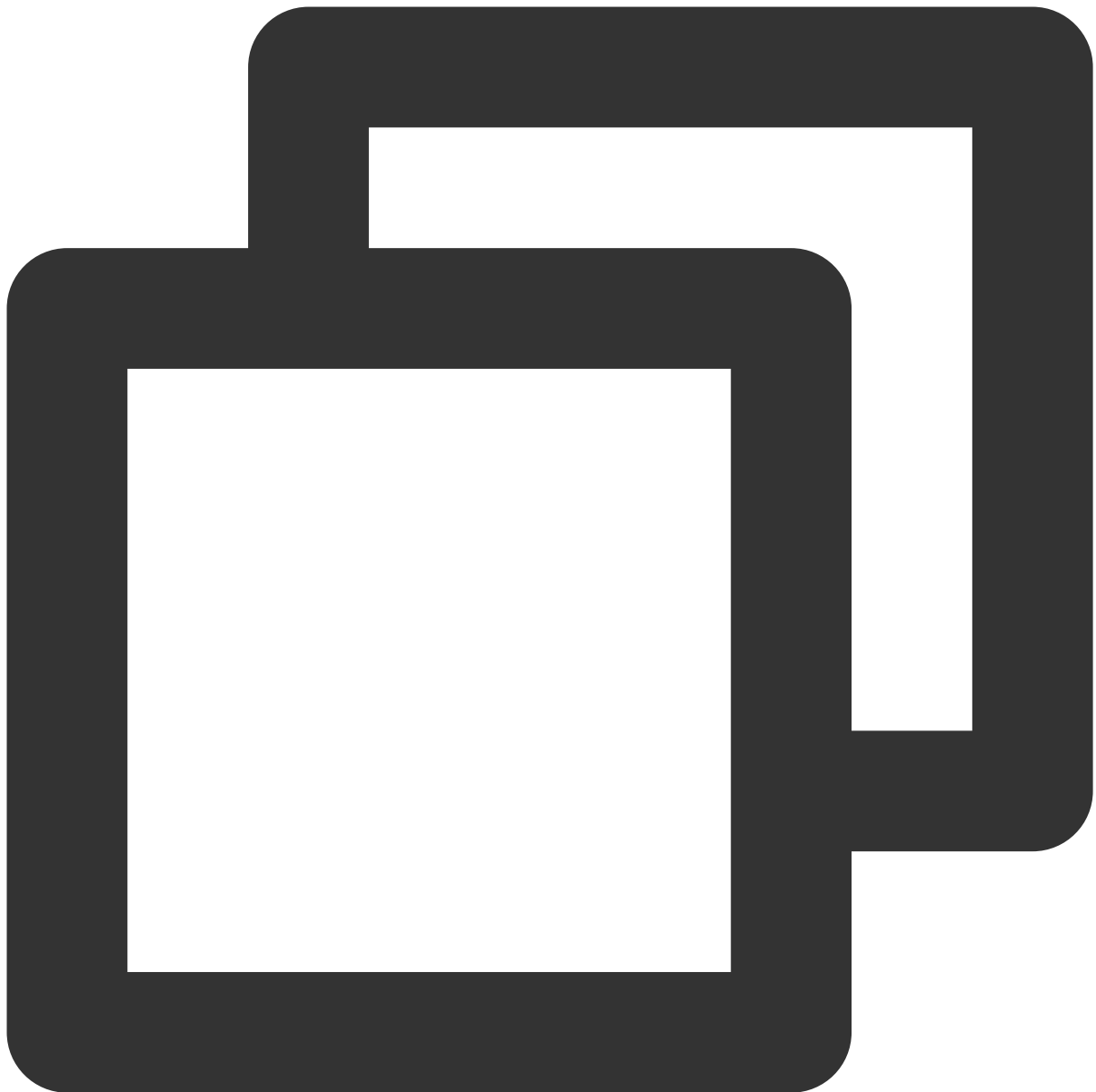
```
/**
 * Description: Message producer
 */
@Service
public class SendMessage {
    // Use the full name of the topic, which can be either customized or concatenated i
    @Value("${rocketmq.namespace}%${rocketmq.producer1.topic}")
    private String topic;
    @Autowired
    private RocketMQTemplate rocketMQTemplate;
}
```

```
* Sync sending
*
* @param message Message content
* @param tags      Subscribed tags
*/
public void syncSend(String message, String tags) {
    // Spring Boot does not support passing tags by using the header. You must
    String destination = StringUtils.isBlank(tags) ? topic : topic + ":" + tag
    SendResult sendResult = rocketMQTemplate.syncSend(destination,
        MessageBuilder.withPayload(message)
            .setHeader(MessageConst.PROPERTY_KEYS, "yo")
            .build());
    System.out.printf("syncSend1 to topic %s sendResult=%s %n", topic, sendRes
}
}
```

### Note

Above is a sync sending sample. For more information on async sending and one-way sending, see the [demo](#) or [TencentCloud/rocketmq-demo](#) in GitHub.

## Step 4. Consume messages



```
@Service
@RocketMQMessageListener(
    consumerGroup = "${rocketmq.namespace}%${rocketmq.consumer1.group}", //
    // Use the full name of the topic, which can be either customized or con
    topic = "${rocketmq.namespace}%${rocketmq.consumer1.topic}",
    selectorExpression = "${rocketmq.consumer1.subExpression}" // Subscripti
)
public class MessageConsumer implements RocketMQListener<String> {

    @Override
    public void onMessage(String message) {
```



```

        System.out.println("Tag1Consumer receive message:" + message);
    }
}

```

You can configure multiple consumers as needed. The consumer configurations depend on your business requirements.

### Note

For a complete sample, download the [demo](#) or ([https://github.com/TencentCloud/rocketmq-demo/rocketmq4/src/main/java/com/tencent/demo/rocketmq4/simple!f2025fba6fb266a8503c27ebf173037b](https://github.com/TencentCloud/rocketmq-demo/tree/main/java/rocketmq-demo/rocketmq4/src/main/java/com/tencent/demo/rocketmq4/simple!f2025fba6fb266a8503c27ebf173037b)) obtain the demo in [TencentCloud/rocketmq-demo](#) in GitHub.

## Step 5. View consumption details

Log in to the [TDMQ console](#), go to the **Cluster > Group** page, and view the list of clients connected to the consumer group. Click **Consumer Details** in the **Operation** column to view consumer details.

The screenshot displays the TDMQ console interface for viewing consumer group details. The top navigation bar includes 'Basic Info', 'Namespace', 'Topic', and 'Group'. The main content area shows a table of consumer groups with columns for Group Name, Consumer Info, Consumption Mode, Description, and Operation. Two groups are listed: 'group-364733' and 'dasda'. Below the table, there are sections for 'Basic Info' and 'Client Address'.

Group Name	Consumer Info	Consumption Mode	Description	Operation
group-364733	Online Consumer 0 TPS 0 Total Heap 0	Unknown		Consumer Details Reset Offset Edit Delete
dasda	Online Consumer 0 TPS 0 Total Heap 0	Unknown		Consumer Details Reset Offset Edit Delete

**Basic Info**

Group Name	group-364733	Creation Time	2022-03-11 15:13:15
Consumption Mode	Unknown	Client Protocol	TCP
Total Heaped Messages	0	Consumer Type	Unknown

**Client Address**

Client Address	Client Language	Client Version	Message Heap	Operation
No data yet				

# Sending and Receiving Filtered Messages

Last updated : 2023-04-12 11:39:41

## Overview

This document describes how to use Spring Boot Starter 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](#).

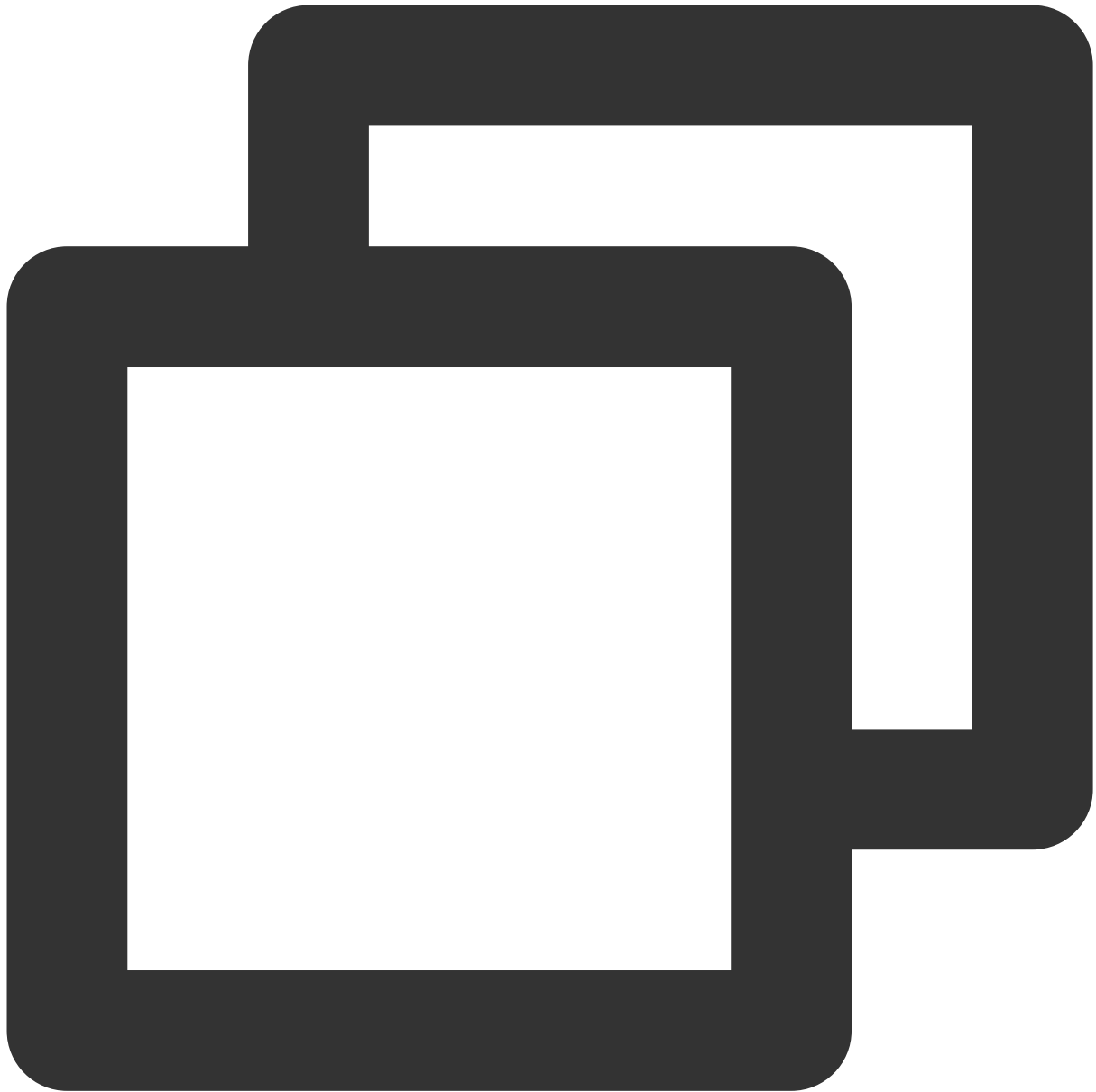
You have learned about the sending and receiving process of general messages.

[You have downloaded the demo here](#) or have downloaded one at the [GitHub project](#).

## Directions

### **Sending a message**

This process is the same as that of general messages, but you need to concatenate the topic sent by `rocketMQTemplate` to corresponding tag.

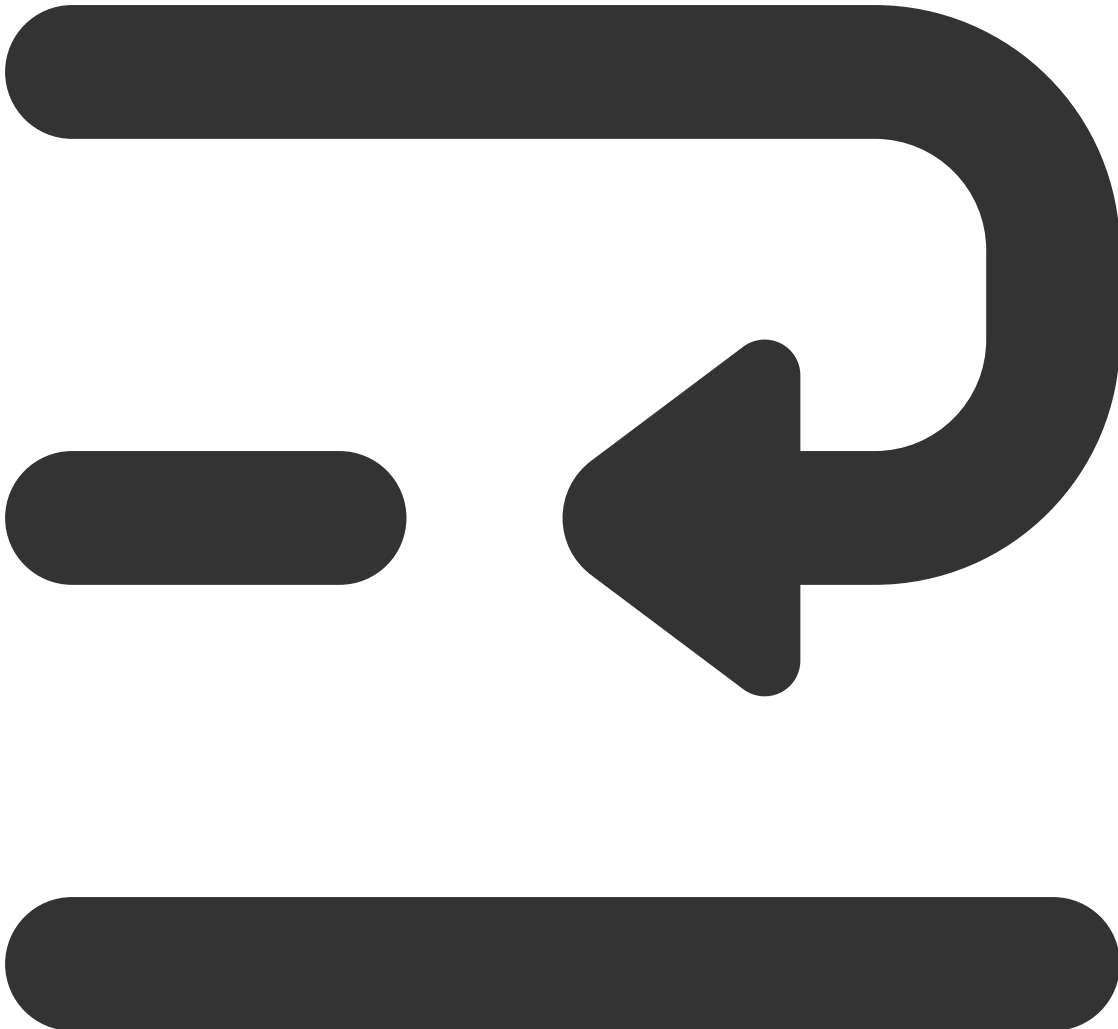


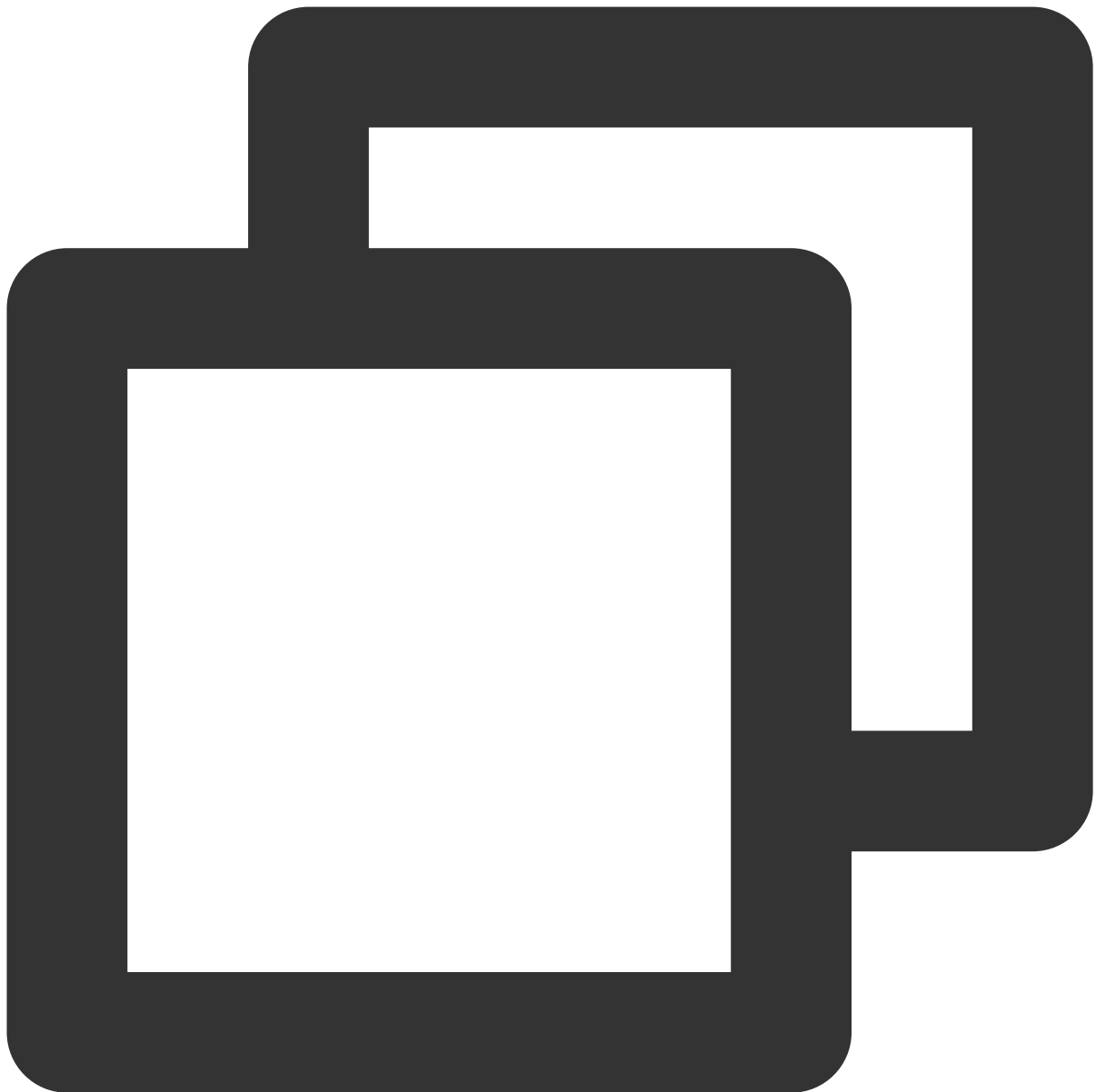
```
// Spring Boot does not support passing tags by using the header. You must concatenate  
String destination = StringUtils.isBlank(tags) ? topic : topic + ":" + tags  
// object message type  
SendResult sendResult = rocketMQTemplate.syncSend(destination,  
    MessageBuilder.withPayload(message)  
        .setHeader(MessageConst.PROPERTY_KEYS, "yourKey") // Specify the key  
        .build());  
System.out.printf("syncSend1 to topic %s sendResult=%s %n", topic, sendResult);
```

For example, topic is `TopicTest` , tag is `TAG1` , then the first parameter to call `rocketMQTemplate` method will be `TopicTest:TAG1`

## Consuming a message

Set the `selectorExpression` field to the corresponding filter tag. In the following code, set `rocketmq.consumer1.subExpression` to `TAG1` to consume the messages of `TAG1` .





```
@Service
@RocketMQMessageListener(
    consumerGroup = "${rocketmq.namespace}%${rocketmq.consumer1.group}", // Co
    // Use the full name of the topic, which can be either customized or concat
    topic = "${rocketmq.namespace}%${rocketmq.consumer1.topic}",
    selectorExpression = "${rocketmq.consumer1.subExpression}" // Subscription
)
public class Tag1Consumer implements RocketMQListener<String> {

    @Override
    public void onMessage(String message) {
```

```
        System.out.println("Tag1Consumer receive message:" + message);  
    }  
}
```

# Sending and Receiving Delayed Messages

Last updated : 2023-04-12 11:41:05

## Overview

This document describes how to use Spring Boot Starter 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 learned about the sending and receiving process of general messages.

[You have downloaded the demo here](#) or have downloaded one at the [GitHub project](#).

## Directions

### Sending a message

This process is the same as that of general messages, but you need to pass in the corresponding delay level when calling the sending method.



```
SendResult sendResult = rocketMQTemplate.syncSend(  
    destination,  
    MessageBuilder.withPayload(message).build(),  
    5000,  
    delayLevel);
```

### The relationship between the delay level and the delay time

The corresponding relationship between other delay levels and specific delay times is as follows:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

1s, 5s, 10s, 30s, 1m, 2m, 3m, 4m, 5m, 6m, 7m, 8m, 9m, 10m, 20m, 30m, 1h, 2h;



## Consuming a message

This process is the same as that of general messages. No other actions are required.

# Spring Cloud Stream

Last updated : 2023-09-12 17:53:17

## Overview

This document describes how to use Spring Cloud Stream 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](#).

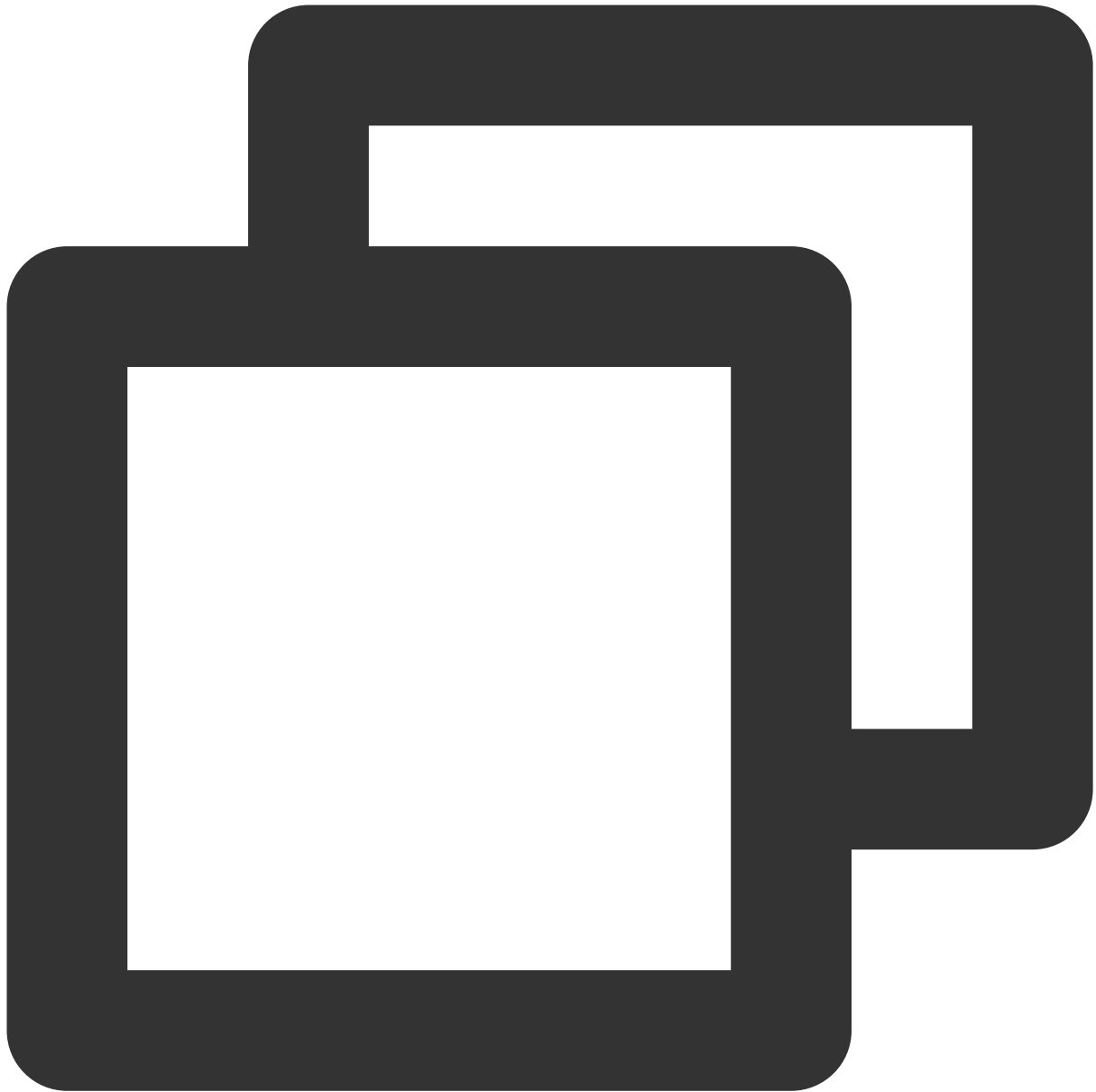
[You have installed Maven 2.5 or later](#).

[You have downloaded the demo here](#) or have downloaded one at the [GitHub project](#).

## Directions

### Step 1. Import dependencies

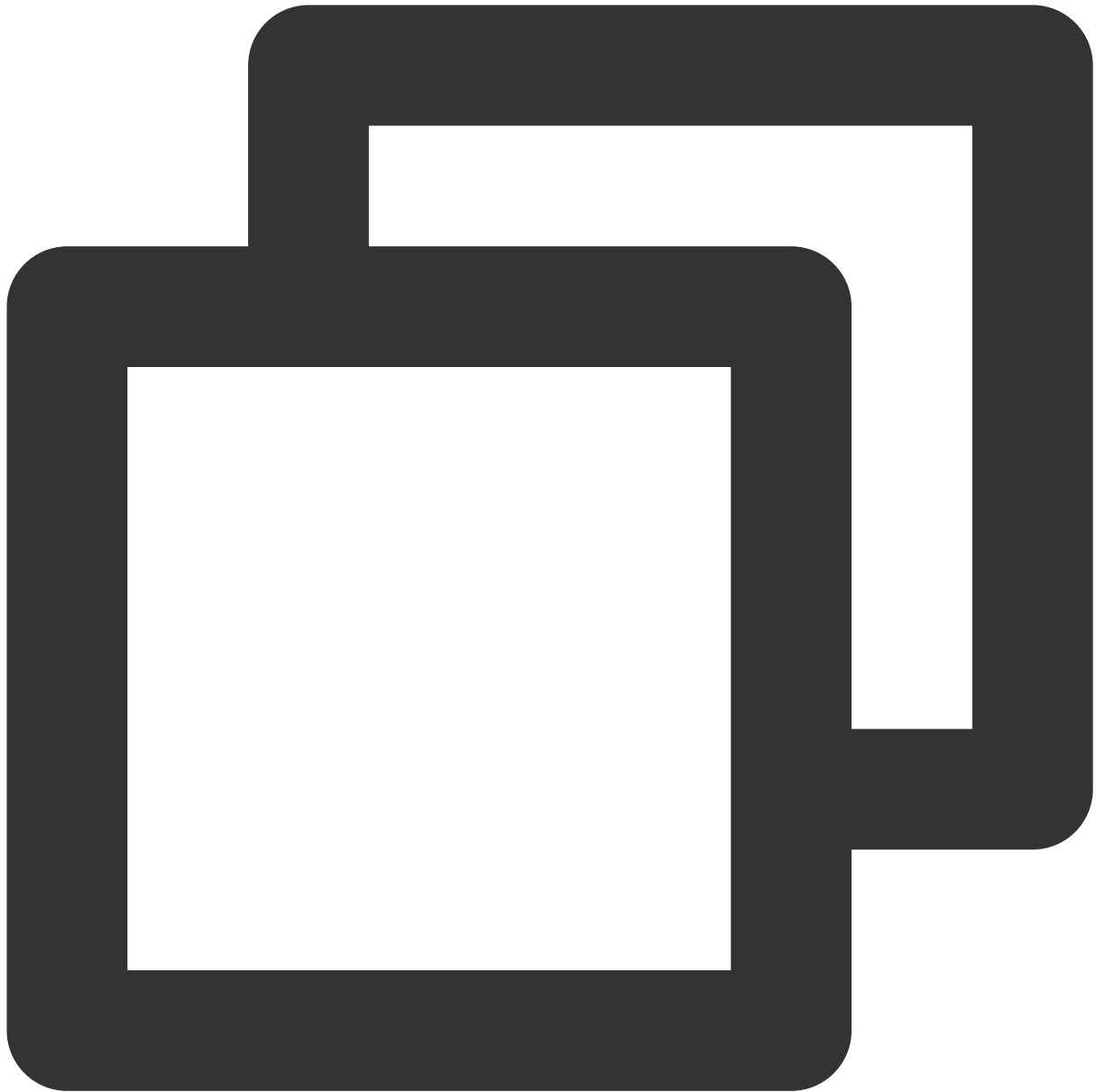
Import `spring-cloud-starter-stream-rocketmq`-related dependencies in pom.xml. It is recommended to use v2021.0.4.0.



```
<dependency>
  <groupId>com.alibaba.cloud</groupId>
  <artifactId>spring-cloud-starter-stream-rocketmq</artifactId>
  <version>2021.0.4.0</version>
</dependency>
```

## Step 2. Add configurations

Add RocketMQ-related configurations to the configuration file.



```
spring:
  cloud:
    stream:
      rocketmq:
        binder:
          # Full service address
          name-server: rocketmq-xxx.rocketmq.ap-bj.public.tencenttdmq.com:9876
          # Role name
          secret-key: admin
          # Role token
          access-key: eyJrZXlJZ...
```

```

# Full namespace name
namespace: rocketmq-xxx|namespace1
# producer group
group: producerGroup
bindings:
# Channel name, which is the same as the channel name in spring.cloud.str
Topic-TAG1-Input:
  consumer:
    # Tag type of the subscription, which is configured based on consumer
    subscription: TAG1
# Channel name
Topic-TAG2-Input:
  consumer:
    subscription: TAG2
bindings:
# Channel name
Topic-send-Output:
# Specify a topic, which refers to the one you created
destination: TopicTest
content-type: application/json
# Channel name
Topic-TAG1-Input:
  destination: TopicTest
  content-type: application/json
  group: consumer-group1
# Channel name
Topic-TAG2-Input:
  destination: TopicTest
  content-type: application/json
  group: consumer-group2

```

**Note**

1. Currently, only `2.2.5-RocketMQ-RC1` and `2.2.5.RocketMQ.RC2` or later versions support **namespace** configuration. If you use other versions, you need to concatenate topic and group names.

The format is as follows:

`rocketmq-pngrpmk94d5o|stream%topic` (format: namespace name %topic name)

`rocketmq-pngrpmk94d5o|stream%group` (format: namespace name%group name)

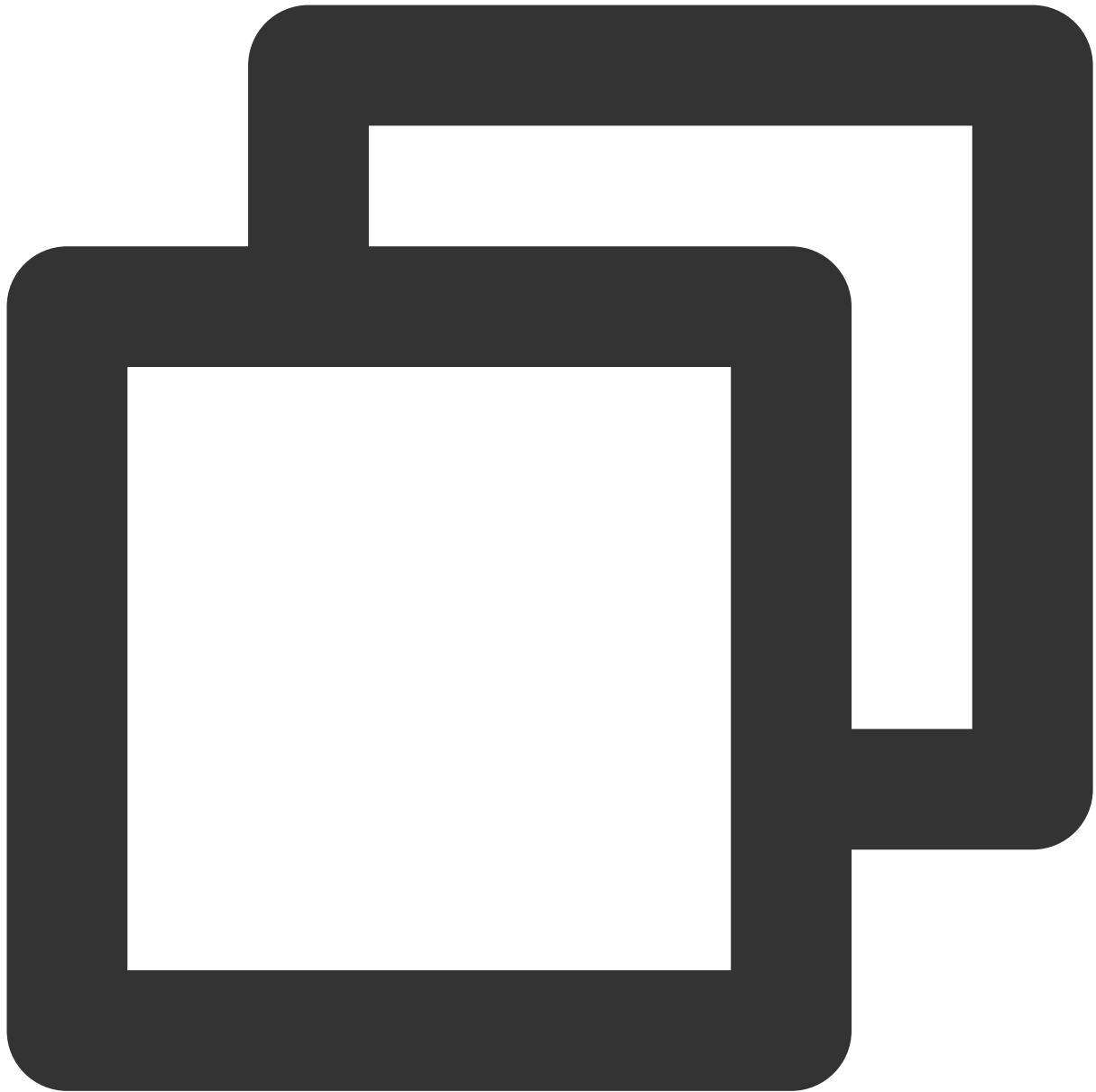
The format for Shared and Exclusive editions is as follows:

`MQ_INST_rocketmqj79obd2ew7v_test%topic` (format: namespace name%topic name)

`MQ_INST_rocketmqj79obd2ew7v_test%group` (format: namespace name%group name)

2. The subscription configuration item is `subscription` for `2.2.5-RocketMQ-RC1` and `2.2.5.RocketMQ.RC2` and is `tags` for other earlier versions.

The complete configuration items of other versions are as follows:

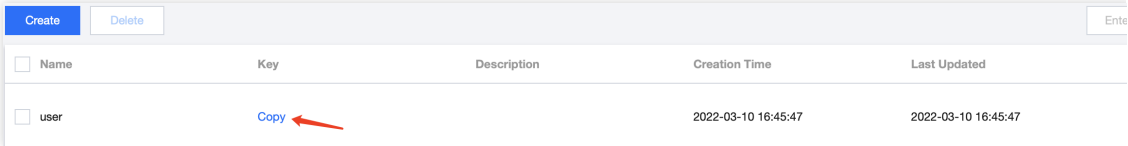


```
spring:
  cloud:
    stream:
      rocketmq:
        bindings:
          # Channel name, which is the same as the channel name in spring.cloud.
          Topic-test1:
            consumer:
              # Tag type of the subscription, which is configured based on consu
              tags: TAG1
          # Channel name
```

```

Topic-test2:
  consumer:
    tags: TAG2
binder:
  # Full service address
name-server: rocketmq-xxx.rocketmq.ap-bj.public.tencenttdmq.com:9876
  # Role name
secret-key: admin
  # Role token
access-key: eyJrZXlJZ...
bindings:
  # Channel name
Topic-send:
  # Specify a topic in the format of `cluster ID|namespace name%topic na
destination: rocketmq-xxx|stream%topic1
content-type: application/json
  # Name of the group to be used in the format of `cluster ID|namespace
group: rocketmq-xxx|stream%group1
  # Channel name
Topic-test1:
  destination: rocketmq-xxx|stream%topic1
  content-type: application/json
  group: rocketmq-xxx|stream%group1
  # Channel name
Topic-test2:
  destination: rocketmq-xxx|stream%topic1
  content-type: application/json
  group: rocketmq-xxx|stream%group2

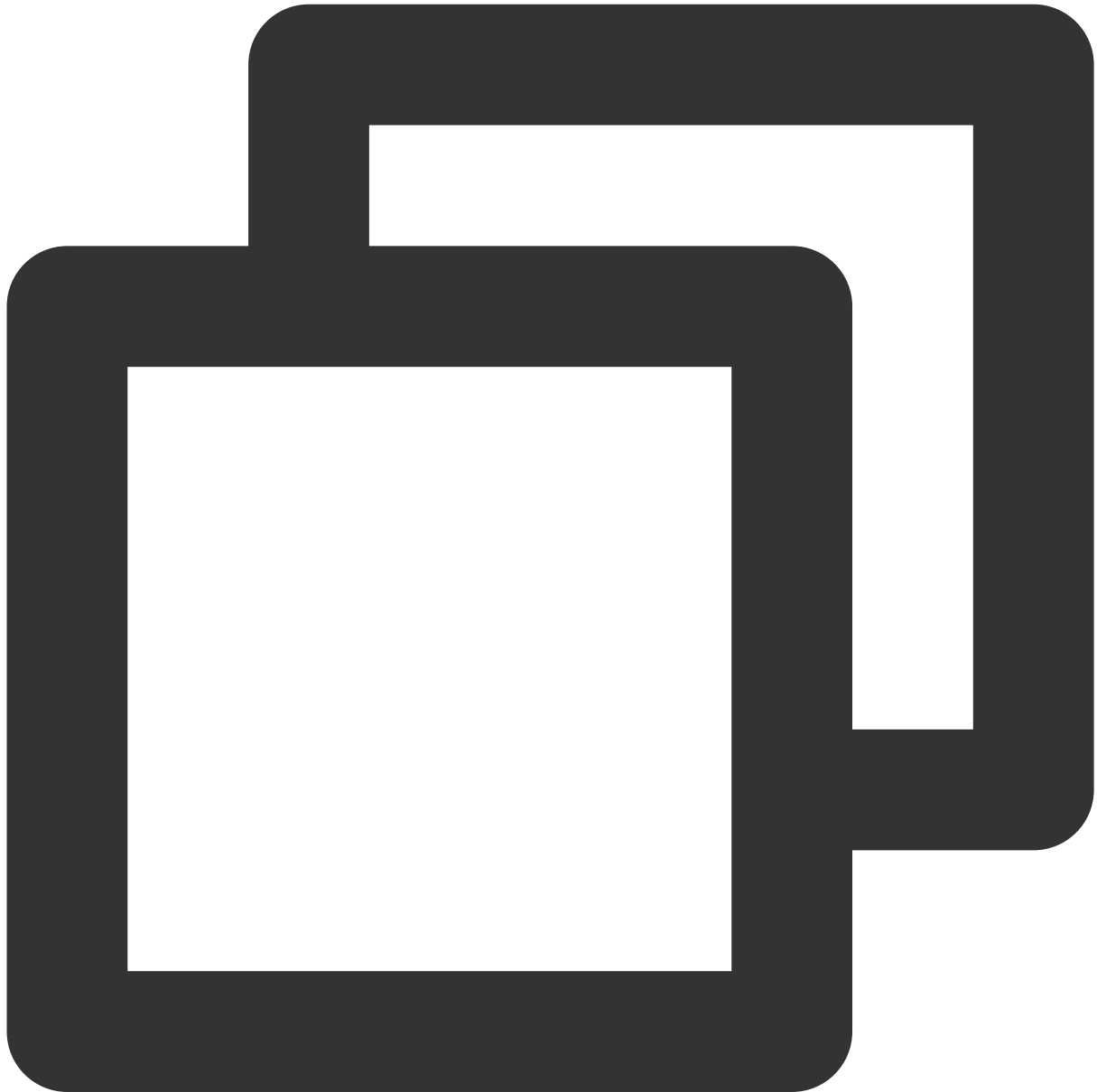
```

Parameter	Description
name-server	Cluster access address, which can be copied from <b>Access Address</b> in the <b>Operation</b> column on the console. Namespace access addresses in new virtual or exclusive clusters can be copied from the
secret-key	Role name, which can be copied on the <a href="#">Role Management</a> page.
access-key	Role token, which can be copied in the <b>Token</b> column on the <a href="#">Role Management</a> page. 
namespace	Namespace name, which can be copied on the <b>Namespace</b> page in the console.
group	Producer group name, which can be copied under the <b>Group</b> tab on the cluster details page.

destination	Topic name, which can be copied on the <b>Topic</b> page in the console.
-------------	--------------------------------------------------------------------------

### Step 3. Configure channels

You can separately configure input and output channels as needed.



```
/**
 * Custom channel binder
 */
public interface CustomChannelBinder {
```



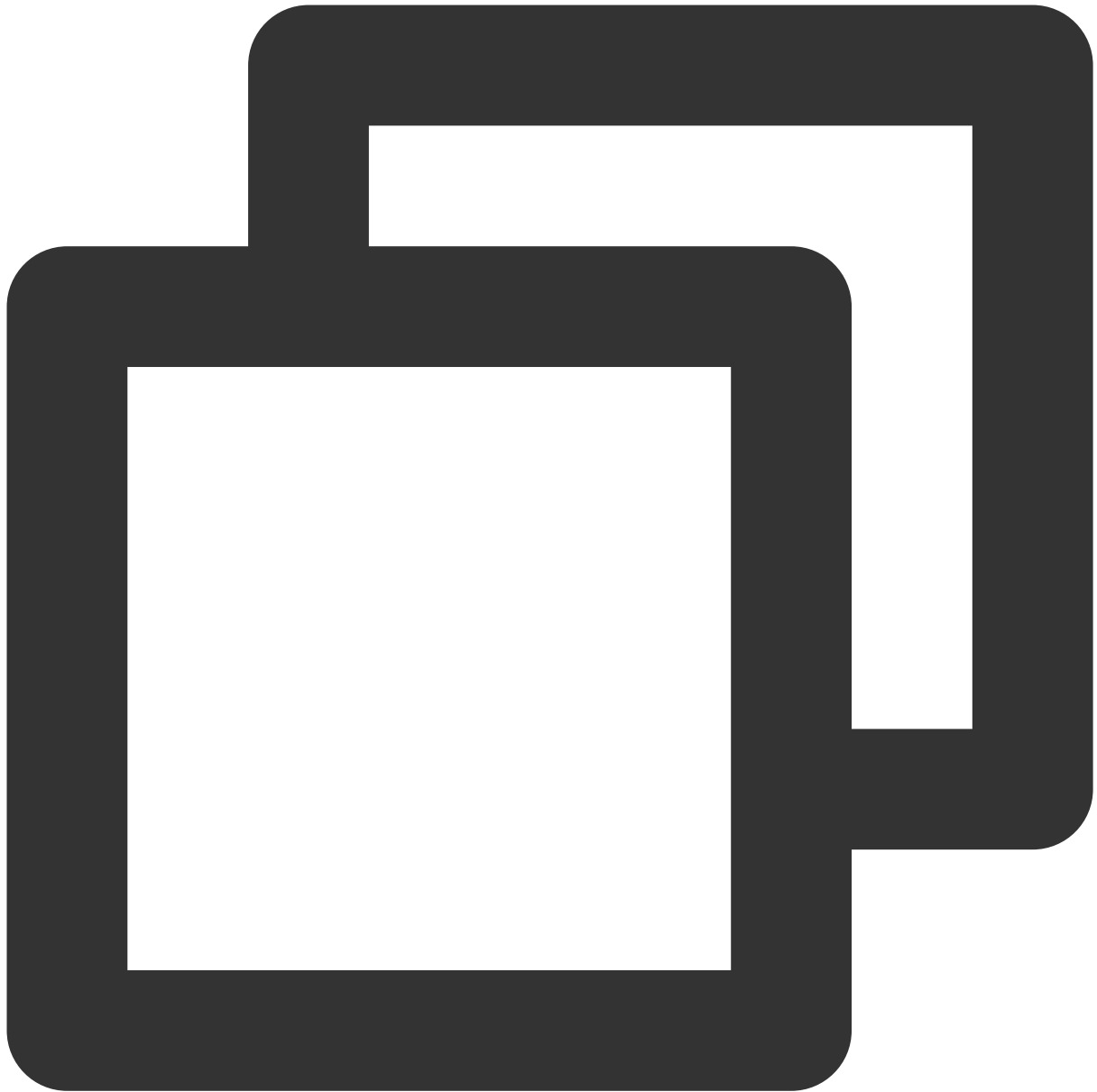
```
/**
 * (Message producers) send messages
 * Bind the channel name in the configurations
 */
@Output("Topic-send-Output")
MessageChannel sendChannel();

/**
 * (Consumer 1) receives message 1
 * Bind the channel name in the configurations
 */
@Input("Topic-TAG1-Input")
MessageChannel testInputChannel1();

/**
 * (Consumer 2) receives message 2
 * Bind the channel name in the configurations
 */
@Input("Topic-TAG2-Input")
MessageChannel testInputChannel2();
}
```

#### Step 4. Add annotations

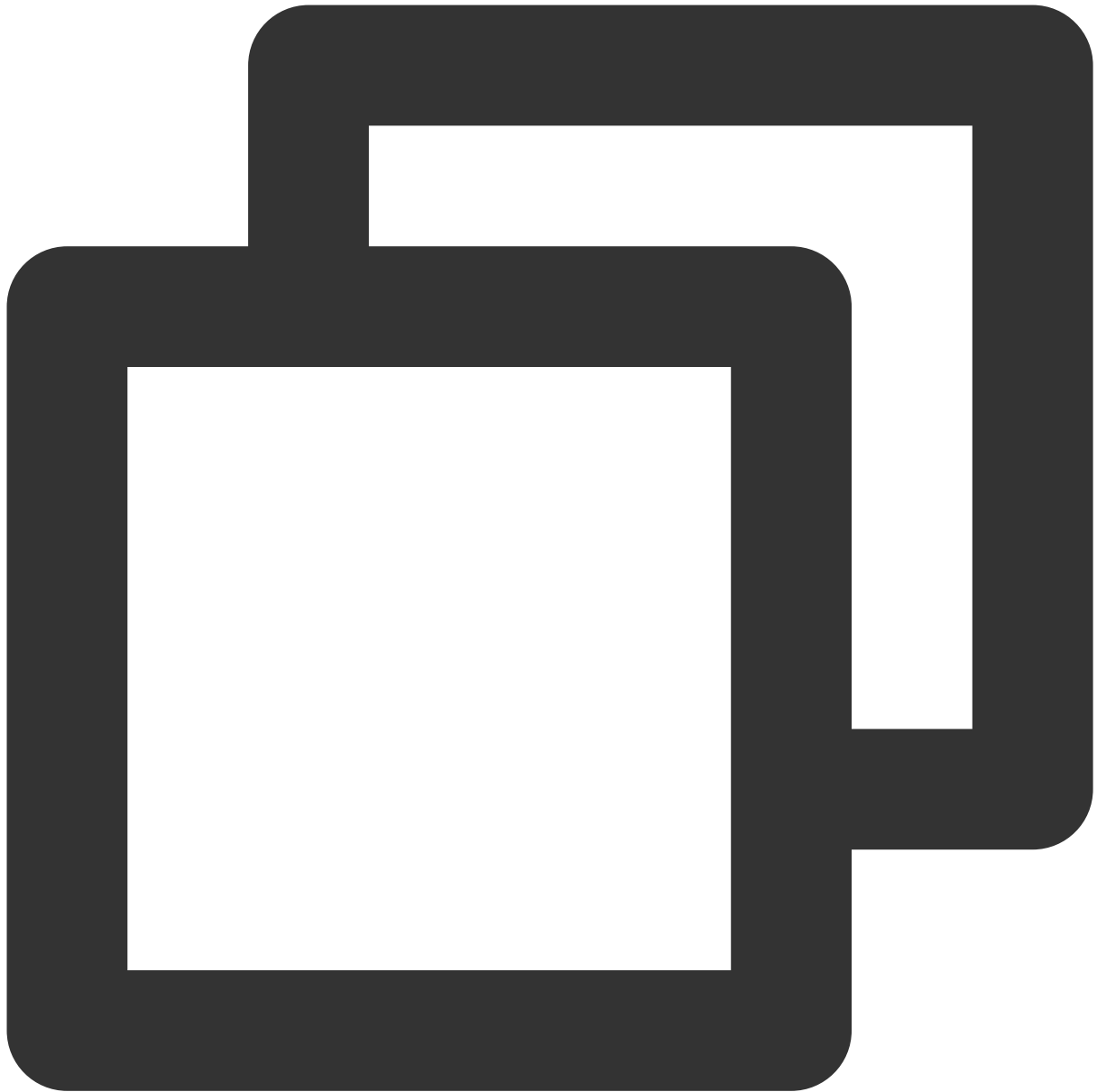
Add annotations to the configuration class or startup class. If multiple binders are configured, specify them in the annotations.



```
@EnableBinding({CustomChannelBinder.class})
```

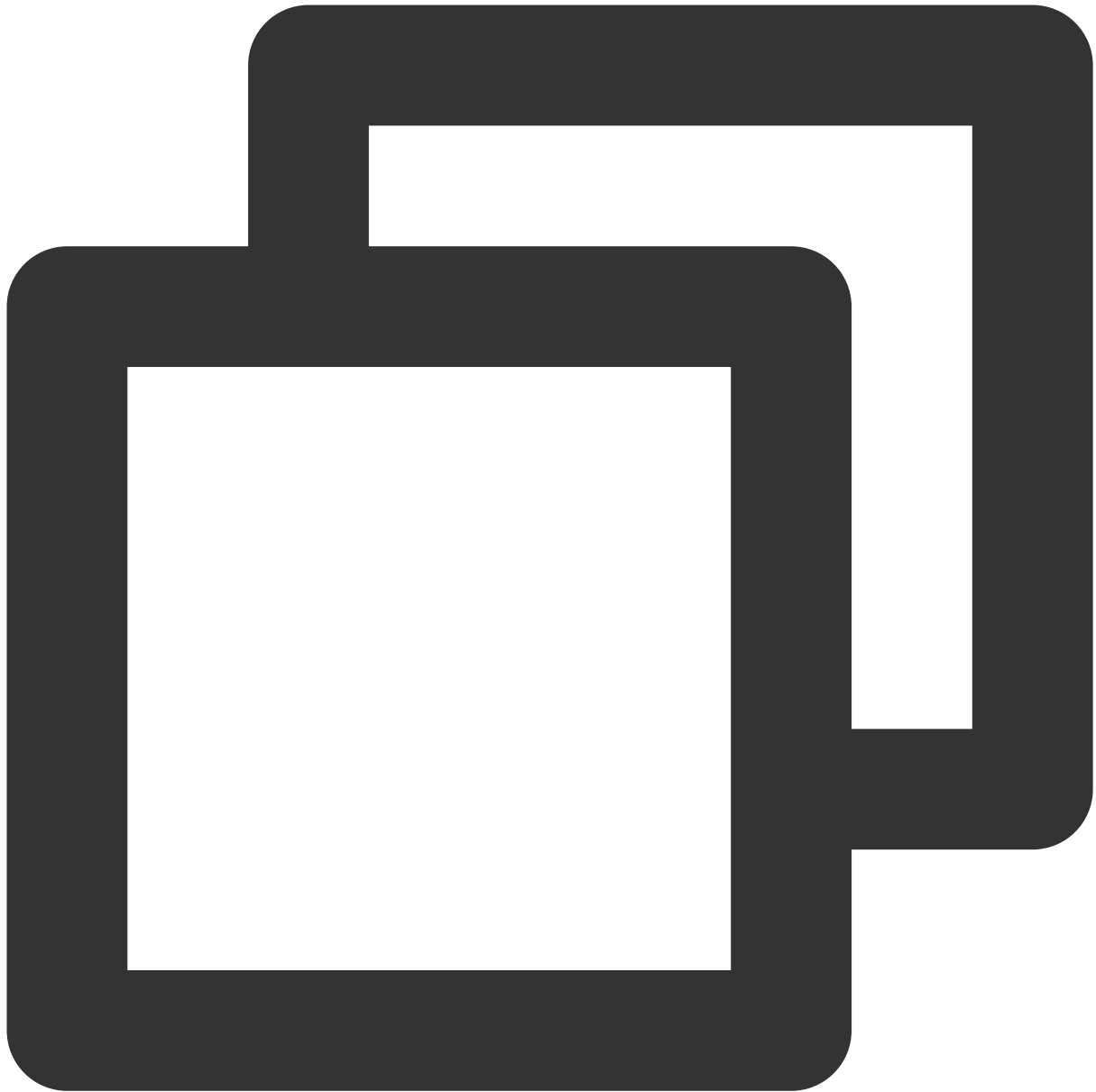
## Step 5. Send messages

1. Inject `CustomChannelBinder` into the class that needs to send messages.



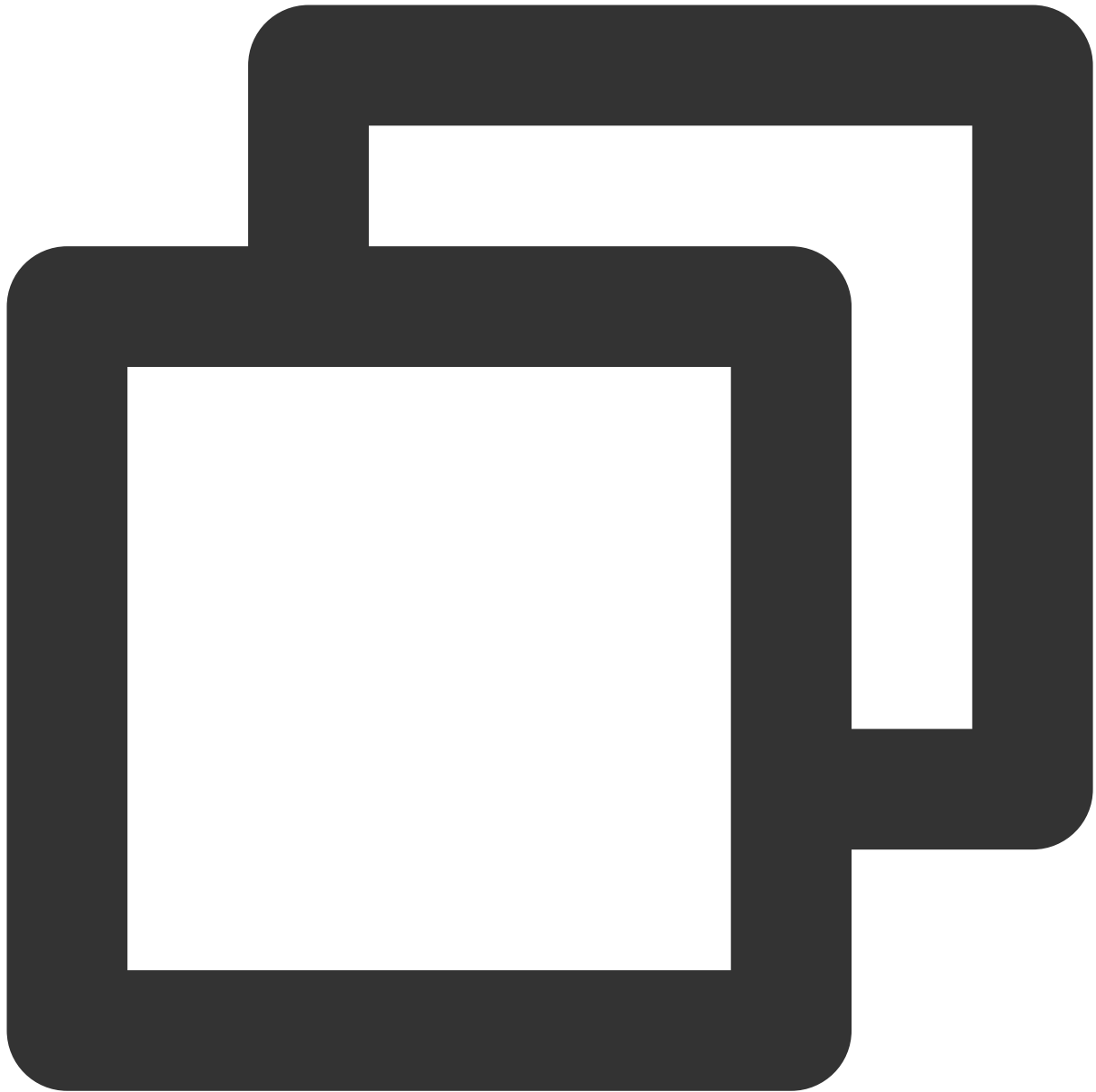
```
@Autowired  
private CustomChannelBinder channelBinder;
```

2. Use the corresponding output stream channel to send messages.



```
Message<String> message = MessageBuilder.withPayload("This is a new message.").build();
channelBinder.sendChannel().send(message);
```

## Step 6. Consume messages



```
@Service
public class StreamConsumer {
    private final Logger logger = LoggerFactory.getLogger(StreamDemoApplication.class);

    /**
     * Listen on the channel configured in the configurations
     *
     * @param messageBody message content
     */
    @StreamListener("Topic-TAG1-Input")
    public void receive(String messageBody) {
```

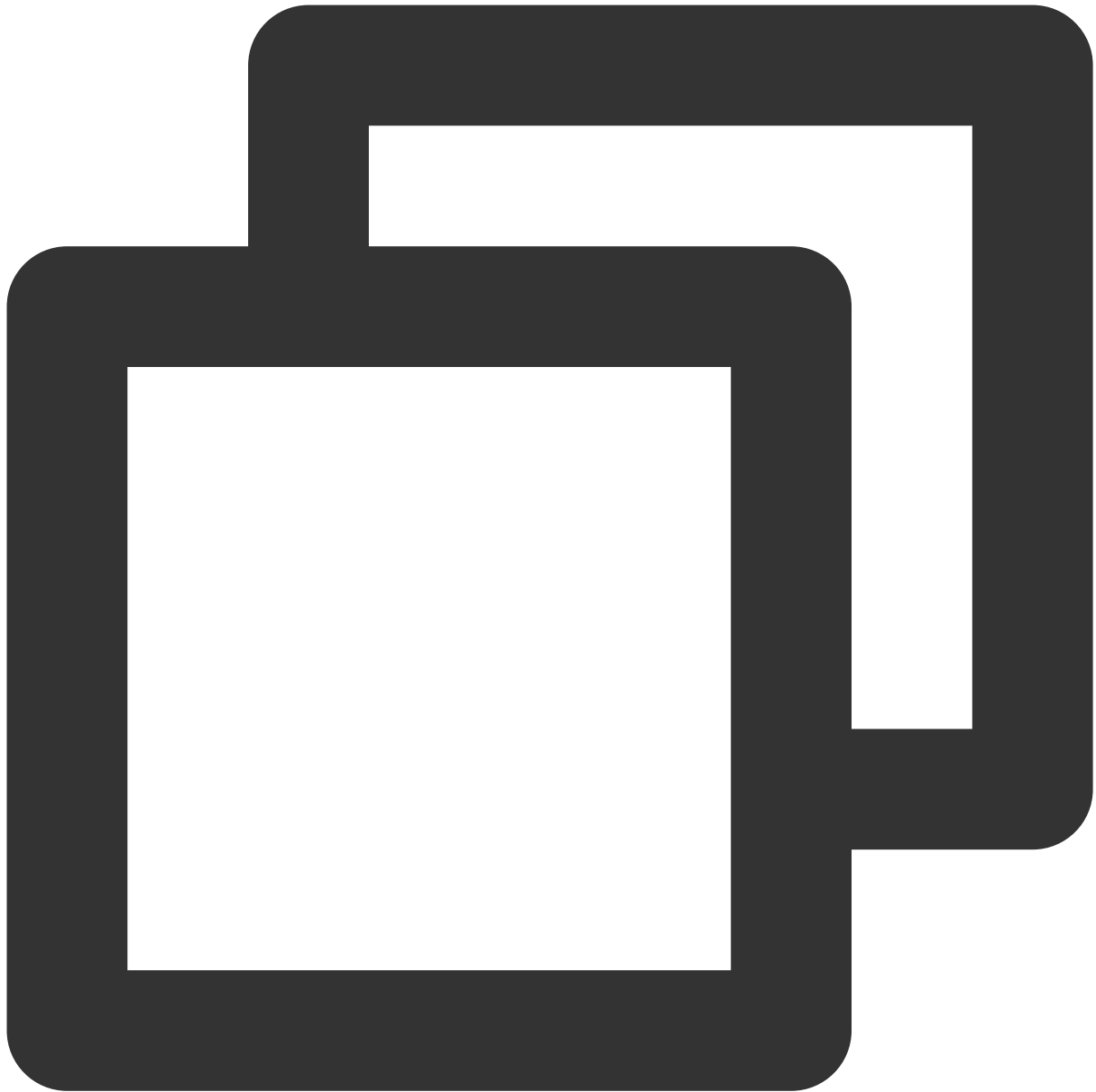
```
        logger.info("Receive1: Messages are received through the stream. messageBod
    }

    /**
     * Listen on the channel configured in the configurations
     *
     * @param messageBody message content
     */
    @StreamListener("Topic-TAG2-Input")
    public void receive2(String messageBody) {
        logger.info("Receive2: Messages are received through the stream. messageBod
    }
}
```

## Step 7: Perform local testing

After starting the project locally, you can see from the console that the startup was successful.

You can see that the sending is successful by checking <http://localhost:8080/test-simple> in the browser. Watch the output log of the development IDE.



```
2023-02-23 19:19:00.441 INFO 21958 --- [nio-8080-exec-1] c.t.d.s.controller.Stream
2023-02-23 19:19:01.138 INFO 21958 --- [nsumer-group1_1] c.t.d.s.StreamDemoApplica
```

You can see that a message of TAG1 is sent, and only the subscribers of TAG1 receive the message.

**Note**

For more information, see [GitHub Demo](#) or [Spring cloud stream official documentation](#).

# SDK for Java

## Sending and Receiving General Messages

Last updated : 2023-10-30 10:38:25

### Overview

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

### Prerequisites

You have created or prepared 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](#) or obtained the demo in [TencentCloud/rocketmq-demo](#) in GitHub.

### Directions

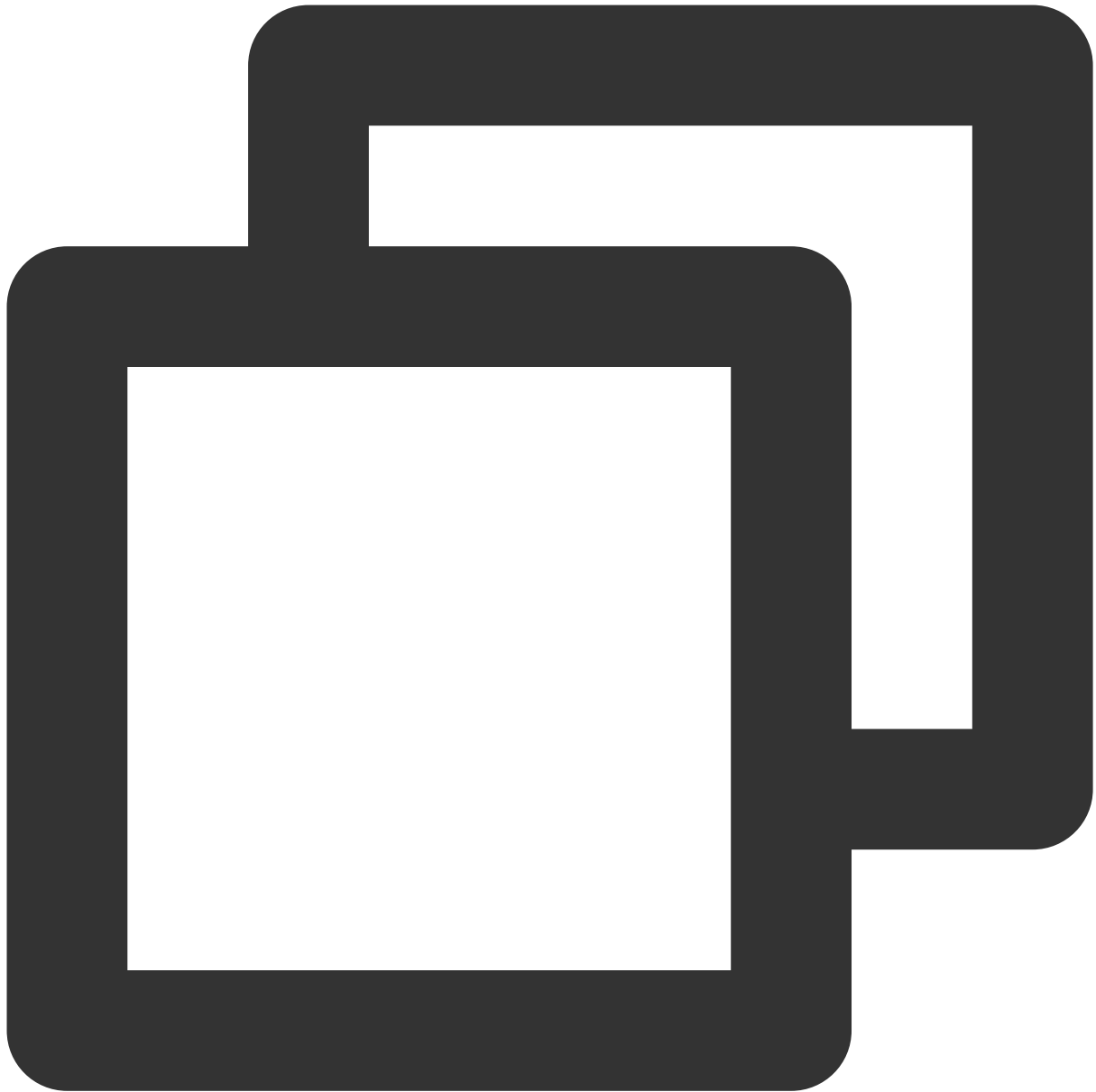
#### Step 1. Install the Java dependent library

Introduce dependencies in a Java project and add the following dependencies to the `pom.xml` file. This document uses a Maven project as an example.

##### Note

The dependency version must be v4.9.3 or later, preferably v4.9.4.





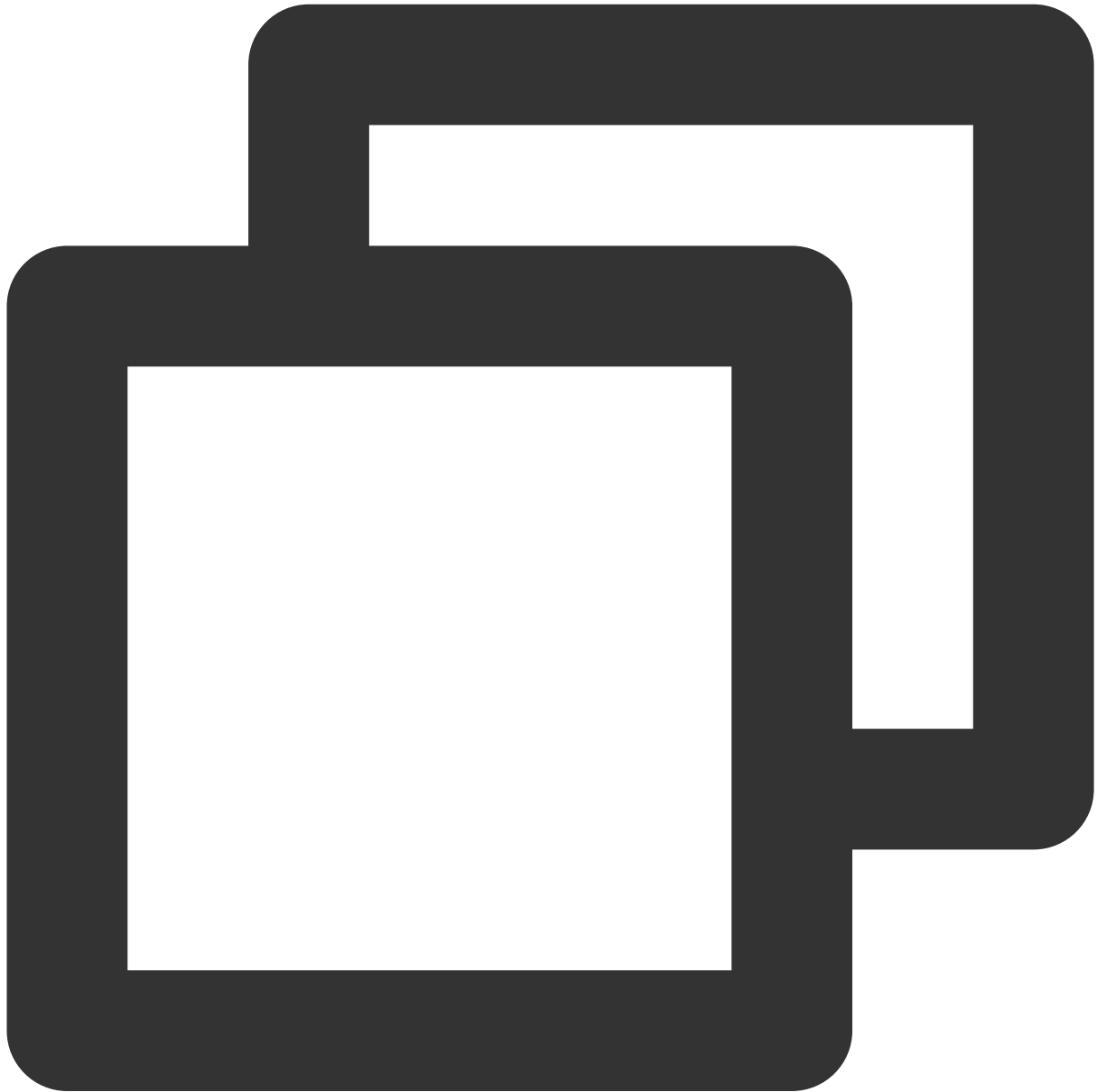
```
<!-- in your <dependencies> block -->
<dependency>
  <groupId>org.apache.rocketmq</groupId>
  <artifactId>rocketmq-client</artifactId>
  <version>4.9.4</version>
</dependency>

<dependency>
  <groupId>org.apache.rocketmq</groupId>
  <artifactId>rocketmq-acl</artifactId>
  <version>4.9.4</version>
```

```
</dependency>
```

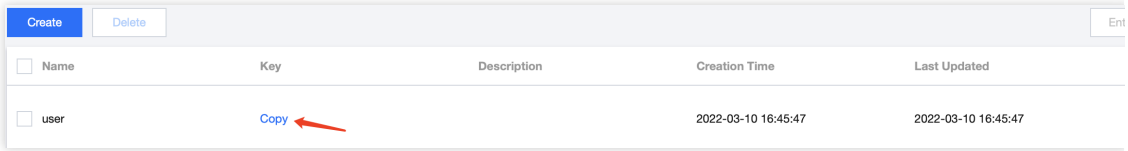
## Step 2. Produce messages

### Creating a message producer



```
// Instantiate the message producer
DefaultMQProducer producer = new DefaultMQProducer(
    groupName,
    new AclClientRPCHook(new SessionCredentials(accessKey, secretKey)) // ACL pe
);
```

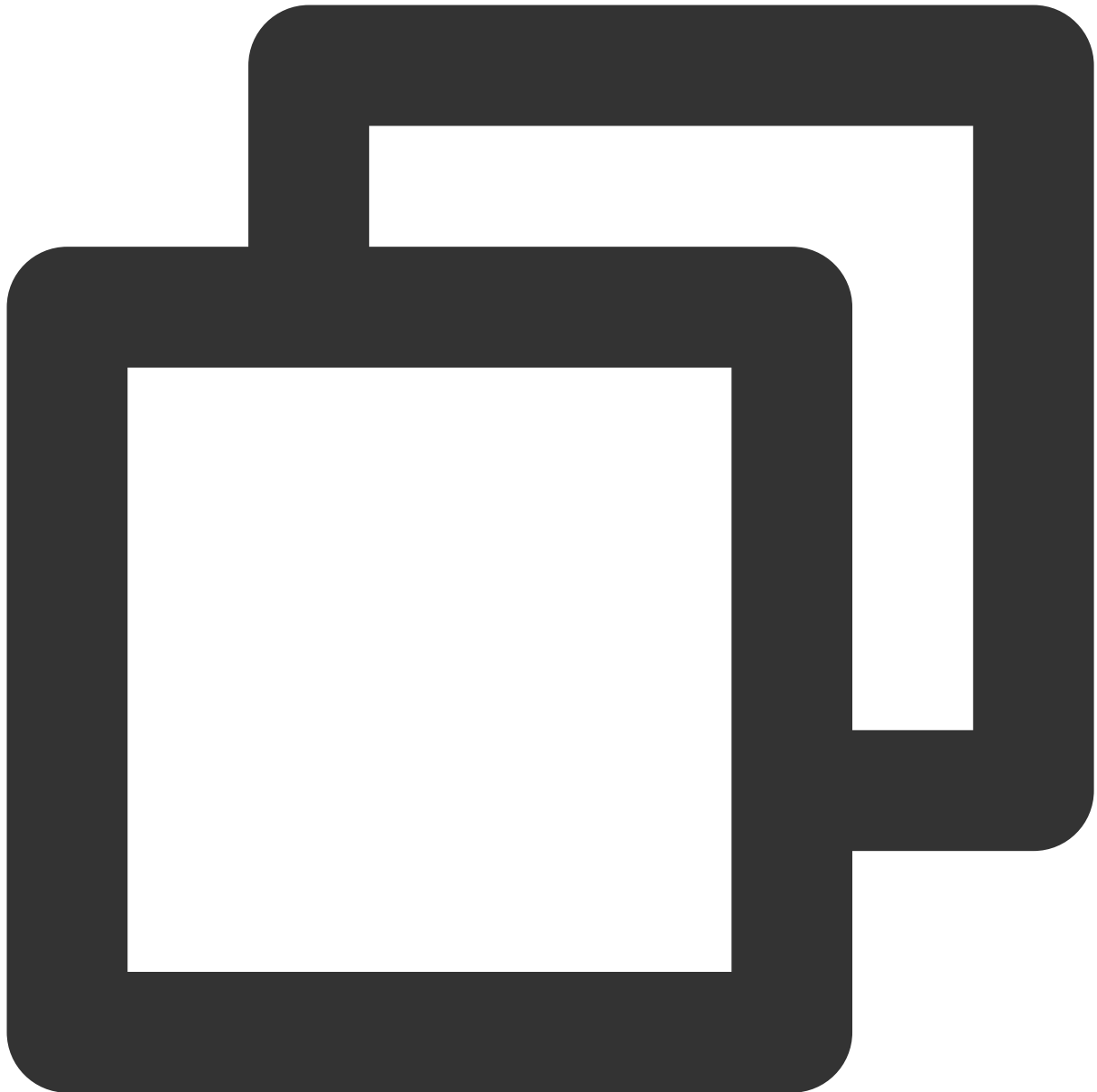
```
// Set the Nameserver address
producer.setNamesrvAddr(nameserver);
// Start the producer instance
producer.start();
```

Parameter	Description
groupName	Producer group name. We recommend that you use the corresponding topic name as the producer g
accessKey	Role token, which can be copied in the <b>Token</b> column on the <a href="#">Role Management</a> page. 
secretKey	Role name, which can be copied on the <a href="#">Role Management</a> page.
nameserver	Cluster access address, which can be obtained from <b>Access Address</b> in the <b>Operation</b> column or the console. The namespace access address can be obtained under the <b>Namespace</b> tab on the <b>CI</b>

## Sending messages

Messages can be sent in the sync, async, or one-way mode.

### Sync sending

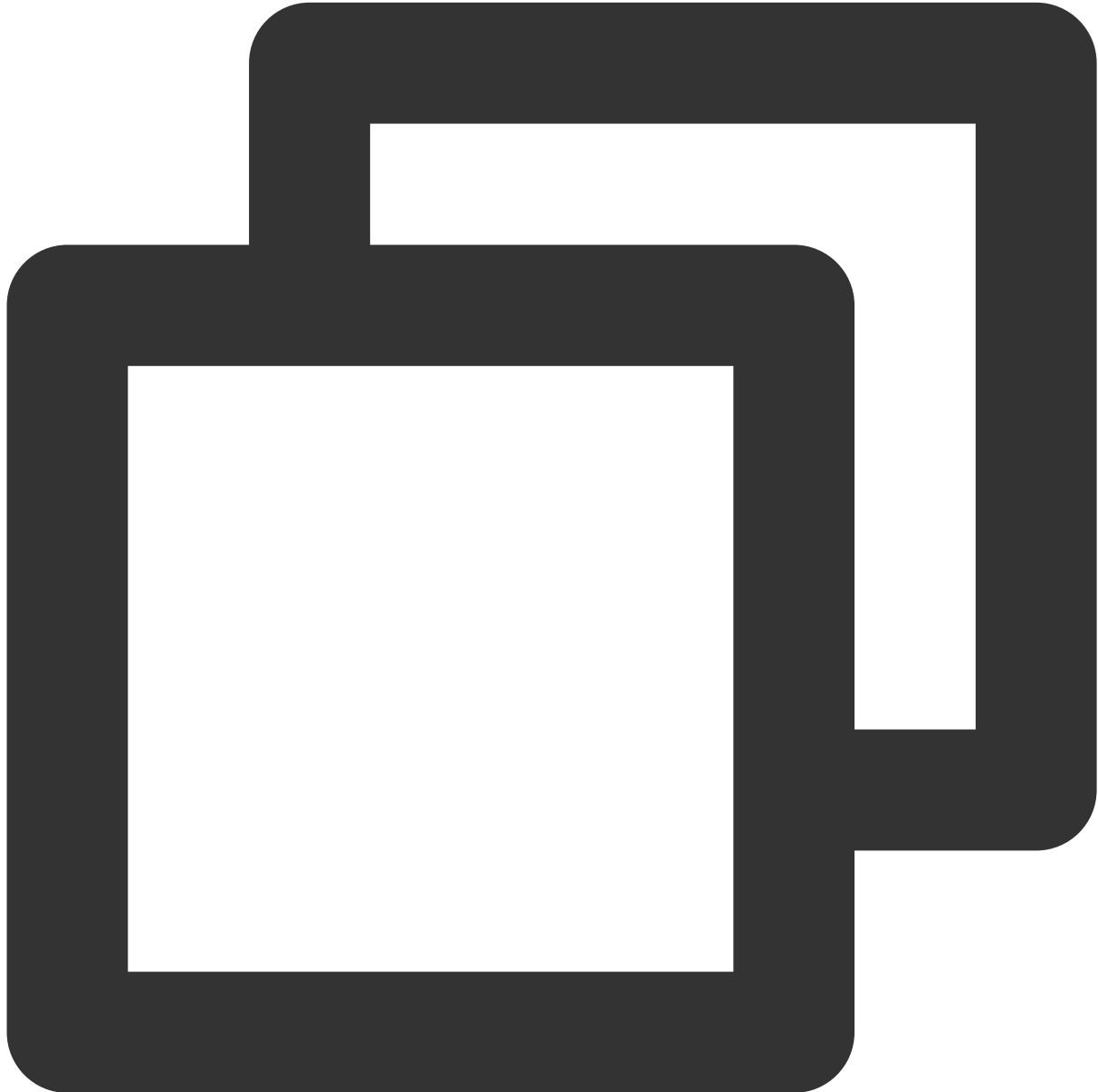


```
for (int i = 0; i < 10; i++) {  
    // Create a message instance and set the topic and message content  
    Message msg = new Message(topic_name, "TAG", ("Hello RocketMQ " + i).getBytes());  
    // Send the message  
    SendResult sendResult = producer.send(msg);  
    System.out.printf("%s\n", sendResult);  
}
```

Parameter	Description

topic_name	Topic name, which can be copied under the <b>Topic</b> tab on the <b>Cluster</b> page in the console.
TAG	A parameter used to set the message tag.

### Async sending



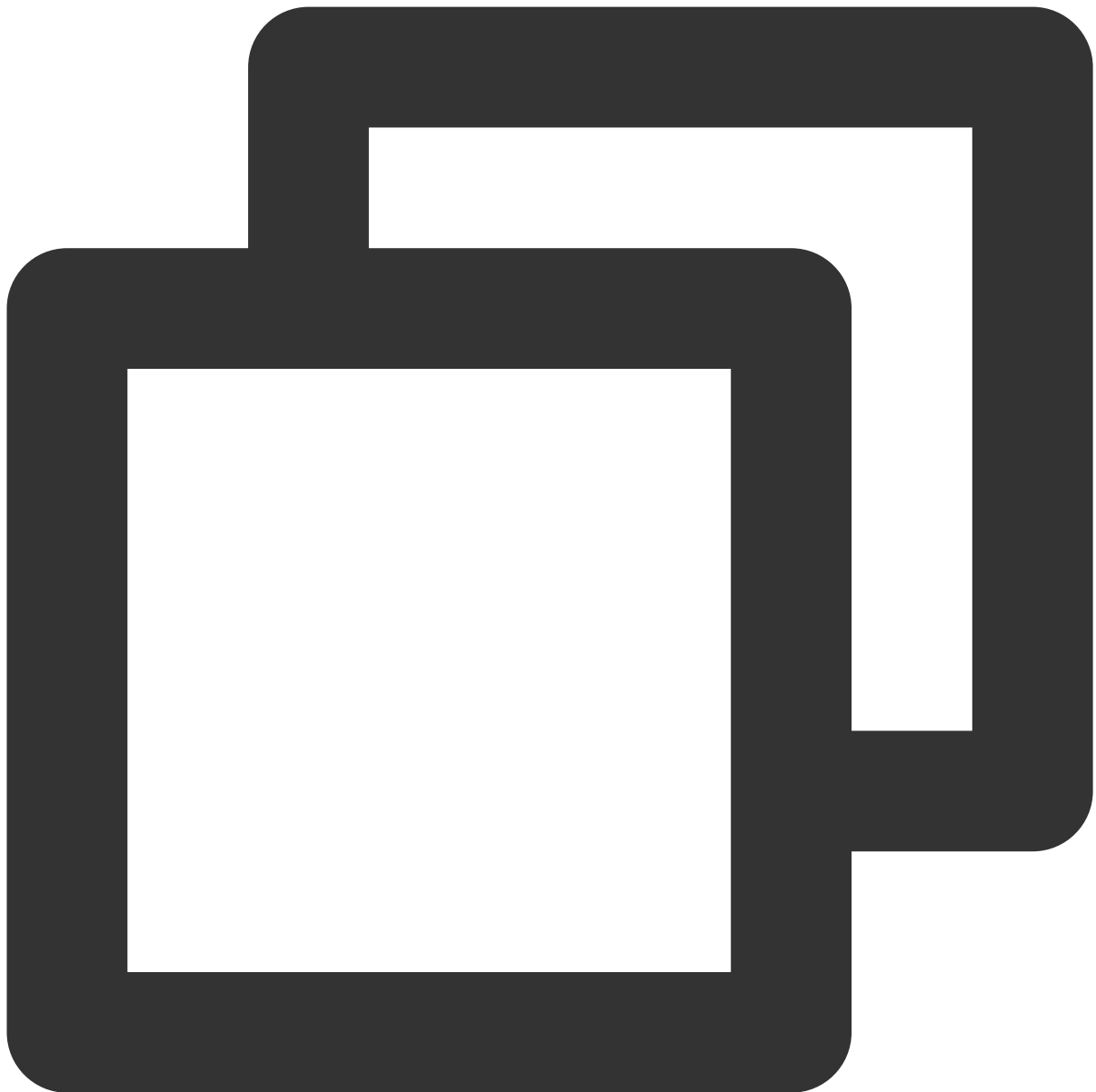
```
// Disable retry upon sending failures
producer.setRetryTimesWhenSendAsyncFailed(0);
// Set the number of messages to be sent
int messageCount = 10;
final CountdownLatch countdownLatch = new CountdownLatch(messageCount);
```

```
for (int i = 0; i < messageCount; i++) {
    try {
        final int index = i;
        // Create a message instance and set the topic and message content
        Message msg = new Message(topic_name, "TAG", ("Hello rocketMq " + ind
        producer.send(msg, new SendCallback() {
            @Override
            public void onSuccess(SendResult sendResult) {
                // Logic for message sending successes
                countdownLatch.countDown();
                System.out.printf("%-10d OK %s %n", index, sendResult.getMsgI
            }

            @Override
            public void onException(Throwable e) {
                // Logic for message sending failures
                countdownLatch.countDown();
                System.out.printf("%-10d Exception %s %n", index, e);
                e.printStackTrace();
            }
        });
    } catch (Exception e){
        e.printStackTrace();
    }
}
countdownLatch.await(5, TimeUnit.SECONDS);
```

Parameter	Description
topic_name	Topic name, which can be copied under the <b>Topic</b> tab on the <b>Cluster</b> page in the console.
TAG	A parameter used to set the message tag.

### One-way sending



```
for (int i = 0; i < 10; i++) {  
    // Create a message instance and set the topic and message content  
    Message msg = new Message(topic_name, "TAG", ("Hello RocketMQ " + i).getBytes());  
    // Send one-way messages  
    producer.sendOneway(msg);  
}
```

Parameter	Description
topic_name	Topic name, which can be copied under the <b>Topic</b> tab on the <b>Cluster</b> page in the console.

TAG

A parameter used to set the message tag.

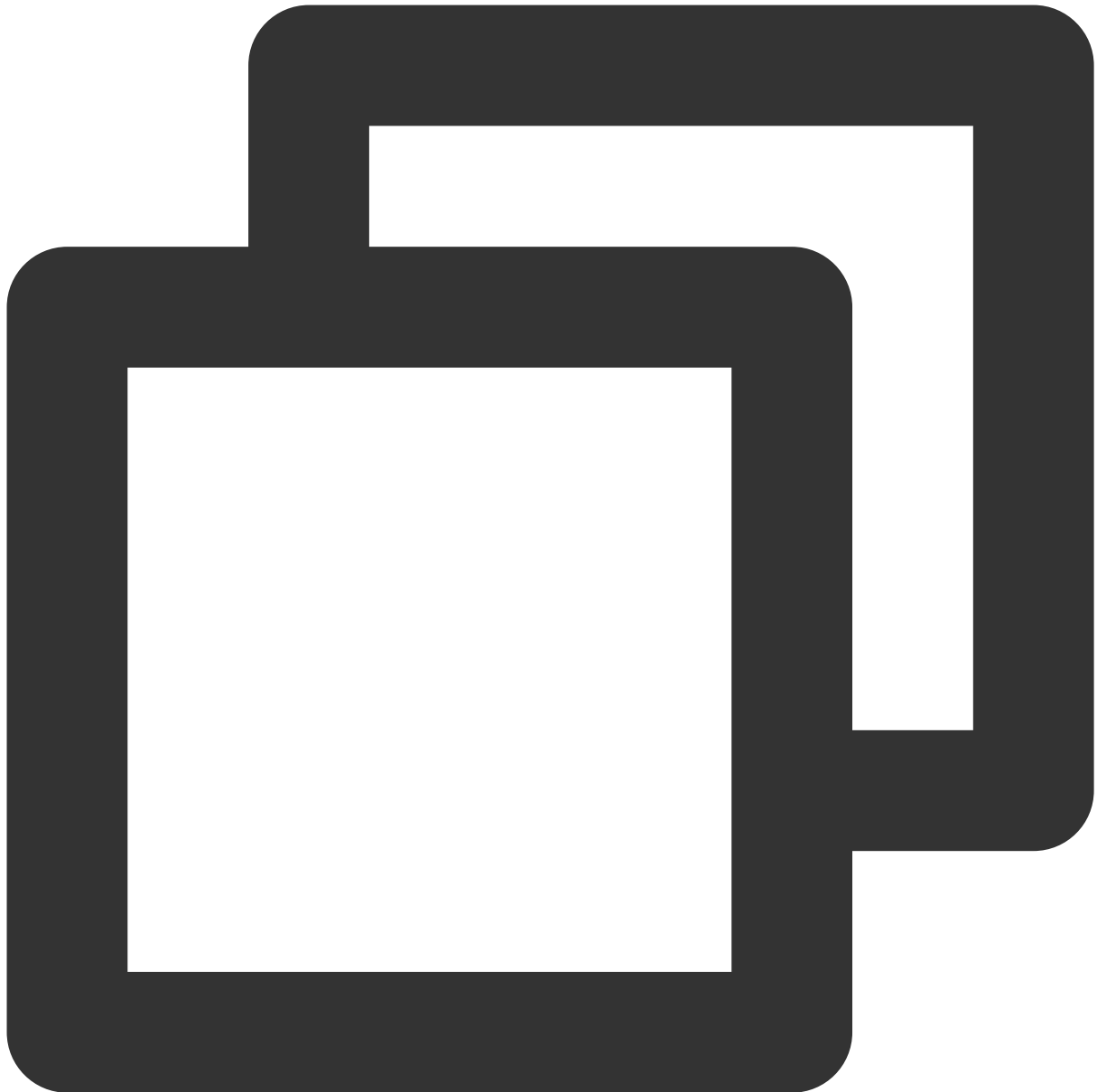
**Note**

For batch sending and other cases, see [TencentCloud/rocketmq-demo](#) in GitHub or the [Apache RocketMQ documentation](#).

**Step 3. Consume messages****Creating a consumer**

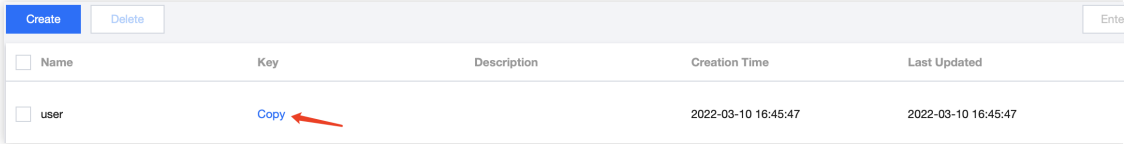
TDMQ for RocketMQ supports two consumption modes: push and pull. The push mode is recommended.





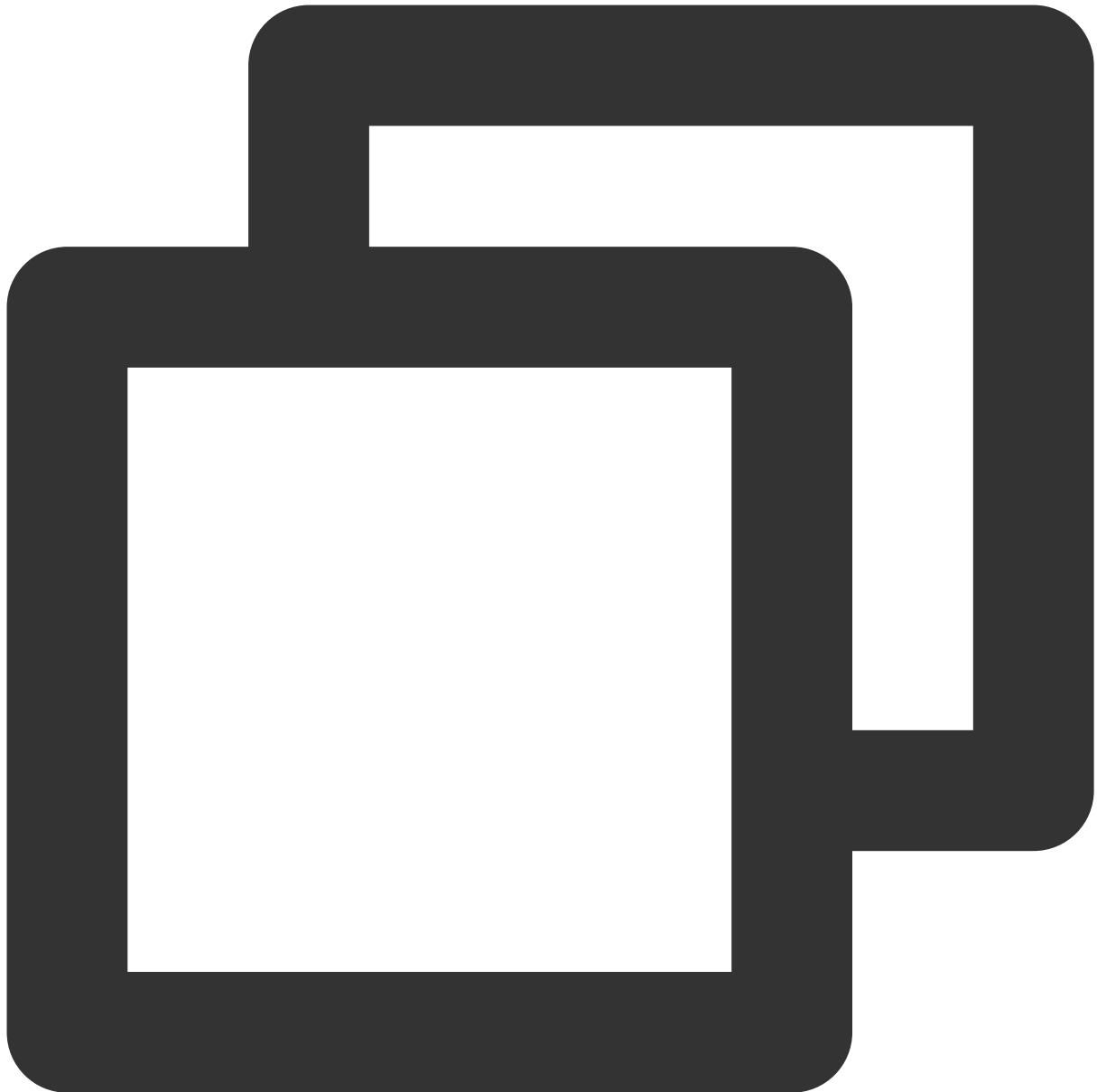
```
// Instantiate the consumer
DefaultMQPushConsumer pushConsumer = new DefaultMQPushConsumer(
    groupName,
    new AclClientRPCHook(new SessionCredentials(accessKey, secretKey))); //
// Set the Nameserver address
pushConsumer.setNamesrvAddr(nameserver);
```

Parameter	Description
groupName	Consumer group name, which can be copied under the <b>Group</b> tab on the <b>Cluster</b> page in the cons

nameserver	Cluster access address, which can be obtained from <b>Access Address</b> in the <b>Operation</b> column or the console. The namespace access address can be obtained under the <b>Namespace</b> tab on the <b>CI</b>
secretKey	Role name, which can be copied on the <a href="#">Role Management</a> page.
accessKey	Role token, which can be copied in the <b>Token</b> column on the <a href="#">Role Management</a> page. 

## Subscribing to messages

The subscription modes vary by consumption mode.



```
// Subscribe to a topic
pushConsumer.subscribe(topic_name, "*");
// Register a callback implementation class to process messages pulled from t
pushConsumer.registerMessageListener((MessageListenerConcurrently) (msgs, con
    // Message processing logic
    System.out.printf("%s Receive New Messages: %s %n", Thread.currentThread(
    // Mark the message as being successfully consumed and return the consump
    return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
});
// Start the consumer instance
pushConsumer.start();
```

Parameter	Description
topic_name	Topic name, which can be copied under the <b>Topic</b> tab on the <b>Cluster</b> page in the console.
"*"	If the subscription expression is left empty or specified as asterisk (*), all messages are subscribed to. <code>tag1    tag2    tag3</code> means subscribing to multiple types of tags.

### Step 4. View consumption details

Log in to the [TDMQ console](#), go to the **Cluster > Group** page, and view the list of clients connected to the consumer group. Click **Consumer Details** in the **Operation** column to view consumer details.

Create (2/10000)

Group Name	Consumer Info	Consumption Mode	Description
▼ group-364733	Online Consumer 0 TPS 0 Total Heap 0	Unknown	

**Basic Info**

Group Name	group-364733	Creation Time	2022-03-11 15:13:15
Consumption Mode	Unknown	Client Protocol	TCP
Total Heaped Messages	0	Consumer Type	Unknown

**Connected Client**

Client Address	Client Language	Client Version	Message Heap
No data yet			

Total items: 0 2

**Note**

Above is a brief introduction to message publishing and subscription. For more information, see [TencentCloud/rocketmq-demo](#) or the [Apache RocketMQ documentation](#).

# Sending and Receiving Delayed Messages

Last updated : 2023-05-16 11:07:52

## Overview

This document describes how to use open-source SDK to send and receive timed messages by using the SDK for Java as an example.

## 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 here](#) or have downloaded one at the [GitHub project](#).

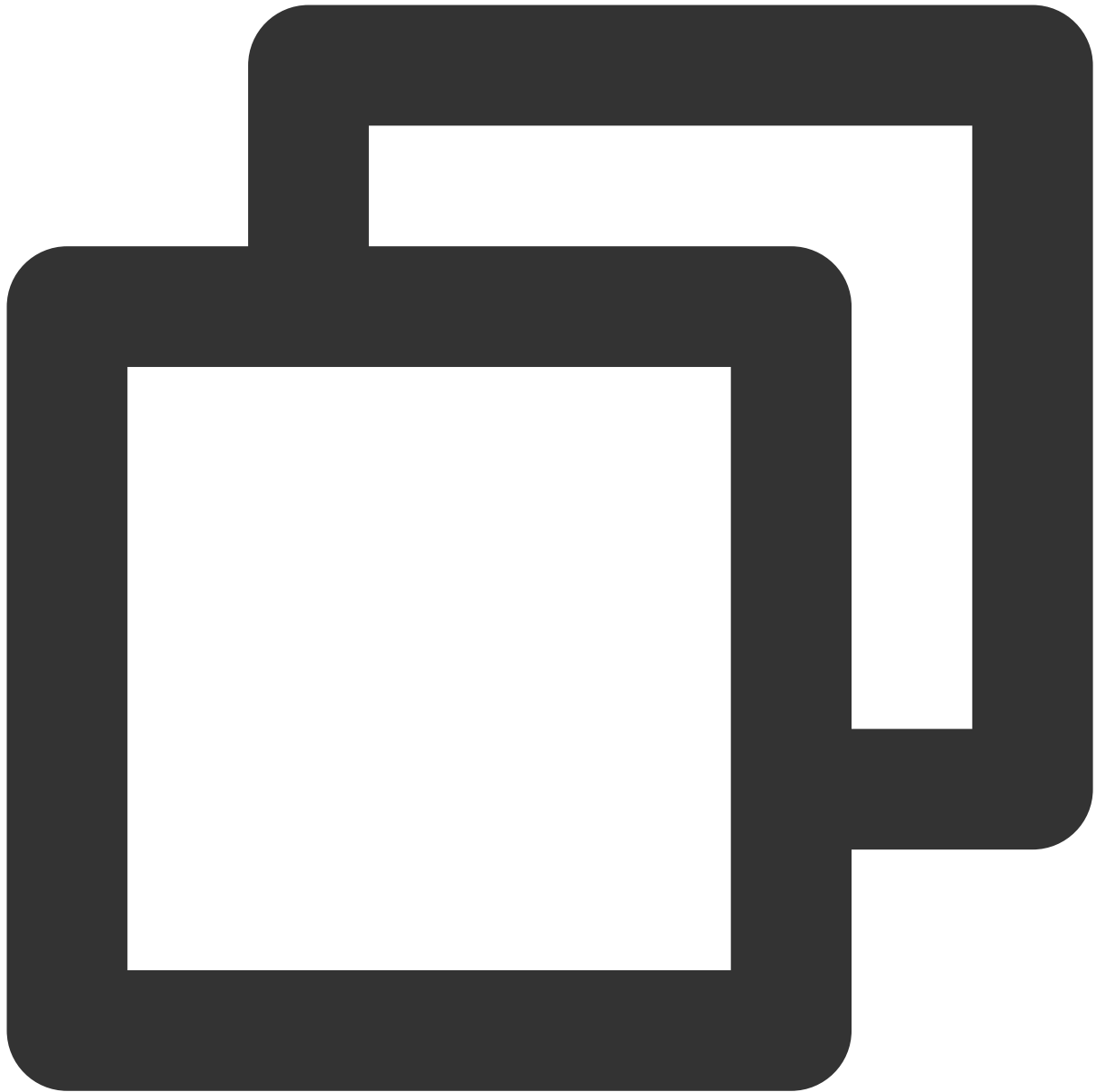
## Directions

### Step 1. Install the Java dependent library

Introduce dependencies in a Java project and add the following dependencies to the `pom.xml` file. This document uses a Maven project as an example.

#### Note

The dependency version must be v4.9.3 or later, preferably v4.9.4.



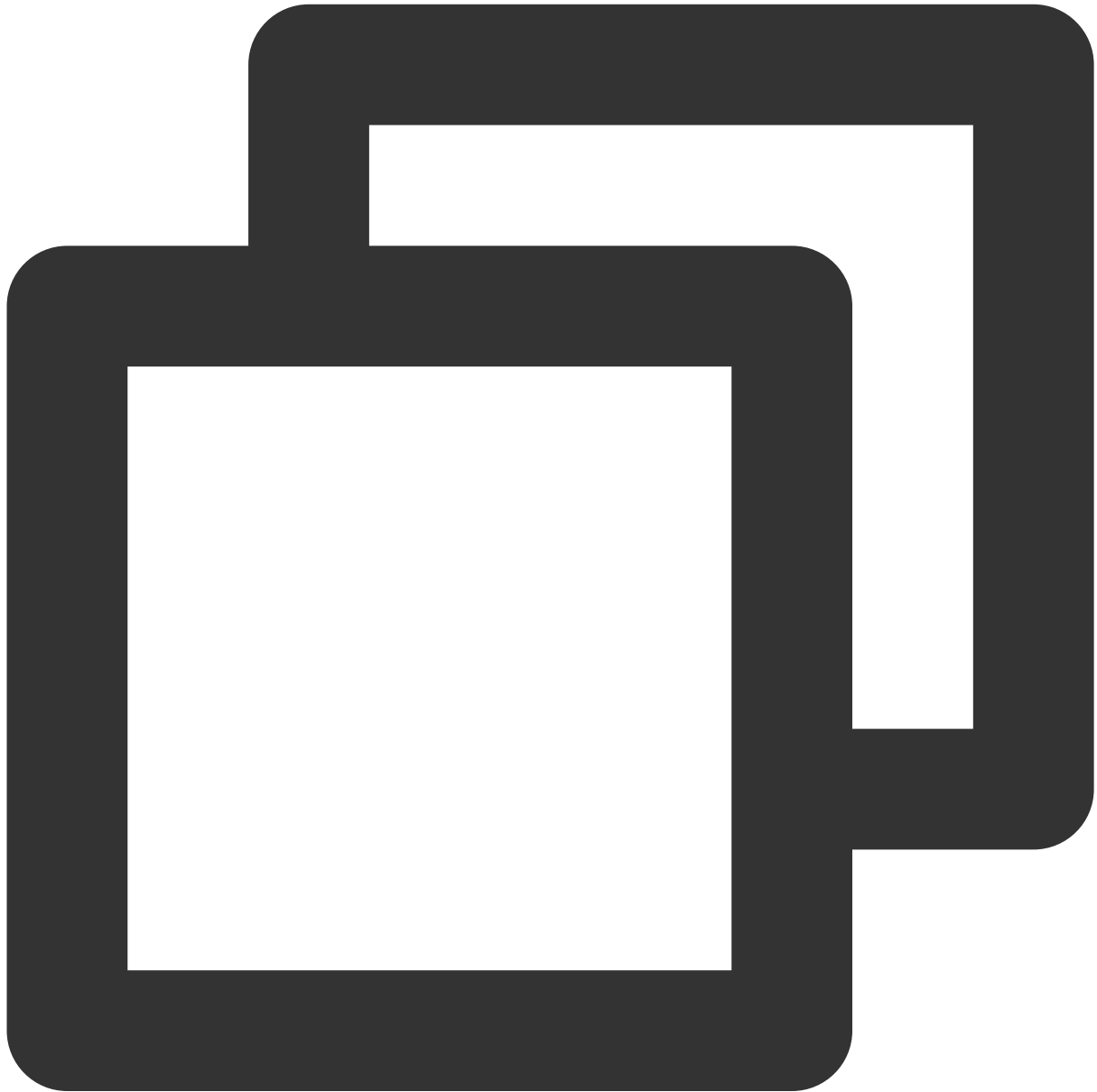
```
<!-- in your <dependencies> block -->
<dependency>
  <groupId>org.apache.rocketmq</groupId>
  <artifactId>rocketmq-client</artifactId>
  <version>4.9.4</version>
</dependency>

<dependency>
  <groupId>org.apache.rocketmq</groupId>
  <artifactId>rocketmq-acl</artifactId>
  <version>4.9.4</version>
```

```
</dependency>
```

## Step 2. Produce messages

### Creating a message producer



```
// Instantiate the message producer
DefaultMQProducer producer = new DefaultMQProducer(
    groupName,
    new AclClientRPCHook(new SessionCredentials(accessKey, secretKey)) // ACL pe
);
```

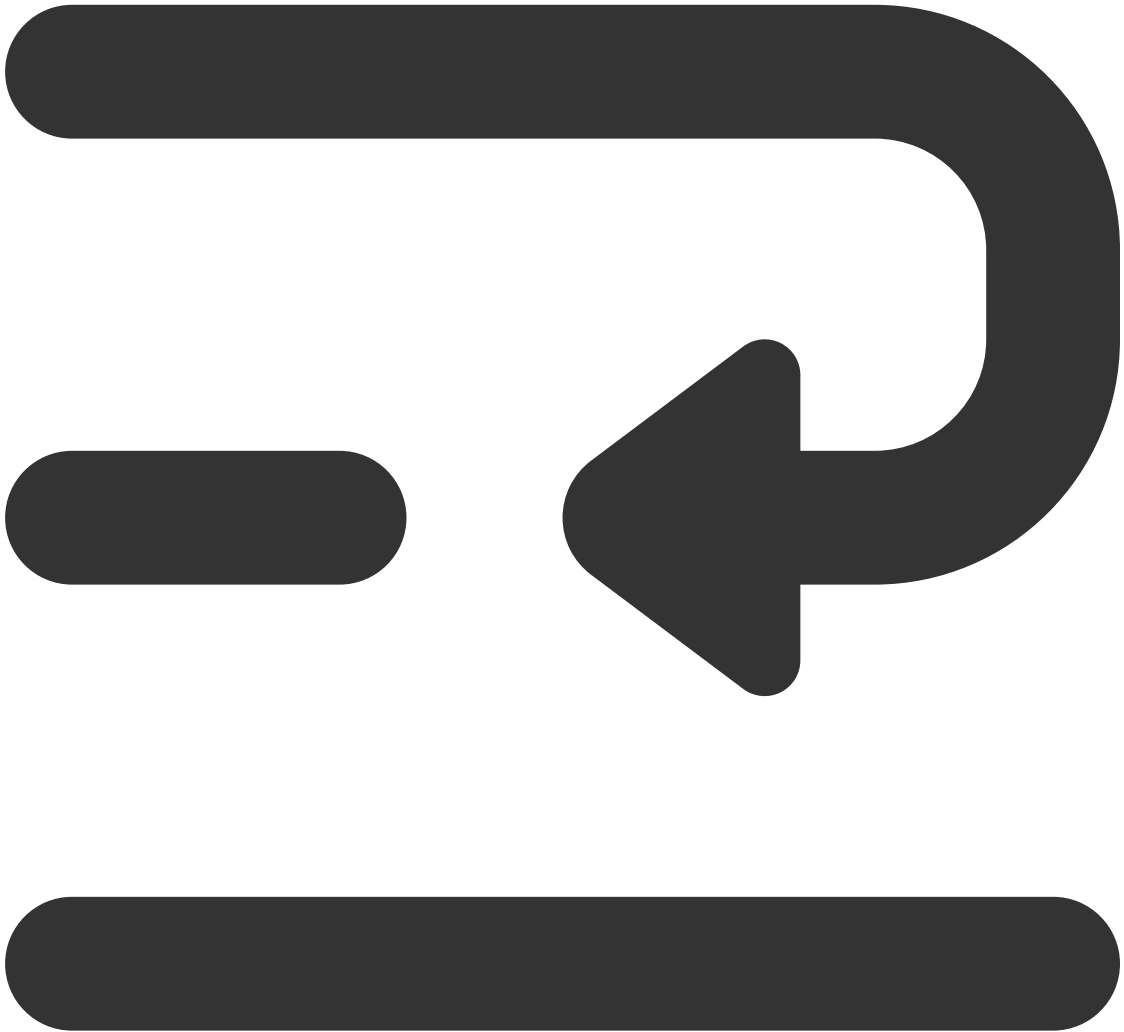
```
// Set the Nameserver address
producer.setNamesrvAddr(nameserver);
// Start the producer instance
producer.start();
```

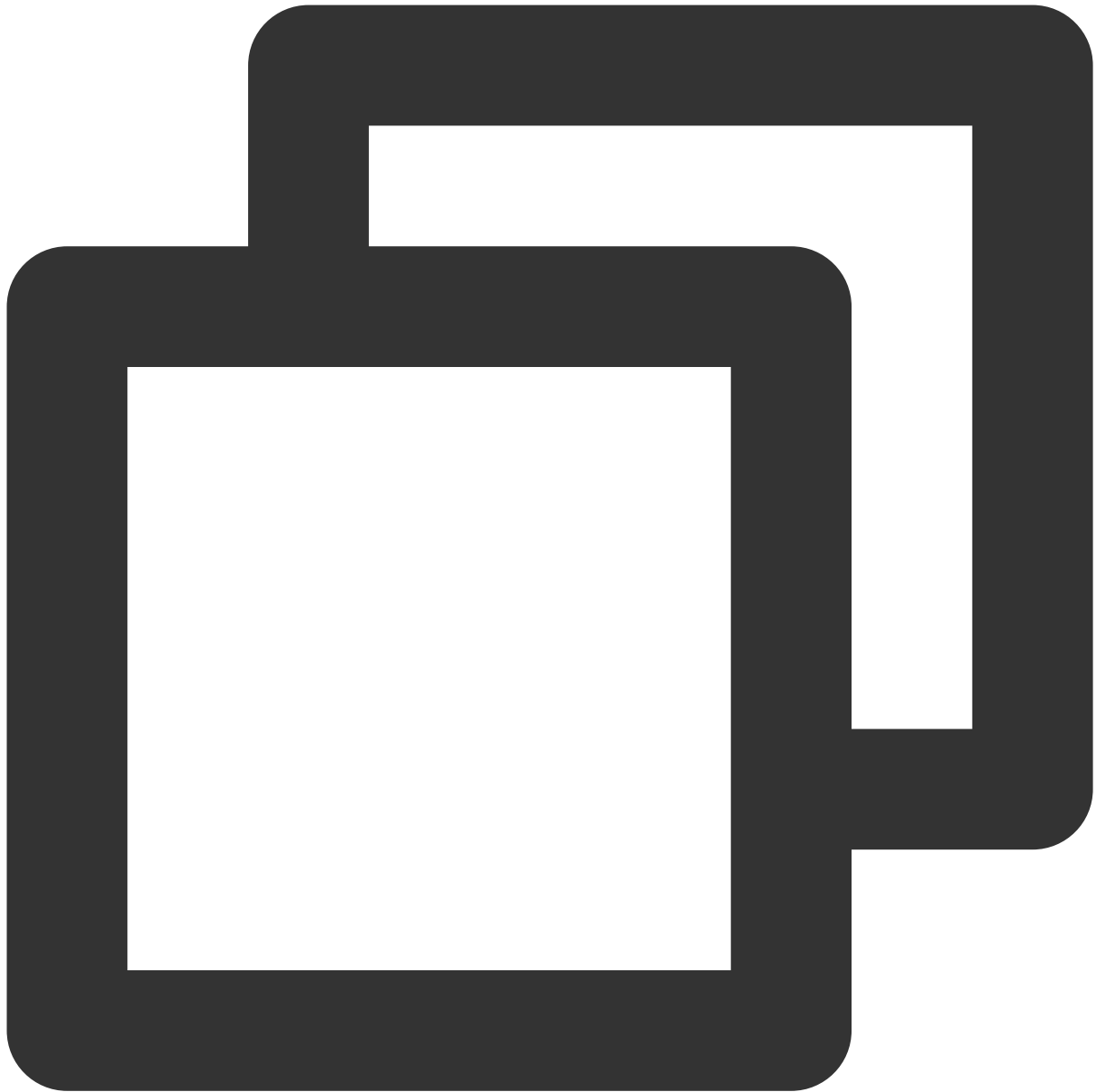
Parameter	Description												
groupName	Producer group name. It is recommended to use the corresponding topic name.												
nameserver	Cluster access address, which can be obtained from <b>Access Address</b> in the <b>Operation</b> column or <b>Management</b> page in the console. Namespace access addresses in new virtual or exclusive clusters are in the <b>Namespace</b> list.												
secretKey	Role name, which can be copied on the <a href="#">Role Management</a> page.												
accessKey	Role token, which can be copied in the <b>Token</b> column on the <a href="#">Role Management</a> page. <div data-bbox="384 902 1517 1151" data-label="Image"> <p>The screenshot shows a table with the following data:</p> <table border="1"> <thead> <tr> <th>Name</th> <th>Token</th> <th>Description</th> <th>Creation Time</th> <th>Last Updated</th> <th>Operation</th> </tr> </thead> <tbody> <tr> <td>test</td> <td>Copy</td> <td>dai_test</td> <td>2022-03-10 16:46:15</td> <td>2022-03-10 16:46:15</td> <td>View, Edit</td> </tr> </tbody> </table> </div>	Name	Token	Description	Creation Time	Last Updated	Operation	test	Copy	dai_test	2022-03-10 16:46:15	2022-03-10 16:46:15	View, Edit
Name	Token	Description	Creation Time	Last Updated	Operation								
test	Copy	dai_test	2022-03-10 16:46:15	2022-03-10 16:46:15	View, Edit								

## Sending a message

### Messages with fixed delay level

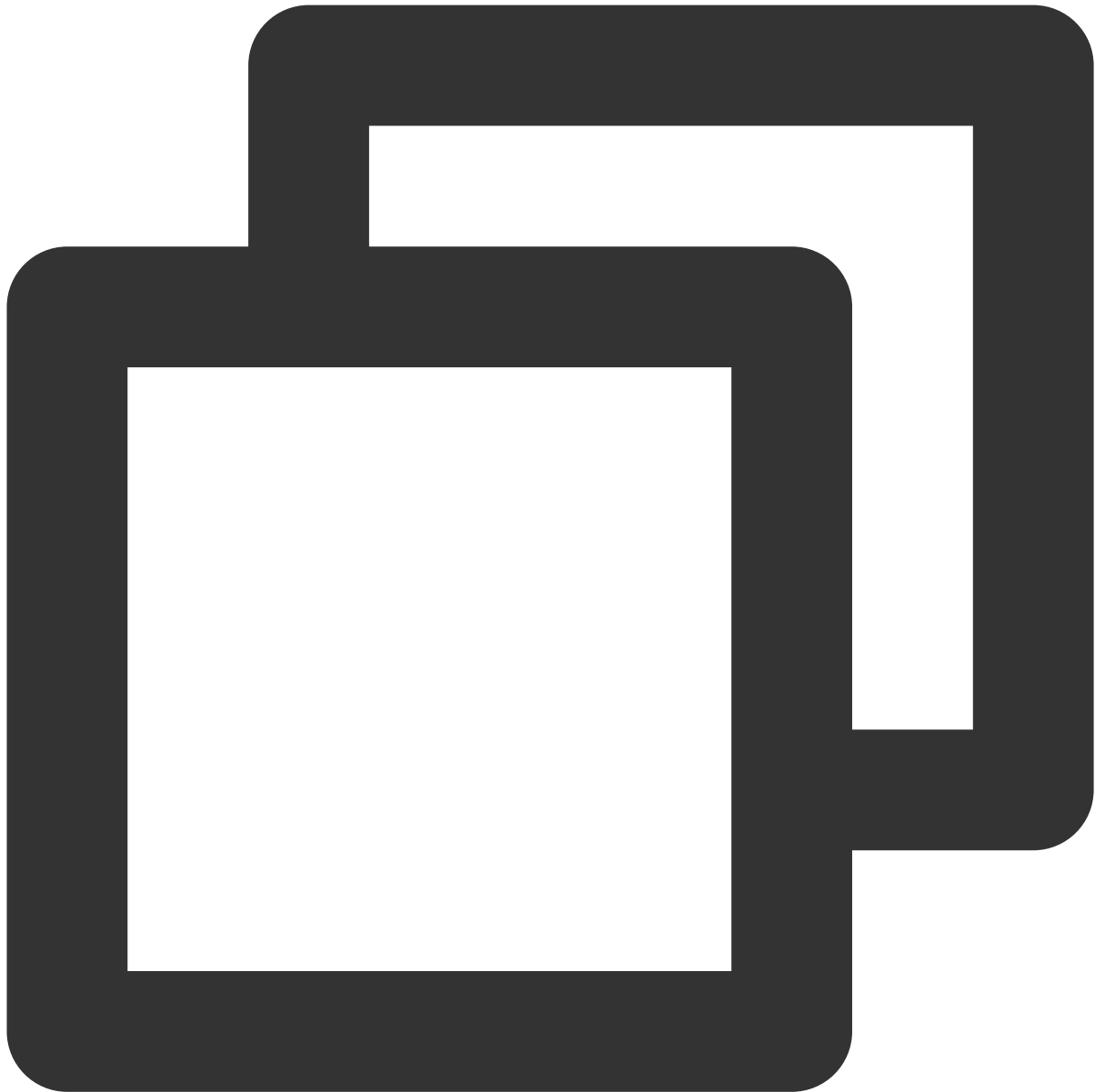






```
int totalMessagesToSend = 5;
for (int i = 0; i < totalMessagesToSend; i++) {
    Message message = new Message(TOPIC_NAME, ("Hello scheduled message " + i).getBytes());
    // Set message delay level
    message.setDelayTimeLevel(5);
    // Send the message
    SendResult sendResult = producer.send(message);
    System.out.println("sendResult = " + sendResult);
}
```

## Messages with random delay time



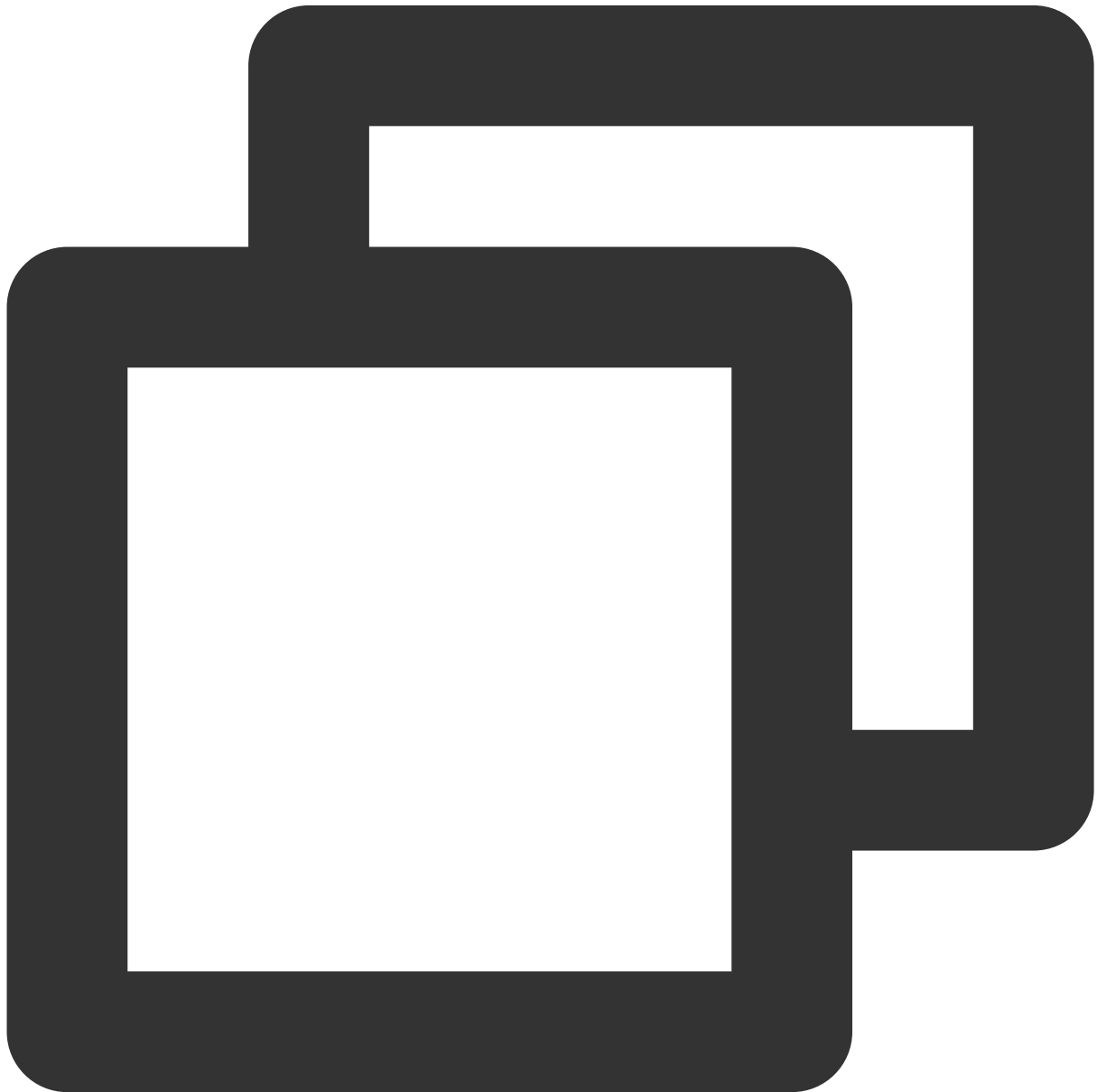
```
int totalMessagesToSend = 1;
for (int i = 0; i < totalMessagesToSend; i++) {
    Message message = new Message(TOPIC_NAME, ("Hello timer message " + i).getBytes
    // Set the time for sending the message
    long timeStamp = System.currentTimeMillis() + 30000;
    // To send a timed message, you need to specify a time for it, and the message
    // If the timestamp is set before the current time, the message will be deliver
    // Set `__STARTDELIVERTIME` into the property of `msg`
    message.putUserProperty("__STARTDELIVERTIME", String.valueOf(timeStamp));
}
```

```
// Send the message
SendResult sendResult = producer.send(message);
System.out.println("sendResult = " + sendResult);
}
```

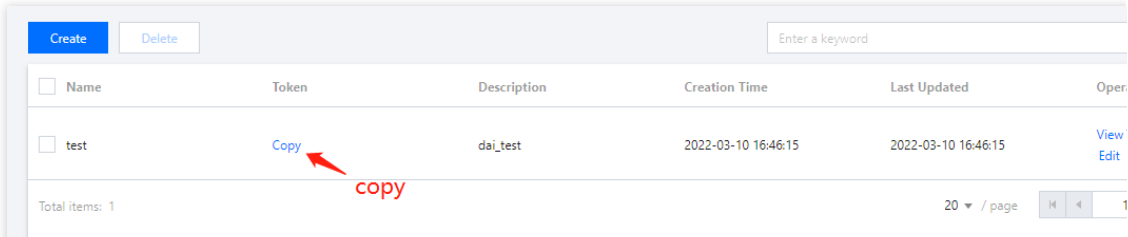
### Step 3. Consume messages

#### ####Creating a consumer

TDMQ for RocketMQ supports two consumption modes: push and pull. Push mode is recommended.

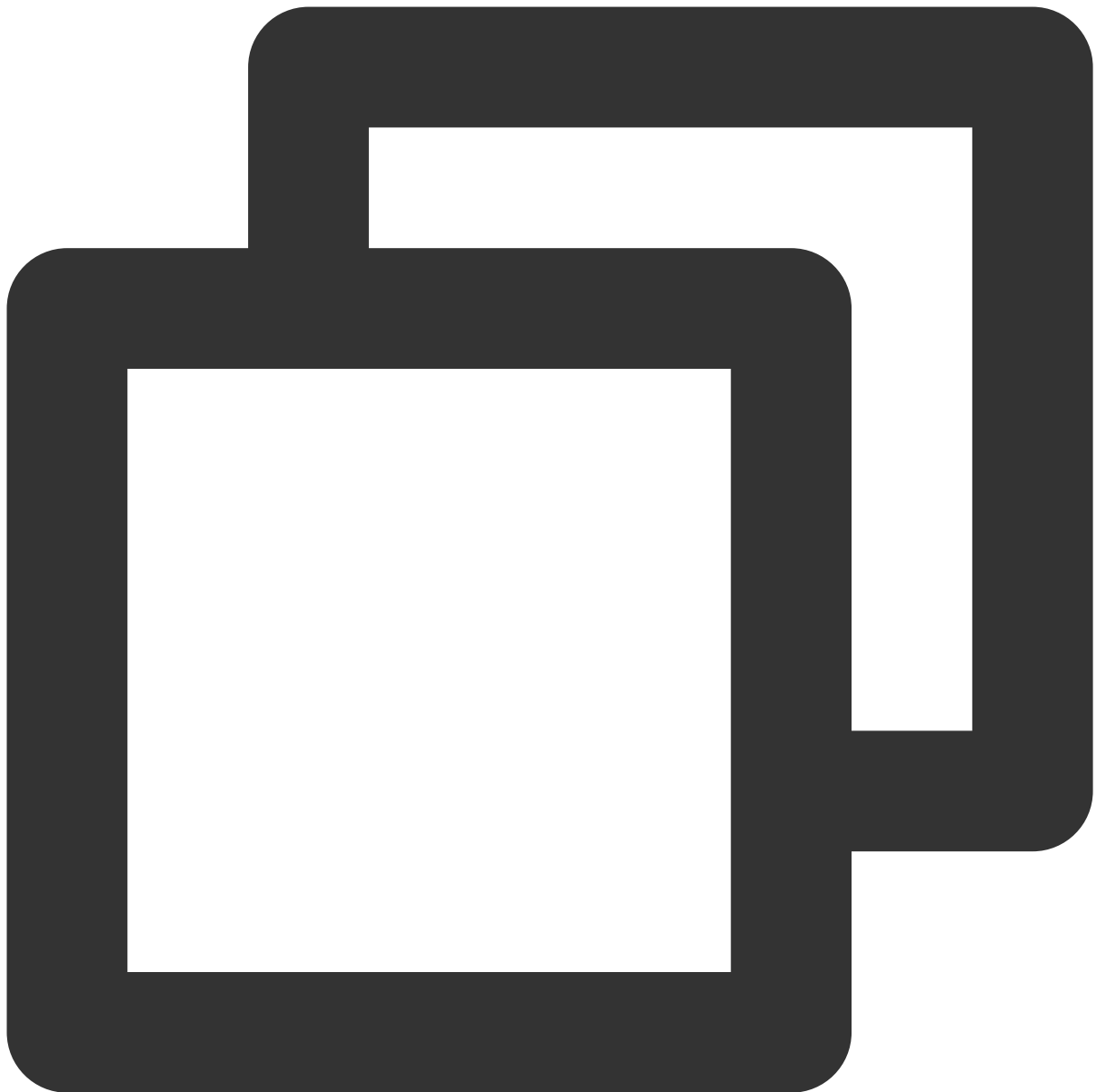


```
// Instantiate the consumer
    DefaultMQPushConsumer pushConsumer = new DefaultMQPushConsumer(
        groupName,
        new AclClientRPCHook(new SessionCredentials(accessKey, secretKey))); //
// Set the Nameserver address
pushConsumer.setNamesrvAddr(nameserver);
```

Parameter	Description
groupName	Producer group name, which can be copied under the <b>Group</b> tab on the <b>Cluster</b> page in the console
nameserver	Cluster access address, which can be obtained from <b>Access Address</b> in the <b>Operation</b> column or <b>Management</b> page in the console. Namespace access addresses in new virtual or exclusive clusters are in the <b>Namespace</b> list.
secretKey	Role name, which can be copied on the <a href="#">Role Management</a> page.
accessKey	Role token, which can be copied in the <b>Token</b> column on the <a href="#">Role Management</a> page. 

## Subscribing to messages

The subscription modes vary by consumption mode.



```
// Subscribe to a topic
pushConsumer.subscribe(topic_name, "*");
// Register a callback implementation class to process messages pulled from t
pushConsumer.registerMessageListener((MessageListenerConcurrently) (msgs, con
    // Message processing logic
    System.out.printf("%s Receive New Messages: %s %n", Thread.currentThread(
    // Mark the message as being successfully consumed and return the consump
    return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
});
// Start the consumer instance
pushConsumer.start();
```

Parameter	Description
topic_name	Topic name, which can be copied under the <b>Topic</b> tab on the <b>Cluster</b> page in the console.
"*"	If the subscription expression is left empty or specified as asterisk (*), all messages are subscribed to. <code>tag1    tag2    tag3</code> means subscribing to multiple types of tags.

### Step 4. View consumption details

Log in to the [TDMQ console](#), go to the **Cluster > Group** page, and view the list of clients connected to the group.

Click **View Details** in the **Operation** column to view consumer details.

**Basic Info**

Group Name	group-364733	Creation Time	2022-03-11 15:13:15
Consumption Mode	Unknown	Client Protocol	TCP
Total Heaped Messages	0	Consumer Type	Unknown

**Client Address**    Subscription

---

Client Address	Client Language	Client Version	Message Heap ↕
No data yet			

Total items: 0

Basic Info
Namespace
Topic
Group

Current Namespace

sdaa ▼

Message Retention Period
3 days
Max TPS ⓘ
4000

Create (2/1500)

Search by keyword

Group Name	Consumer Info ↕	Consumption Mode	Descriptio
group-364733	Online Consumer <span style="margin-left: 10px;">0</span> TPS <span style="margin-left: 10px;">0</span> Total Heap <span style="margin-left: 10px;">0</span> <span style="float: right;">↻</span>	Unknown	
dasda	Online Consumer <span style="margin-left: 10px;">0</span> TPS <span style="margin-left: 10px;">0</span> Total Heap <span style="margin-left: 10px;">0</span> <span style="float: right;">↻</span>	Unknown	

Total items: 2

**Note**  
 Above is a brief introduction to message publishing and subscription. For more information, see [Demo](#) or [RocketMQ documentation](#).



# Sending and Receiving Sequential Messages

Last updated : 2023-05-16 11:07:52

## Overview

This document describes how to use open-source SDK to send and receive timed messages by using the SDK for Java as an example.

## Prerequisites

You have created the required resources. If it is a globally sequential message, you need to create a single-queue topic. For more information, see [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 here](#) or have downloaded one at the [GitHub project](#).

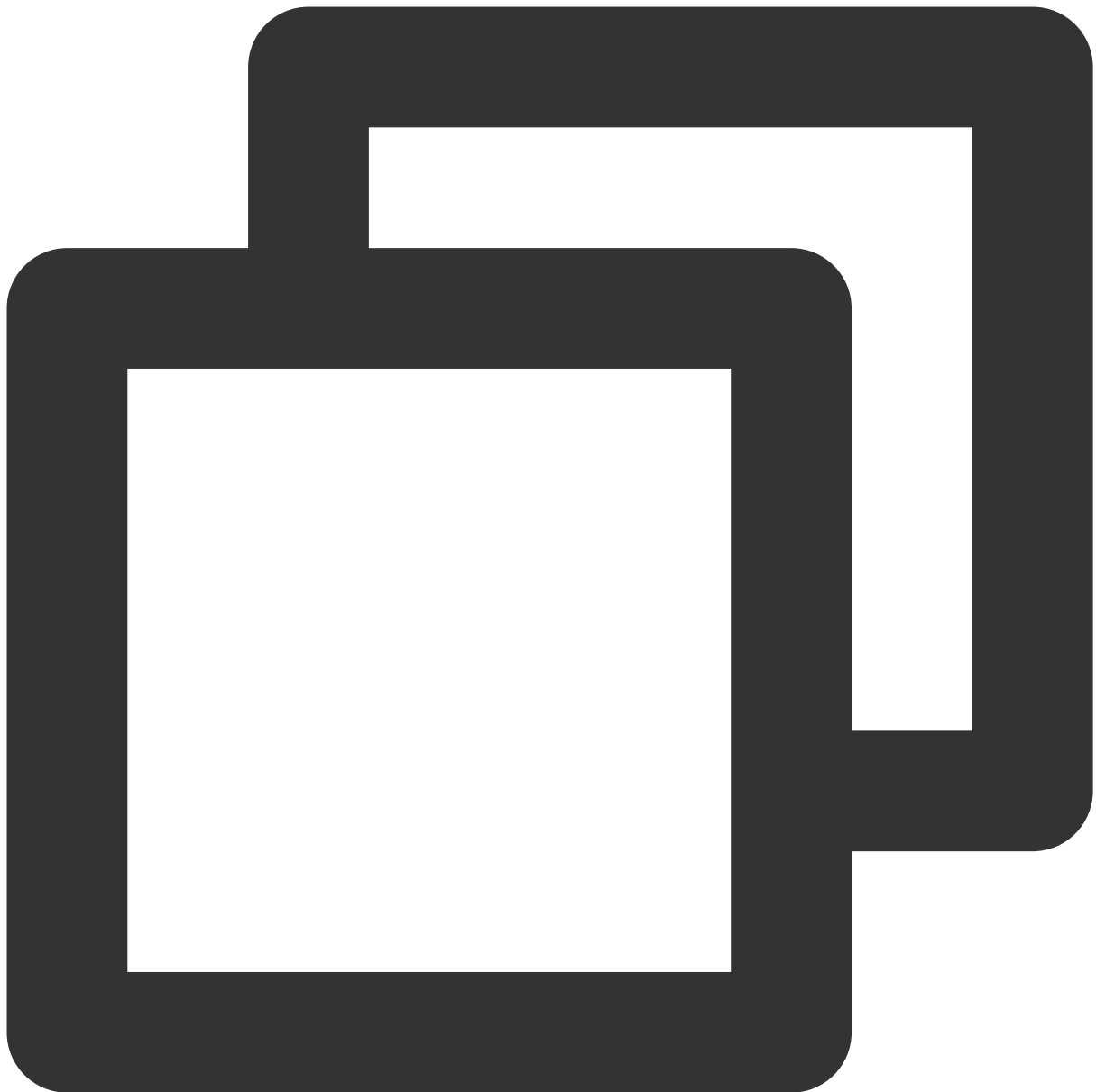
## Directions

### Step 1. Install the Java dependent library

Introduce dependencies in a Java project and add the following dependencies to the `pom.xml` file. This document uses a Maven project as an example.

#### Note

The dependency version must be v4.9.3 or later, preferably v4.9.4.



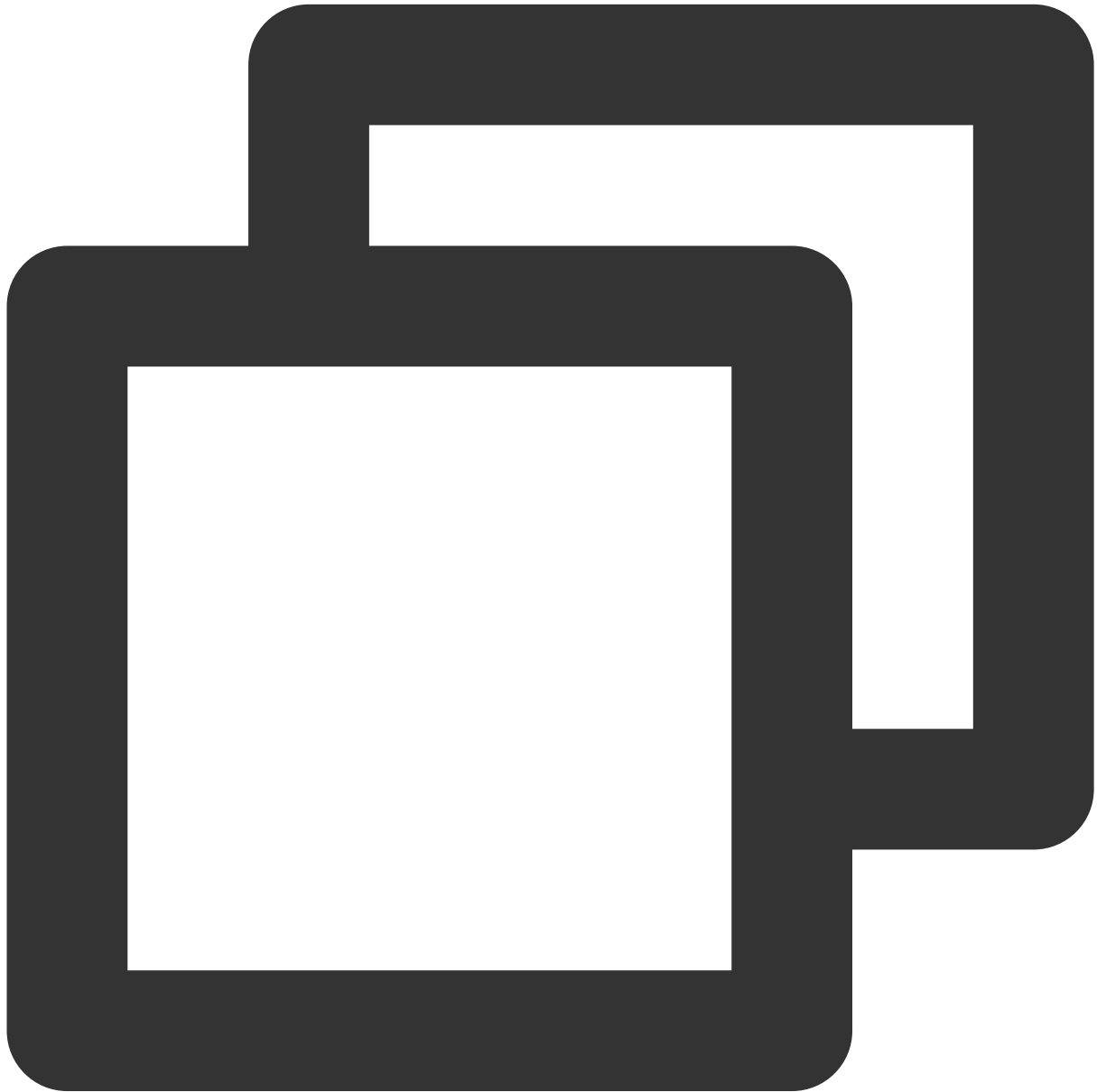
```
<!-- in your <dependencies> block -->
<dependency>
  <groupId>org.apache.rocketmq</groupId>
  <artifactId>rocketmq-client</artifactId>
  <version>4.9.4</version>
</dependency>

<dependency>
  <groupId>org.apache.rocketmq</groupId>
  <artifactId>rocketmq-acl</artifactId>
  <version>4.9.4</version>
```

```
</dependency>
```

## Step 2. Produce messages

### Creating a message producer



```
// Instantiate the message producer
DefaultMQProducer producer = new DefaultMQProducer(
    groupName,
    new AclClientRPCHook(new SessionCredentials(accessKey, secretKey)) // ACL pe
);
```

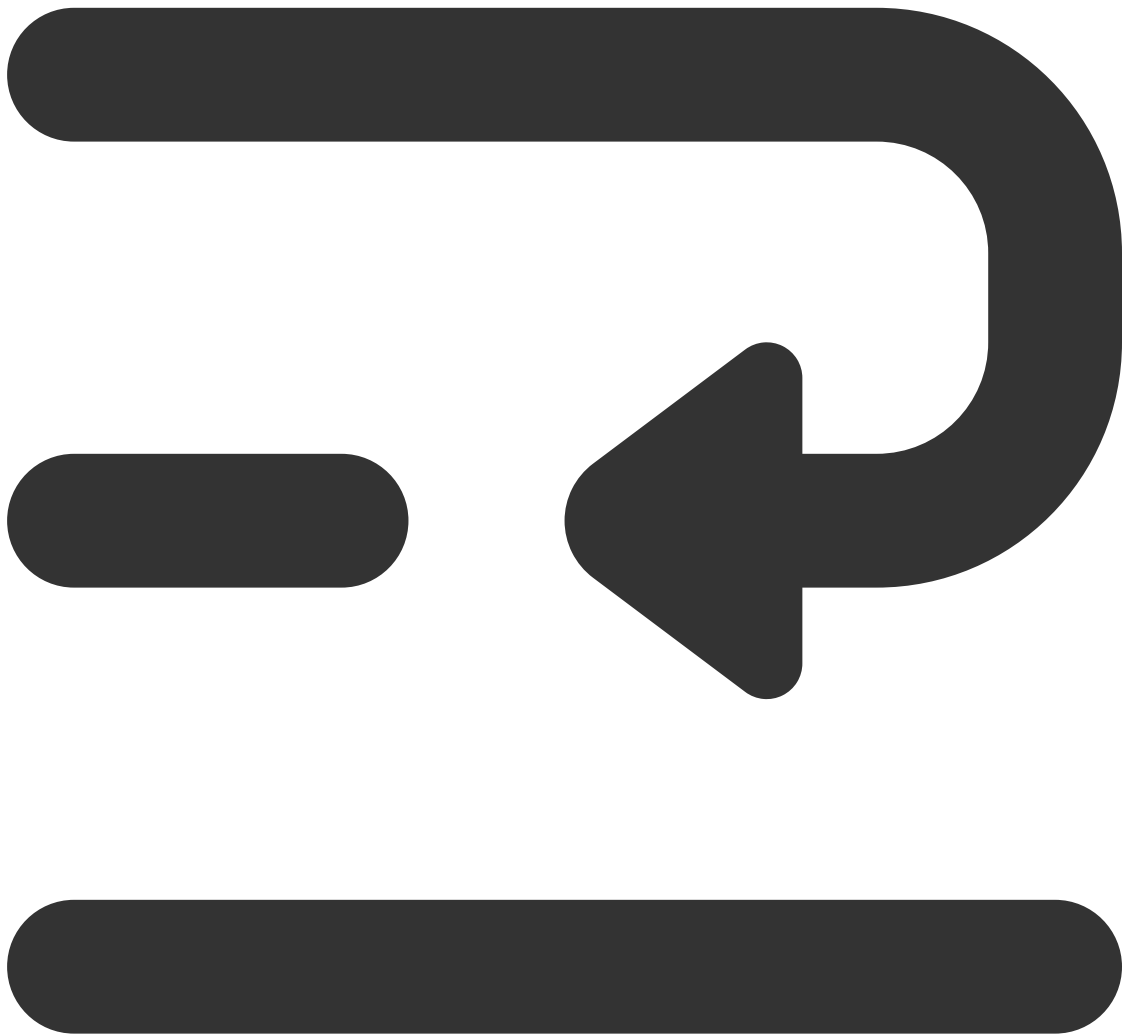
```
// Set the Nameserver address
producer.setNamesrvAddr(nameserver);
// Start the producer instance
producer.start();
```

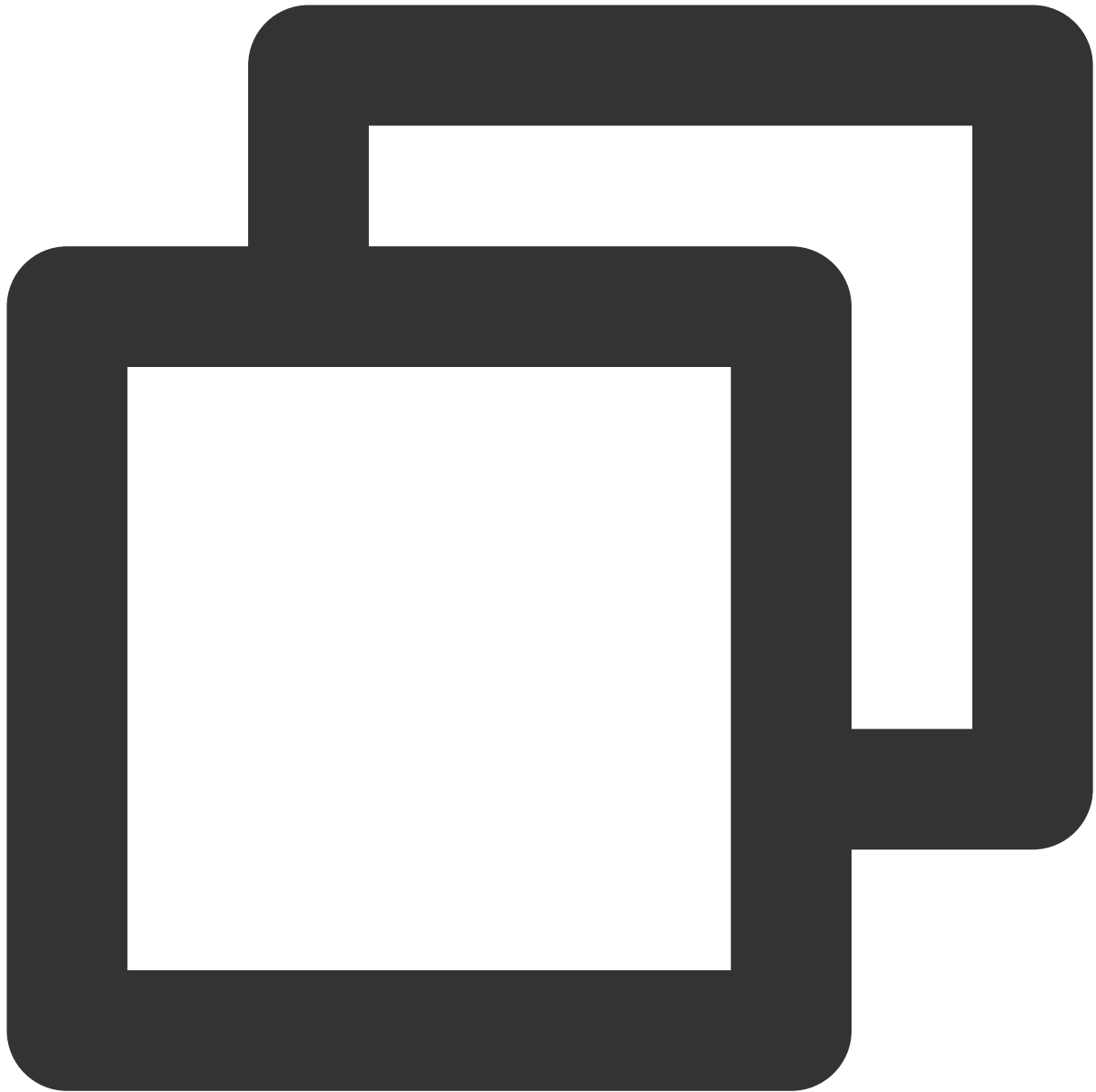
Parameter	Description												
groupName	Producer group name. It is recommended to use the corresponding topic name.												
nameserver	Cluster access address, which can be obtained from <b>Access Address</b> in the <b>Operation</b> column or <b>Management</b> page in the console. Namespace access addresses in new virtual or exclusive clusters from the <b>Namespace</b> list.												
secretKey	Role name, which can be copied on the <a href="#">Role Management</a> page.												
accessKey	Role token, which can be copied in the <b>Token</b> column on the <a href="#">Role Management</a> page. <div data-bbox="384 904 1517 1149" data-label="Image"> <table border="1"> <thead> <tr> <th>Name</th> <th>Token</th> <th>Description</th> <th>Creation Time</th> <th>Last Updated</th> <th>Operations</th> </tr> </thead> <tbody> <tr> <td>test</td> <td>Copy</td> <td>dai_test</td> <td>2022-03-10 16:46:15</td> <td>2022-03-10 16:46:15</td> <td>View T Edit  </td> </tr> </tbody> </table> </div>	Name	Token	Description	Creation Time	Last Updated	Operations	test	Copy	dai_test	2022-03-10 16:46:15	2022-03-10 16:46:15	View T Edit
Name	Token	Description	Creation Time	Last Updated	Operations								
test	Copy	dai_test	2022-03-10 16:46:15	2022-03-10 16:46:15	View T Edit								

## Sending a message

### Globally sequential message

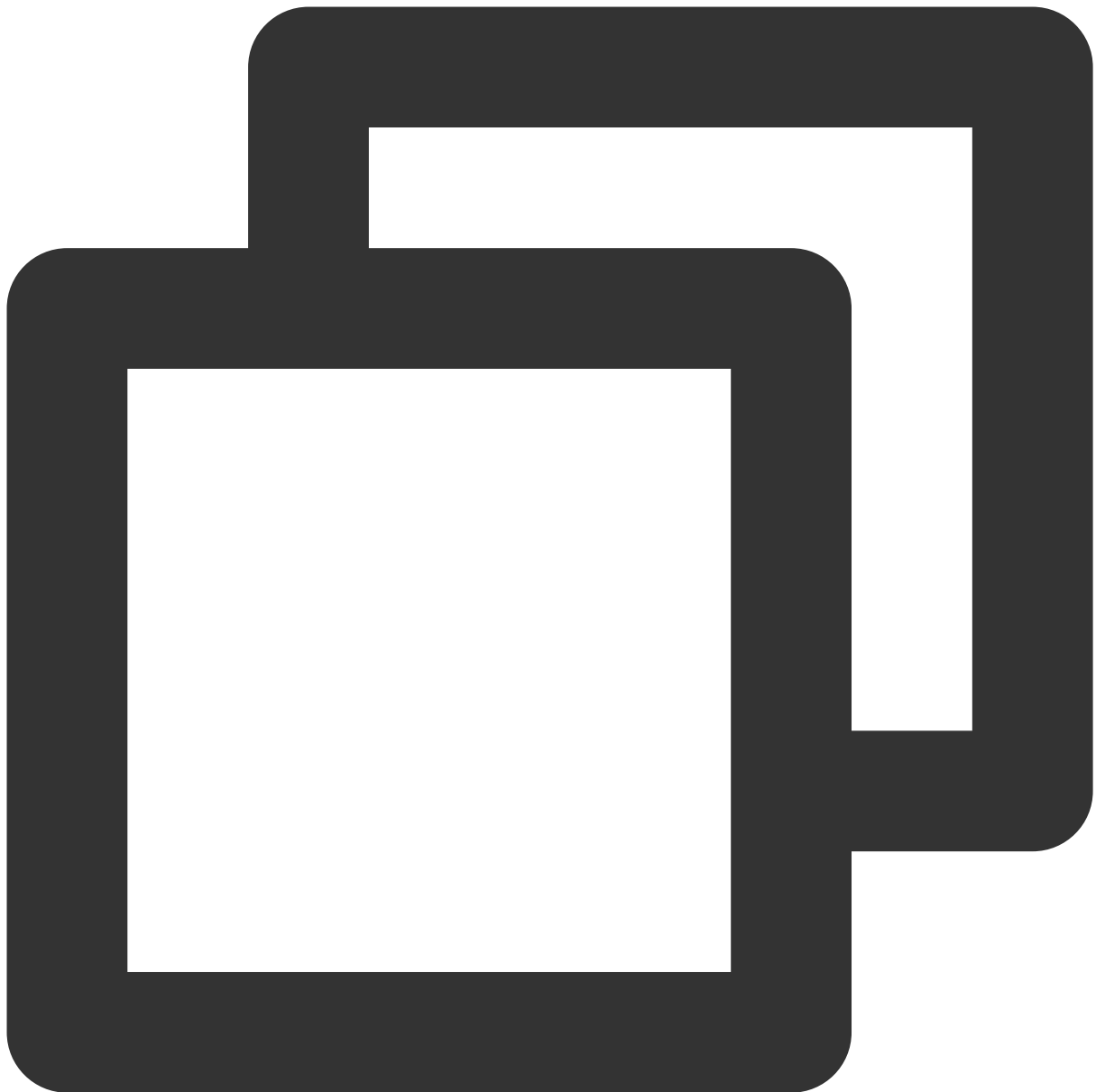
This process is the same as that of general messages.





```
int totalMessagesToSend = 5;
for (int i = 0; i < totalMessagesToSend; i++) {
    Message message = new Message(TOPIC_NAME, ("Hello scheduled message " + i).getBytes());
    // Send the message
    SendResult sendResult = producer.send(message);
    System.out.println("sendResult = " + sendResult);
}
```

### Partially sequential message



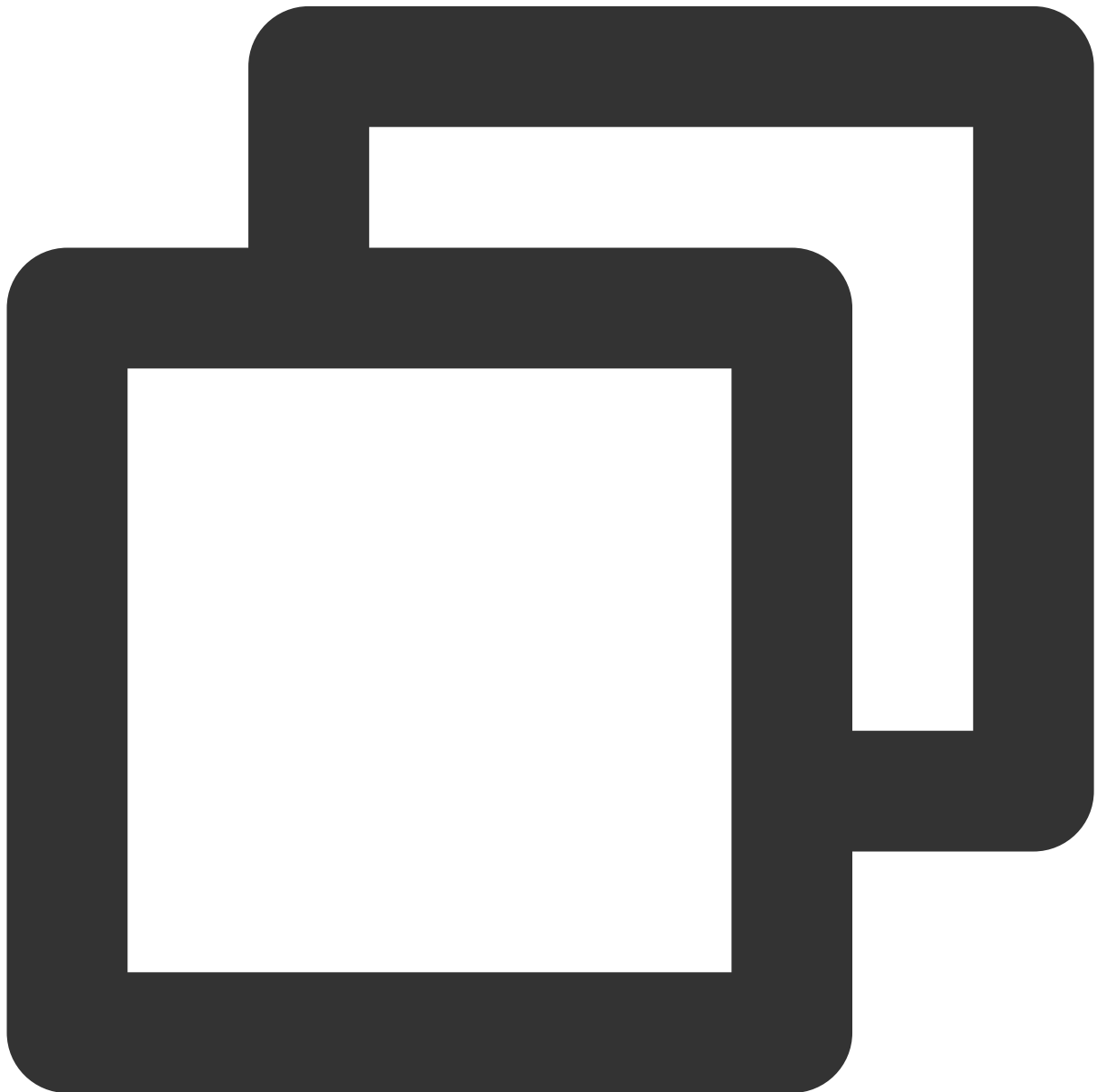
```
for (int i = 0; i < 3; i++) {
    int orderId = i % 3;
    // Construct message instance
    Message msg = new Message(TOPIC_NAME, "your tag", "KEY" + i, ("Hello RocketMQ ")
    SendResult sendResult = producer.send(msg, new MessageQueueSelector() {
        @Override
        public MessageQueue select(List<MessageQueue> mqs, Message msg1, Object arg)
            Integer id = (Integer) arg;
            int index = id % mqs.size();
            return mqs.get(index);
        }
    }
```

```
    }, orderId);  
    System.out.printf("%s%n", sendResult);  
}
```

### Step 3. Consume messages

#### ####Creating a consumer

TDMQ for RocketMQ supports two consumption modes: push and pull. Push mode is recommended.



```
// Instantiate the consumer
```

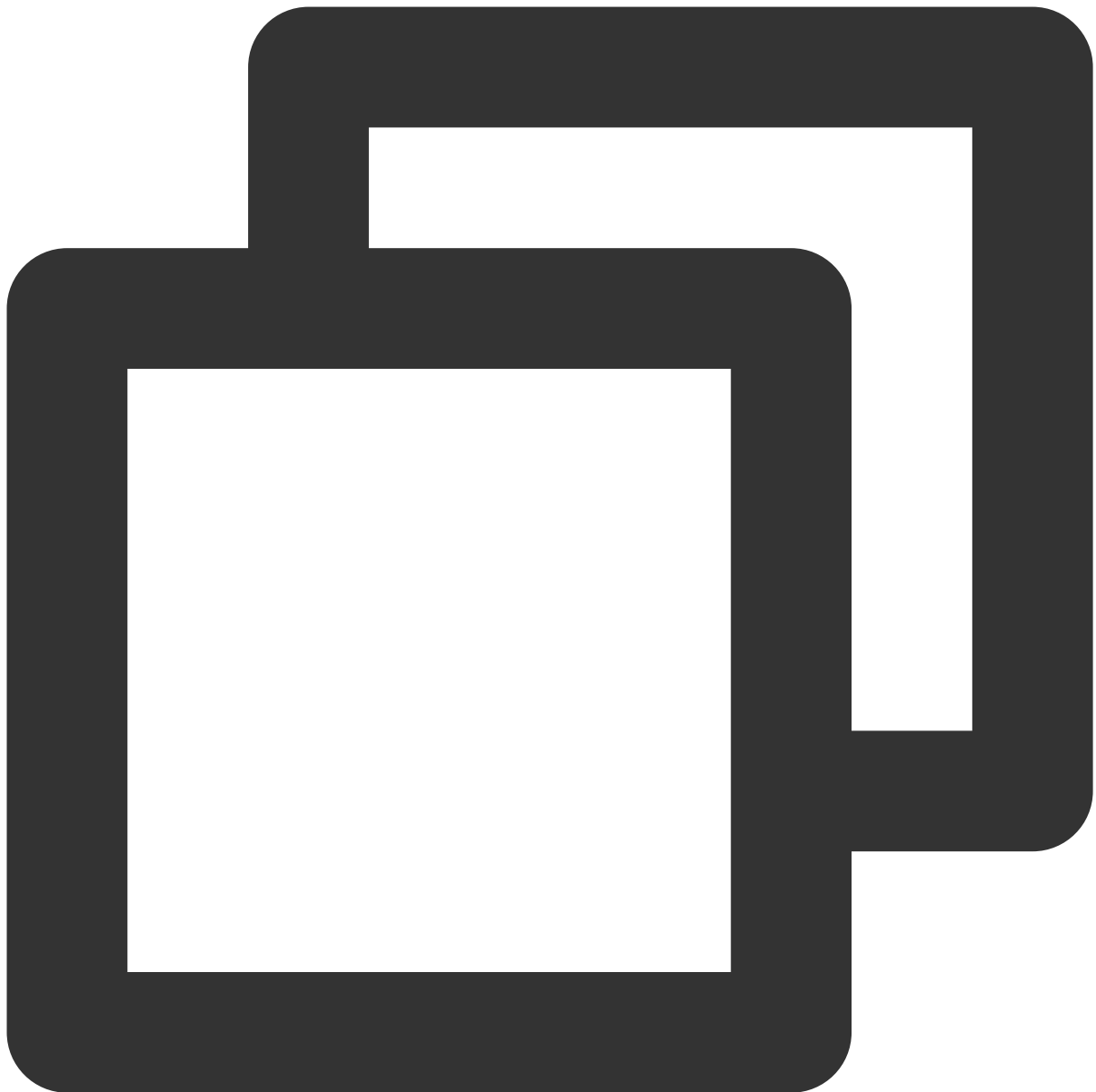


```
DefaultMQPushConsumer pushConsumer = new DefaultMQPushConsumer(
    groupName,
    new AclClientRPCHook(new SessionCredentials(accessKey, secretKey))); //
// Set the Nameserver address
pushConsumer.setNamesrvAddr(nameserver);
```

Parameter	Description												
groupName	Producer group name, which can be copied under the <b>Group</b> tab on the <b>Cluster</b> page in the console												
nameserver	Cluster access address, which can be obtained from <b>Access Address</b> in the <b>Operation</b> column or <b>Management</b> page in the console. Namespace access addresses in new virtual or exclusive clusters from the <b>Namespace</b> list.												
secretKey	Role name, which can be copied on the <a href="#">Role Management</a> page.												
accessKey	Role token, which can be copied in the <b>Token</b> column on the <a href="#">Role Management</a> page. <div data-bbox="384 902 1530 1149" data-label="Image"> <table border="1"> <thead> <tr> <th>Name</th> <th>Token</th> <th>Description</th> <th>Creation Time</th> <th>Last Updated</th> <th>Operations</th> </tr> </thead> <tbody> <tr> <td>test</td> <td>Copy</td> <td>dai_test</td> <td>2022-03-10 16:46:15</td> <td>2022-03-10 16:46:15</td> <td>View T Edit T</td> </tr> </tbody> </table> </div>	Name	Token	Description	Creation Time	Last Updated	Operations	test	Copy	dai_test	2022-03-10 16:46:15	2022-03-10 16:46:15	View T Edit T
Name	Token	Description	Creation Time	Last Updated	Operations								
test	Copy	dai_test	2022-03-10 16:46:15	2022-03-10 16:46:15	View T Edit T								

## Subscribing to messages

The subscription modes vary by consumption mode.



```
// Subscribe to a topic
pushConsumer.subscribe(topic_name, "*");
// Register a callback implementation class to process messages pulled from t
pushConsumer.registerMessageListener((MessageListenerConcurrently) (msgs, con
    // Message processing logic
    System.out.printf("%s Receive New Messages: %s %n", Thread.currentThread(
    // Mark the message as being successfully consumed and return the consump
    return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
});
// Start the consumer instance
pushConsumer.start();
```

Parameter	Description
topic_name	Topic name, which can be copied under the <b>Topic</b> tab on the <b>Cluster</b> page in the console.
"*"	If the subscription expression is left empty or specified as asterisk (*), all messages are subscribed to. <code>tag1    tag2    tag3</code> means subscribing to multiple types of tags.

### Step 4. View consumption details

Log in to the [TDMQ console](#), go to the **Cluster > Group** page, and view the list of clients connected to the group.

Click **View Details** in the **Operation** column to view consumer details.

**Basic Info**

Group Name	group-364733	Creation Time	2022-03-11 15:13:15
Consumption Mode	Unknown	Client Protocol	TCP
Total Heaped Messages	0	Consumer Type	Unknown

**Client Address**    Subscription

---

Client Address	Client Language	Client Version	Message Heap ↕
No data yet			

Total items: 0

Basic Info
Namespace
Topic
Group

Current Namespace sdaa

Message Retention Period 3 days

Max TPS ⓘ 4000

Create (2/1500)

Search by keyword

Group Name	Consumer Info ↕	Consumption Mode	Descriptio
group-364733	Online Consumer <span style="margin-left: 10px;">0</span> TPS <span style="margin-left: 10px;">0</span> Total Heap <span style="margin-left: 10px;">0</span> <span style="float: right;">↻</span>	Unknown	
dasda	Online Consumer <span style="margin-left: 10px;">0</span> TPS <span style="margin-left: 10px;">0</span> Total Heap <span style="margin-left: 10px;">0</span> <span style="float: right;">↻</span>	Unknown	

Total items: 2

**Note**  
 Above is a brief introduction to message publishing and subscription. For more information, see [Demo](#) or [RocketMQ documentation](#).

# Sending and Receiving Transactional Messages

Last updated : 2023-05-16 11:07:52

## Overview

This document describes how to use open-source SDK to send and receive transactional messages by using the SDK for Java as an example.

## Prerequisites

You have created the required resources. If it is a globally sequential message, you need to create a single-queue topic. For more information, see [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 here](#) or have downloaded one at the [GitHub project](#).

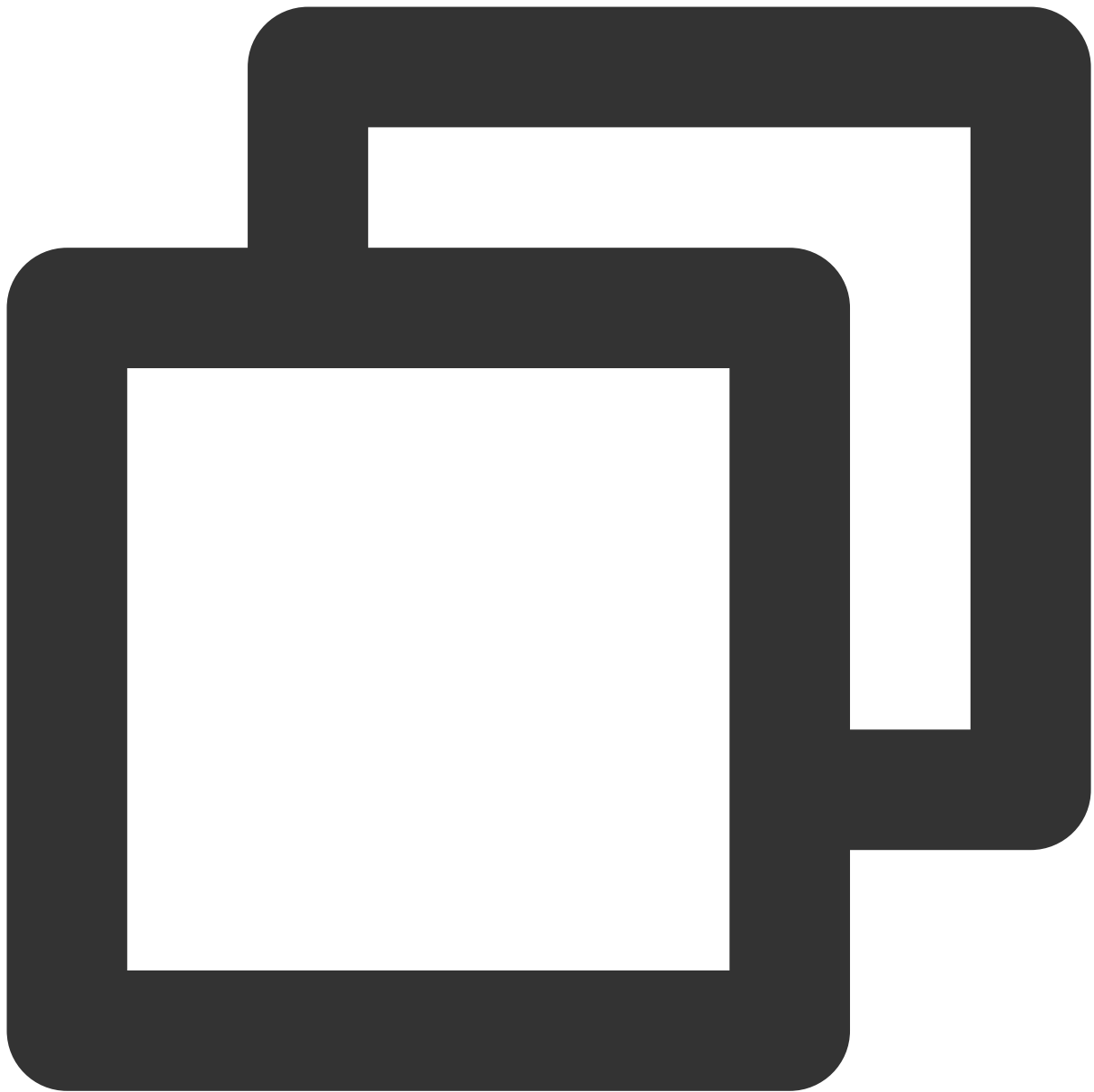
## Directions

### Step 1. Install the Java dependent library

Introduce dependencies in a Java project and add the following dependencies to the `pom.xml` file. This document uses a Maven project as an example.

#### Note

The dependency version must be v4.9.3 or later, preferably v4.9.4.



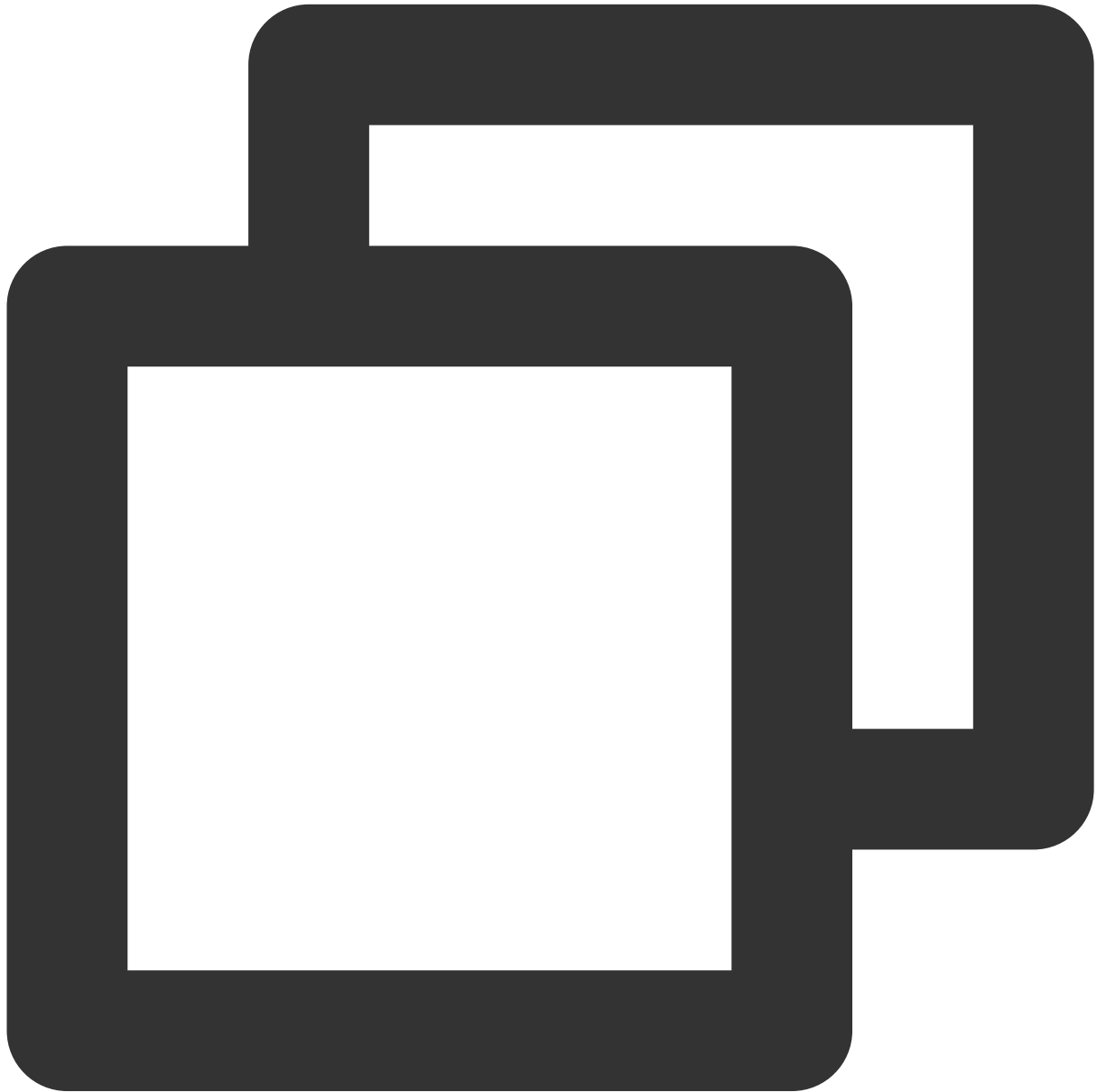
```
<!-- in your <dependencies> block -->
<dependency>
  <groupId>org.apache.rocketmq</groupId>
  <artifactId>rocketmq-client</artifactId>
  <version>4.9.4</version>
</dependency>

<dependency>
  <groupId>org.apache.rocketmq</groupId>
  <artifactId>rocketmq-acl</artifactId>
  <version>4.9.4</version>
```

```
</dependency>
```

## Step 2. Produce messages

### Implementing TransactionListener

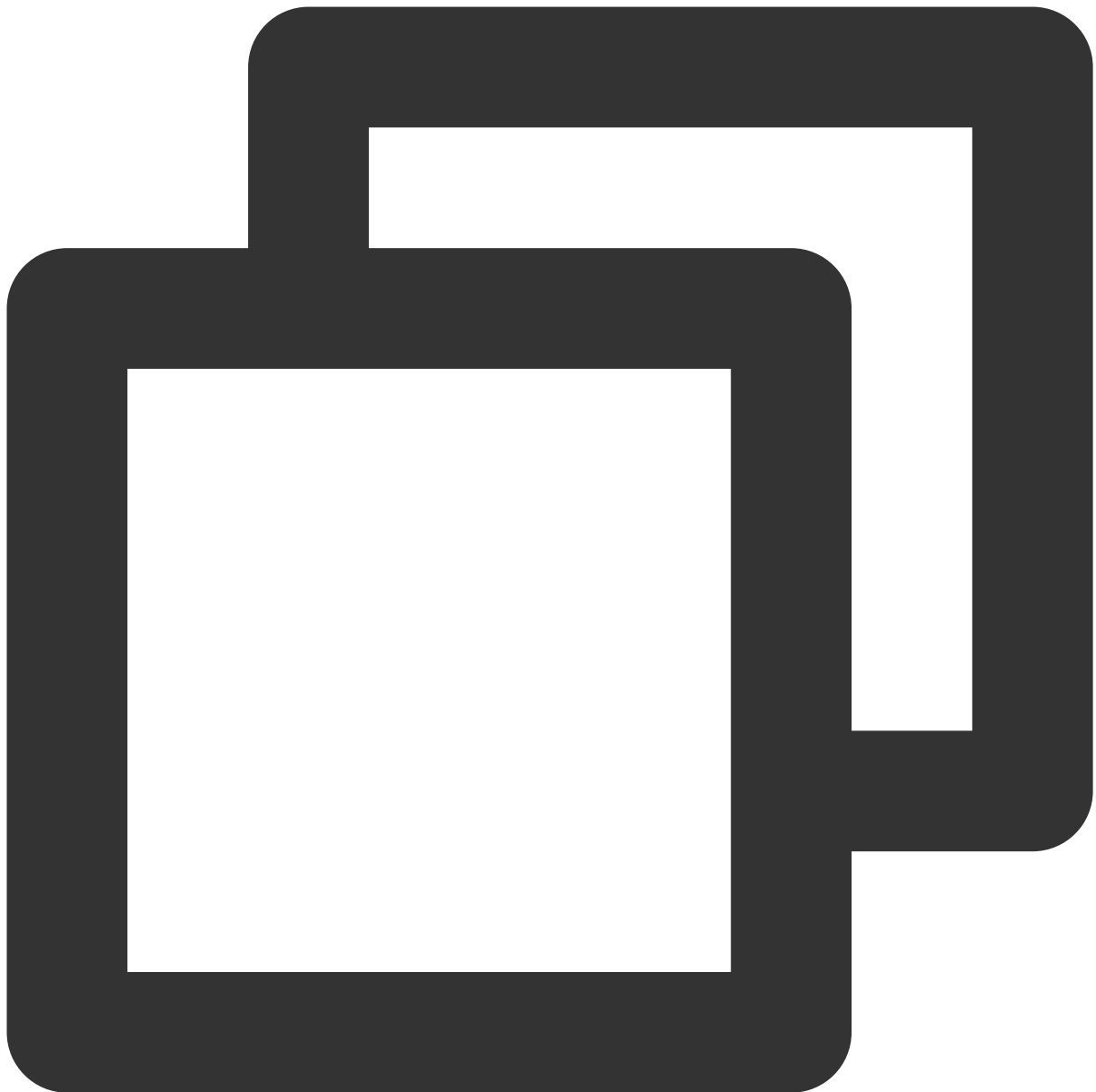


```
public class TransactionListenerImpl implements TransactionListener {  
  
    // After the half message is sent successfully, call back this method to execute  
    @Override  
    public LocalTransactionState executeLocalTransaction(Message msg, Object arg) {
```

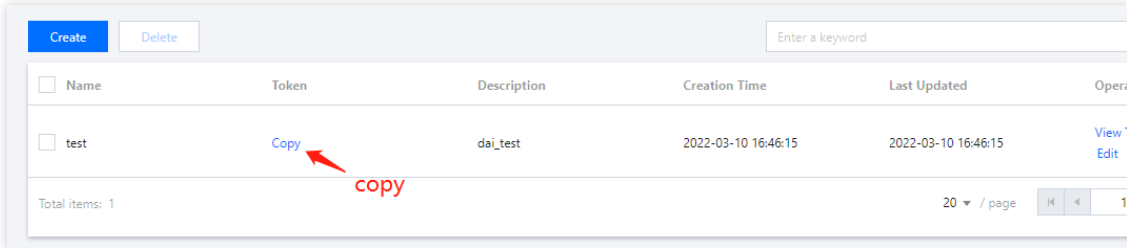
```
        // Execute the database transaction here. If the execution is successful, it
        return LocalTransactionState.UNKNOW;
    }
    // Check back local transaction
    @Override
    public LocalTransactionState checkLocalTransaction(MessageExt msg) {
        // Here query the data status of the local database, and then decide whether
        return LocalTransactionState.COMMIT_MESSAGE;
    }
}
```

## Creating a message producer

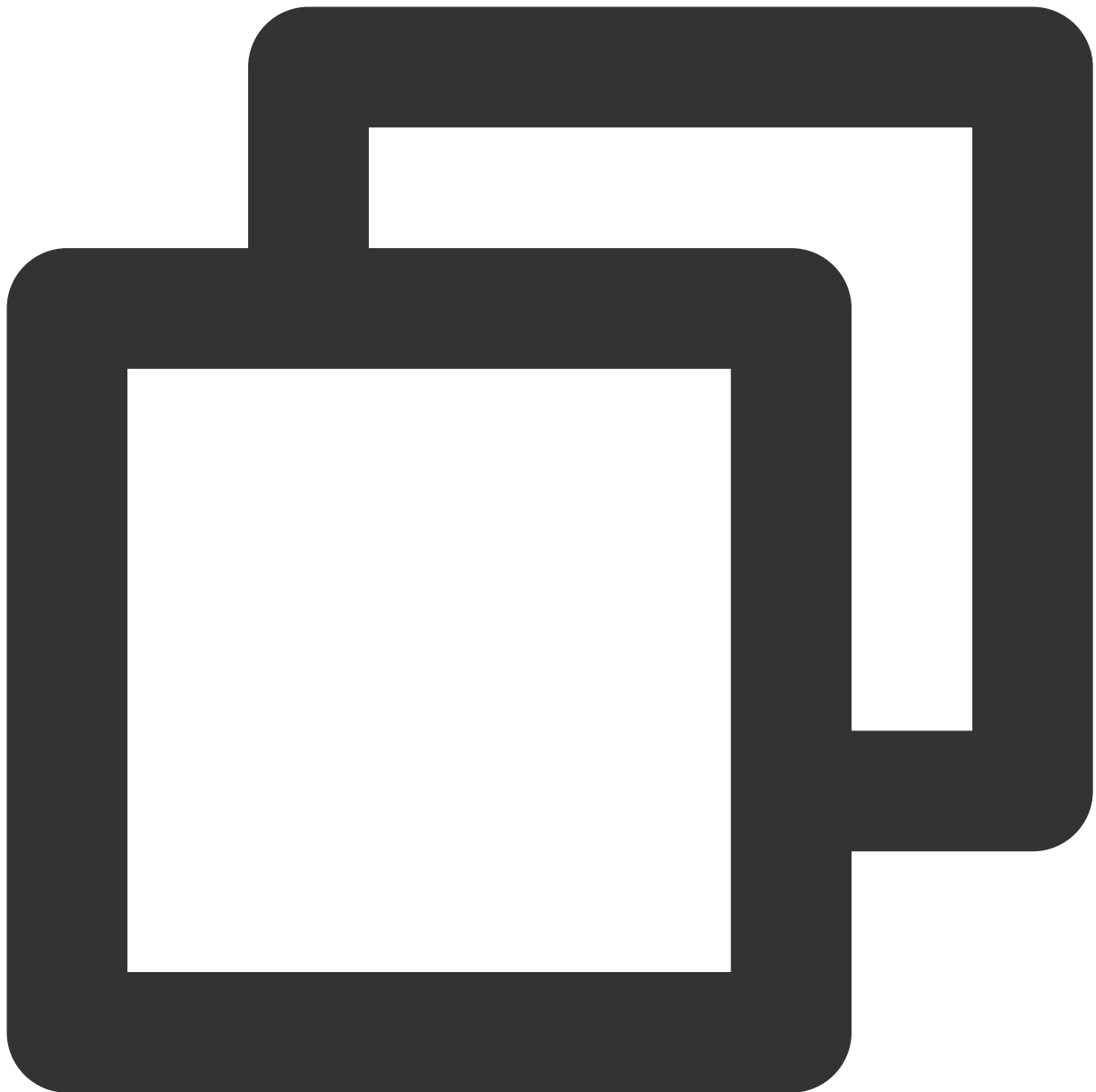




```
//Users need to implement a TransactionListener instance,  
TransactionListener transactionListener = new TransactionListenerImpl();  
// Instantiate a transactional message producer  
ProducerTransactionMQProducer producer = new TransactionMQProducer("transaction_gro  
// ACL permission  
new AclClientRPCHook(new SessionCredentials(ClientCreator.ACCESS_KEY, ClientCreator  
// Set the Nameserver address  
producer.setNamesrvAddr(ClientCreator.NAMESERVER);  
producer.setTransactionListener(transactionListener);  
producer.start();
```

Parameter	Description
groupName	Producer group name. It is recommended to use the corresponding topic name.
nameserver	Cluster access address, which can be obtained from <b>Access Address</b> in the <b>Operation</b> column or <b>Management</b> page in the console. Namespace access addresses in new virtual or exclusive clusters from the <b>Namespace</b> list.
secretKey	Role name, which can be copied on the <a href="#">Role Management</a> page.
accessKey	Role token, which can be copied in the <b>Token</b> column on the <a href="#">Role Management</a> page. 

## Sending a message

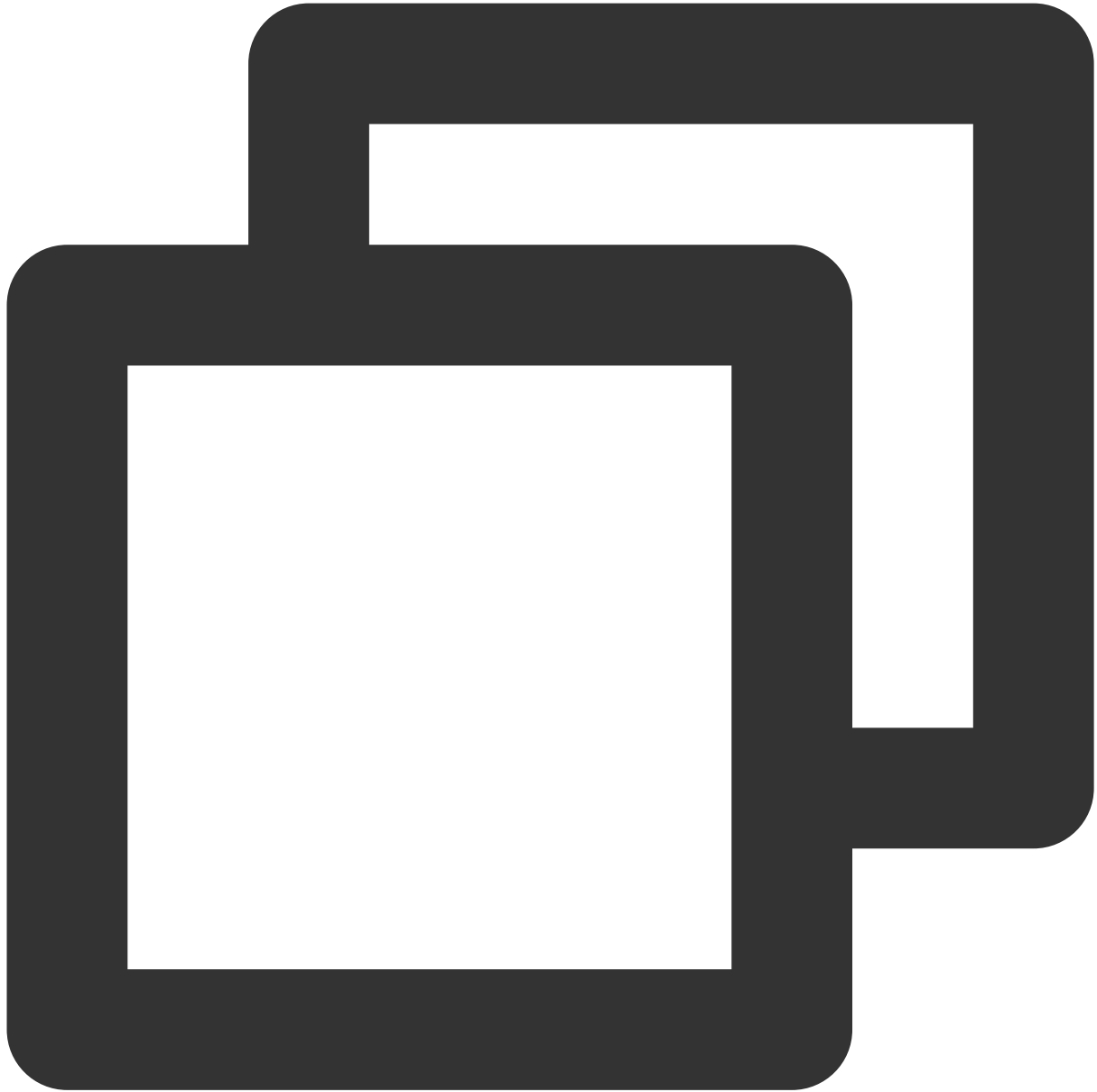


```
for (int i = 0; i < 3; i++) {  
    // Construct message instance  
    Message msg = new Message(TOPIC_NAME, "your tag", "KEY" + i, ("Hello RocketMQ "  
    SendResult sendResult = producer.sendMessageInTransaction(msg, null);  
    System.out.printf("%s%n", sendResult);  
}
```

### Step 3. Consume messages

#### ####Creating a consumer

TDMQ for RocketMQ supports two consumption modes: push and pull. Push mode is recommended.



```
// Instantiate the consumer
DefaultMQPushConsumer pushConsumer = new DefaultMQPushConsumer(
    groupName,
    new AclClientRPCHook(new SessionCredentials(accessKey, secretKey))); //
// Set the Nameserver address
pushConsumer.setNamesrvAddr(nameserver);
pushConsumer.registerMessageListener((MessageListenerConcurrently) (msgs, c
    // Message processing logic
    System.out.printf("%s Receive transaction messages: %s %n", Thread.curre
```

```
// Mark that the message has been successfully consumed
return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
});
```

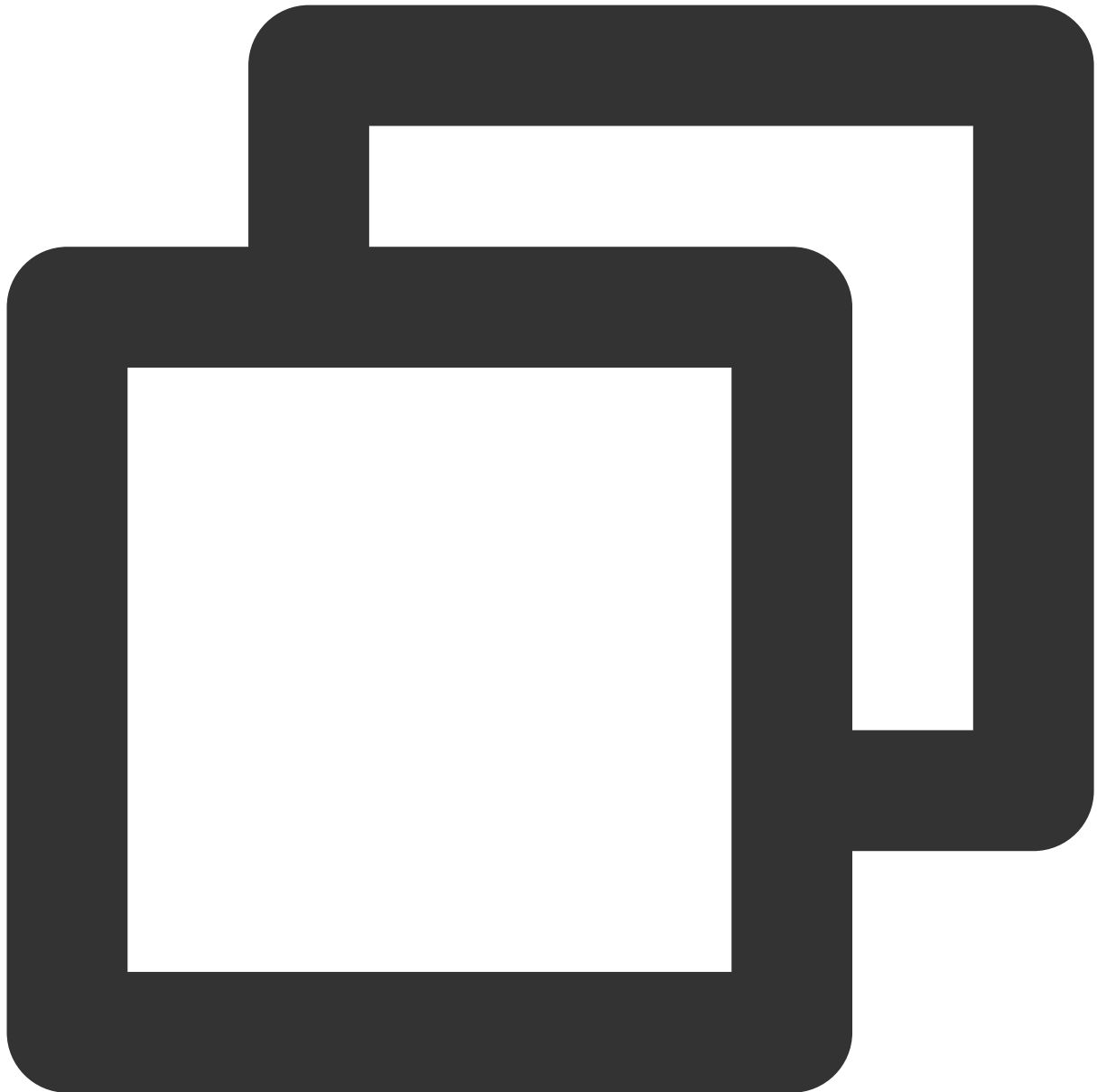
Parameter	Description
groupName	Producer group name, which can be copied under the <b>Group</b> tab on the <b>Cluster</b> page in the console
nameserver	Cluster access address, which can be obtained from <b>Access Address</b> in the <b>Operation</b> column or <b>Management</b> page in the console. Namespace access addresses in new virtual or exclusive clusters are in the <b>Namespace</b> list.
secretKey	Role name, which can be copied on the <a href="#">Role Management</a> page.
accessKey	Role token, which can be copied in the <b>Token</b> column on the <a href="#">Role Management</a> page.

<input type="checkbox"/>	Name	Token	Description	Creation Time	Last Updated	Operations
<input type="checkbox"/>	test	Copy	dai_test	2022-03-10 16:46:15	2022-03-10 16:46:15	View Edit

Total items: 1      20 / page

## Subscribing to messages

The subscription modes vary by consumption mode.



```
// Subscribe to a topic
pushConsumer.subscribe(topic_name, "*");
// Register a callback implementation class to process messages pulled from t
pushConsumer.registerMessageListener((MessageListenerConcurrently) (msgs, con
    // Message processing logic
    System.out.printf("%s Receive New Messages: %s %n", Thread.currentThread(
    // Mark the message as being successfully consumed and return the consump
    return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
});
// Start the consumer instance
pushConsumer.start();
```

### Step 4. View consumption details

Log in to the [TDMQ console](#), go to the **Cluster > Group** page, and view the list of clients connected to the group. Click **View Details** in the **Operation** column to view consumer details.

**Basic Info**

Group Name	group-364733	Creation Time	2022-03-11 15:13:15
Consumption Mode	Unknown	Client Protocol	TCP
Total Heaped Messages	0	Consumer Type	Unknown

---

**Client Address**    Subscription

Client Address	Client Language	Client Version	Message Heap ↕
No data yet			

Total items: 0

Basic Info
Namespace
Topic
Group

Current Namespace: sdaa    Message Retention Period: 3 days    Max TPS ⓘ: 4000

Create (2/1500)
Search by keyword

Group Name	Consumer Info ↕	Consumption Mode	Descriptive
group-364733	Online Consumer    0    TPS 0    Total Heap 0    ↻	Unknown	
dasda	Online Consumer    0    TPS 0    Total Heap 0    ↻	Unknown	

Total items: 2

**Note**

Above is a brief introduction to message publishing and subscription. For more information, see [Demo](#) or [RocketMQ documentation](#).



# Sending and Receiving Filtered Messages

Last updated : 2023-03-28 10:15:45

## Overview

This document describes how to use open-source SDK to send and receive filtered messages by using the SDK for Java as an example. You can do so with tags or SQL expressions.

## Prerequisites

You have created the required resources. If it is a globally sequential message, you need to create a single-queue topic. For more information, see [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 here](#) or have downloaded one at the [GitHub project](#).

You have learned about the sending and receiving processes of general messages.

## Tag-based option

The main code of creating producer and consumer is basically same as that for general messages.

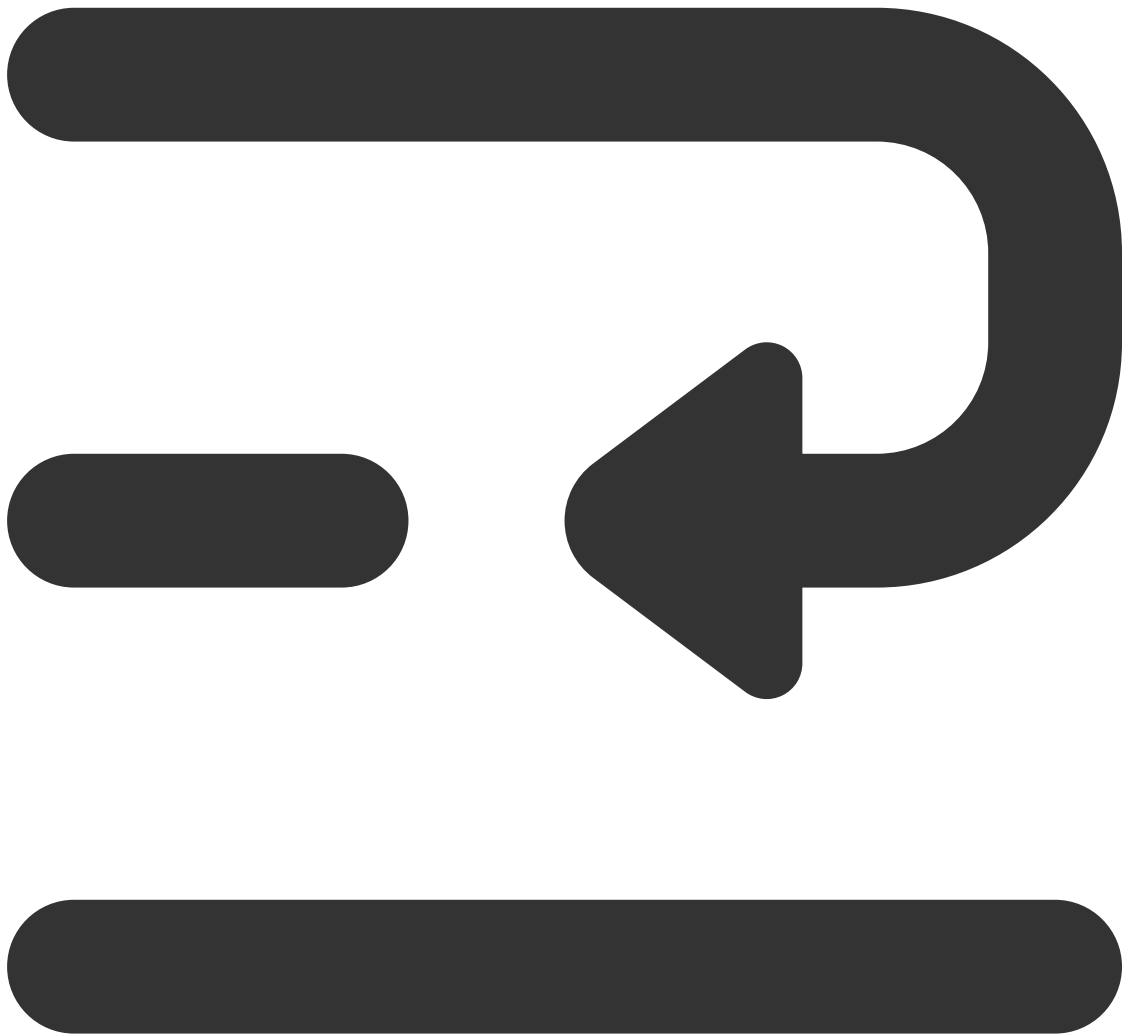
For message production, a message need to be carried with a or more tags when constructing the message body.

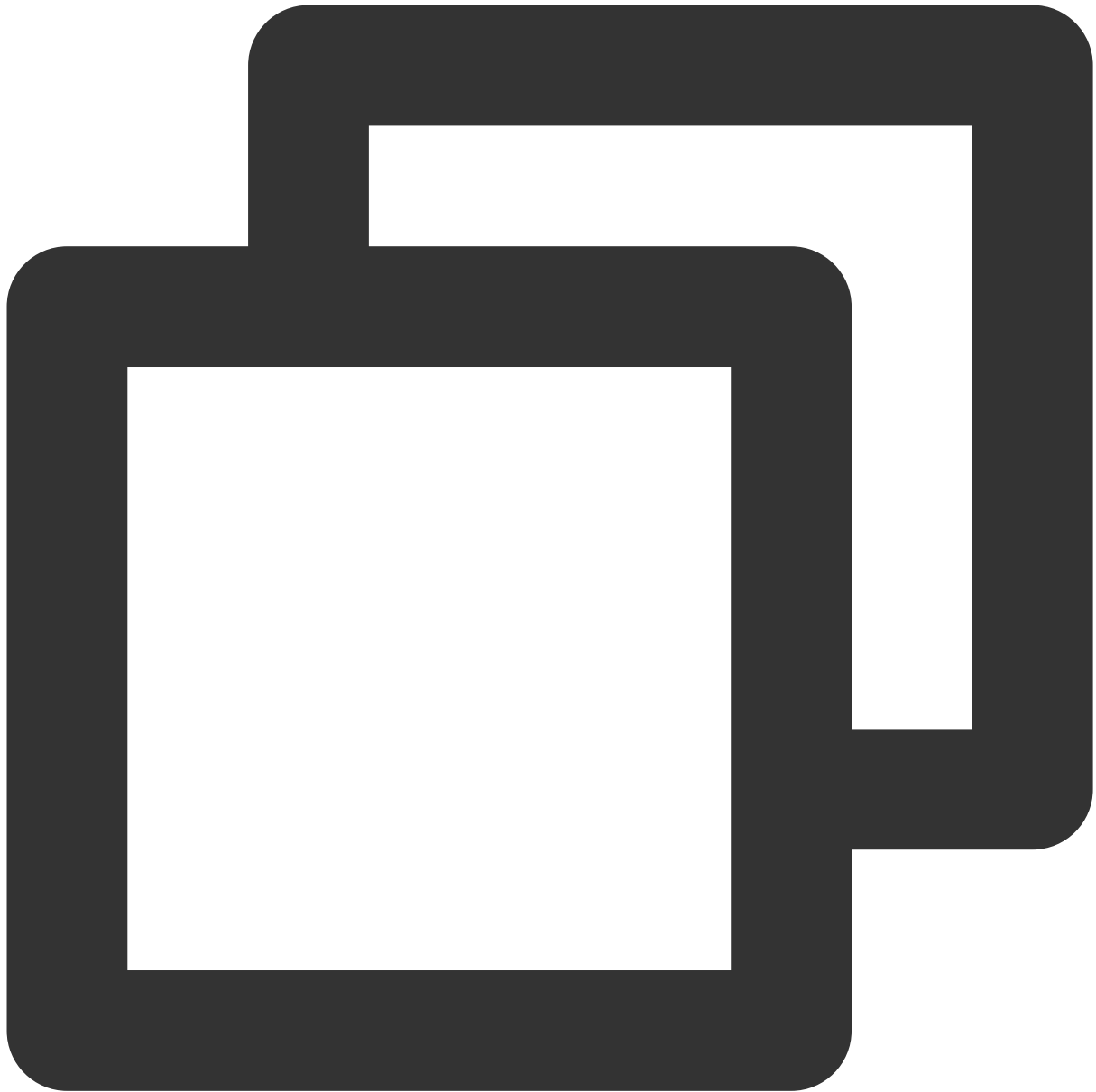
For message consumption, a message need to be carried with a tag, an asterisk (\*), or multiple tag expressions when being subscribed to.

### Step 1. Produce messages

#### Sending messages

The main code of sending messages is basically same as that for general messages. However, a message is allowed to carry only a tag when constructing the message body.

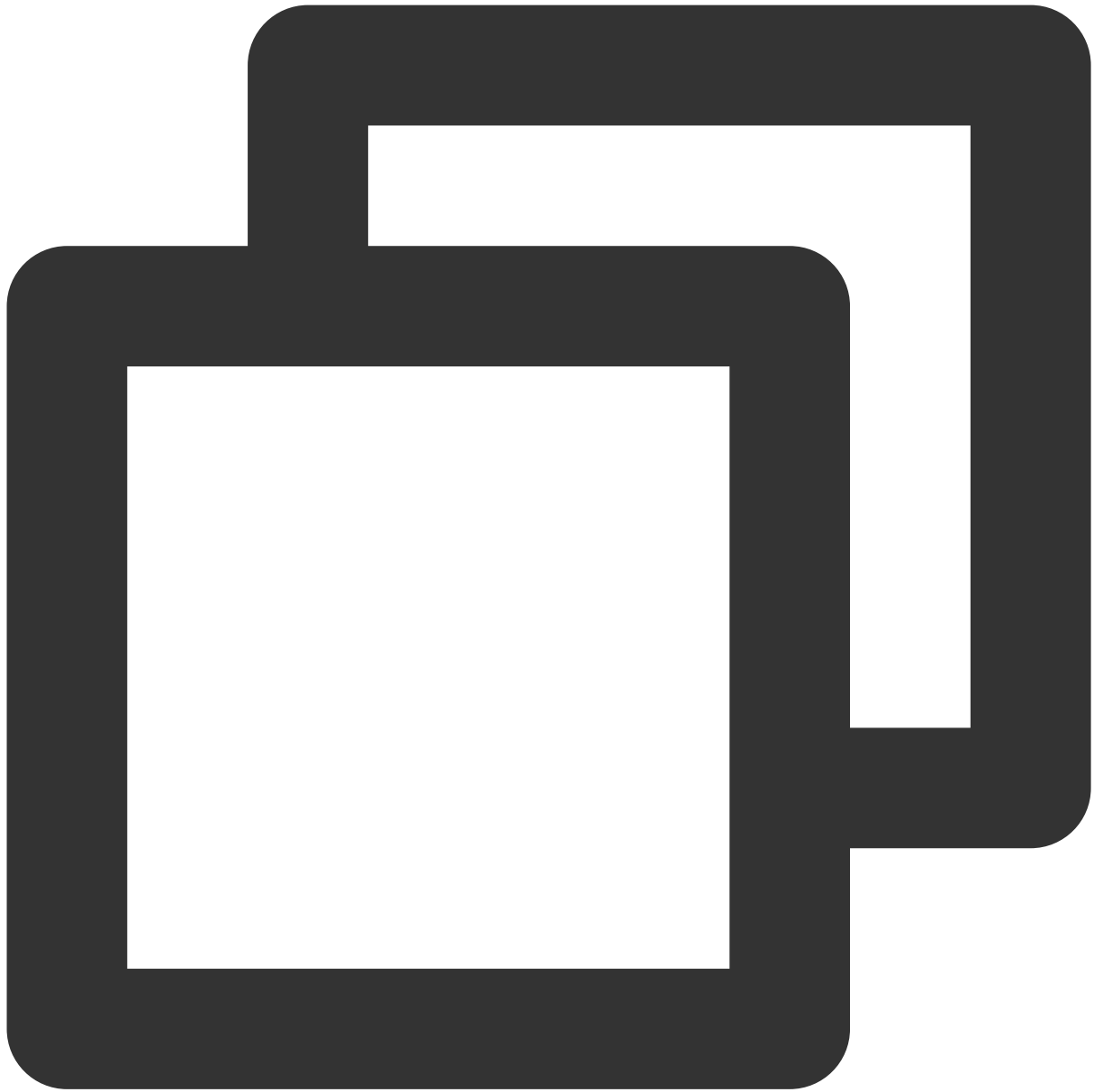




```
int totalMessagesToSend = 5;
for (int i = 0; i < totalMessagesToSend; i++) {
    Message msg = new Message(TOPIC_NAME, "Tag1", "Hello RocketMQ.".getBytes(StandardCharsets.UTF_8));
    // Send the message
    SendResult sendResult = producer.send(message);
    System.out.println("sendResult = " + sendResult);
}
```

## Step 2. Consume messages

## Subscribing to messages



```
// Subscribe to all tags when subscribing to a topic
pushConsumer.subscribe(topic_name, "*");

//Subscribe to the specified tags
//pushConsumer.subscribe(TOPIC_NAME, "Tag1");

// Subscribe to multiple tags
//pushConsumer.subscribe(TOPIC_NAME, "Tag1||Tag2");
```

```
// Register a callback implementation class to process messages pulled from the broker
pushConsumer.registerMessageListener((MessageListenerConcurrently) (msgs, context)
    // Message processing logic
    System.out.printf("%s Receive New Messages: %s %n", Thread.currentThread().getName(), msgs);
    // Mark the message as being successfully consumed and return the consumption status
    return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
});
// Start the consumer instance
pushConsumer.start();
```

Parameter	Description
topic_name	Topic name, which can be copied under the <b>Topic</b> tab on the <b>Cluster</b> page in the console.
sql	If the subscription expression is left empty or specified as asterisk (*), all messages are subscribed to. <code>tag1    tag2    tag3</code> means subscribing to multiple types of tags.

### Note

Above is a brief introduction to message publishing and subscription. For more information, see [GitHub Demo](#) or [official RocketMQ documentation](#).

## SQL expression-based option

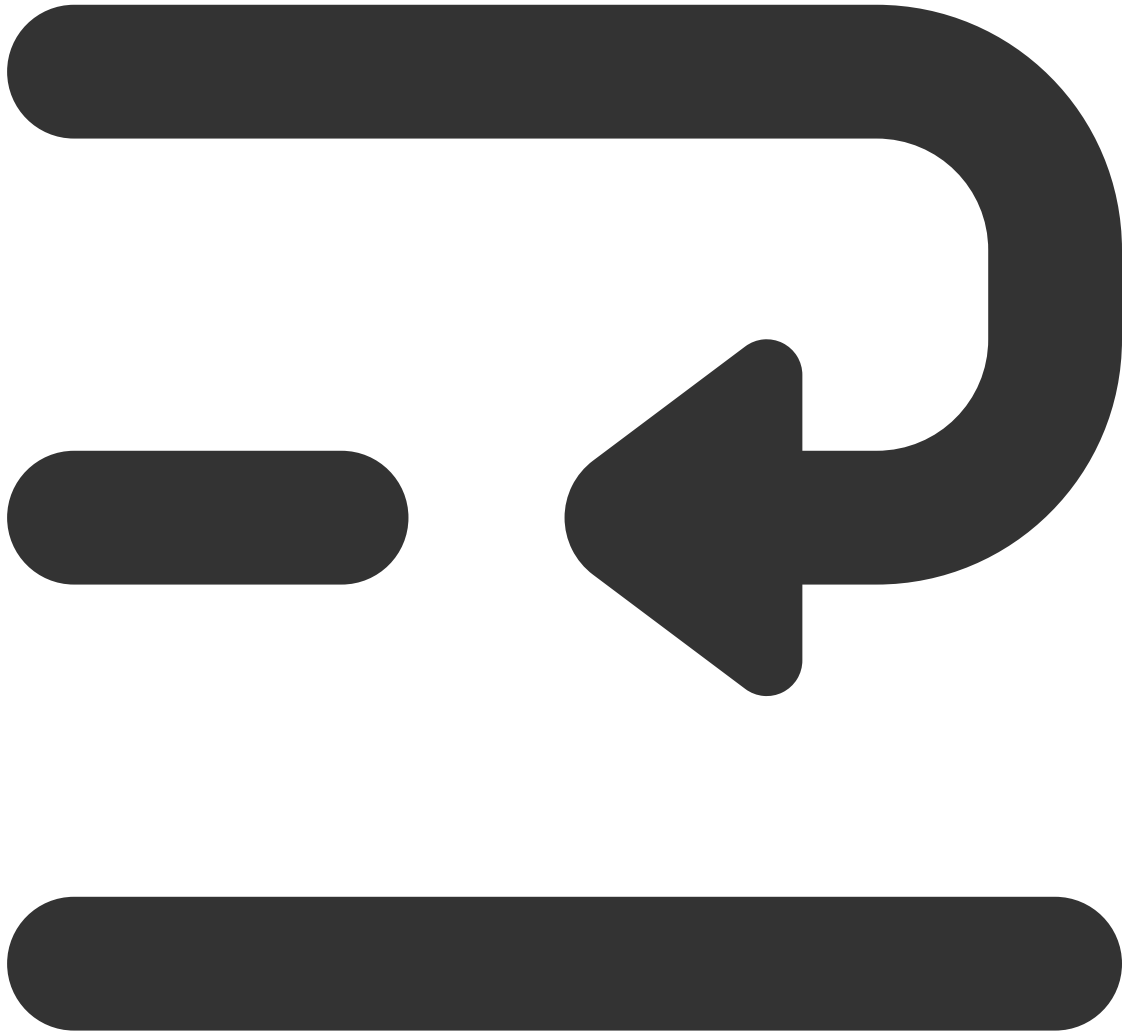
The main code of creating producer and consumer is basically same as that for general messages.

For message production, a message need to be carried with user-defined properties when constructing the message body.

For message consumption, a message need to be carried with corresponding SQL expression when being subscribed to.

### Step 1. Produce messages

The main code of sending messages is basically same as that for general messages. However, a message is allowed to carry multiple user-defined properties when constructing the message body.

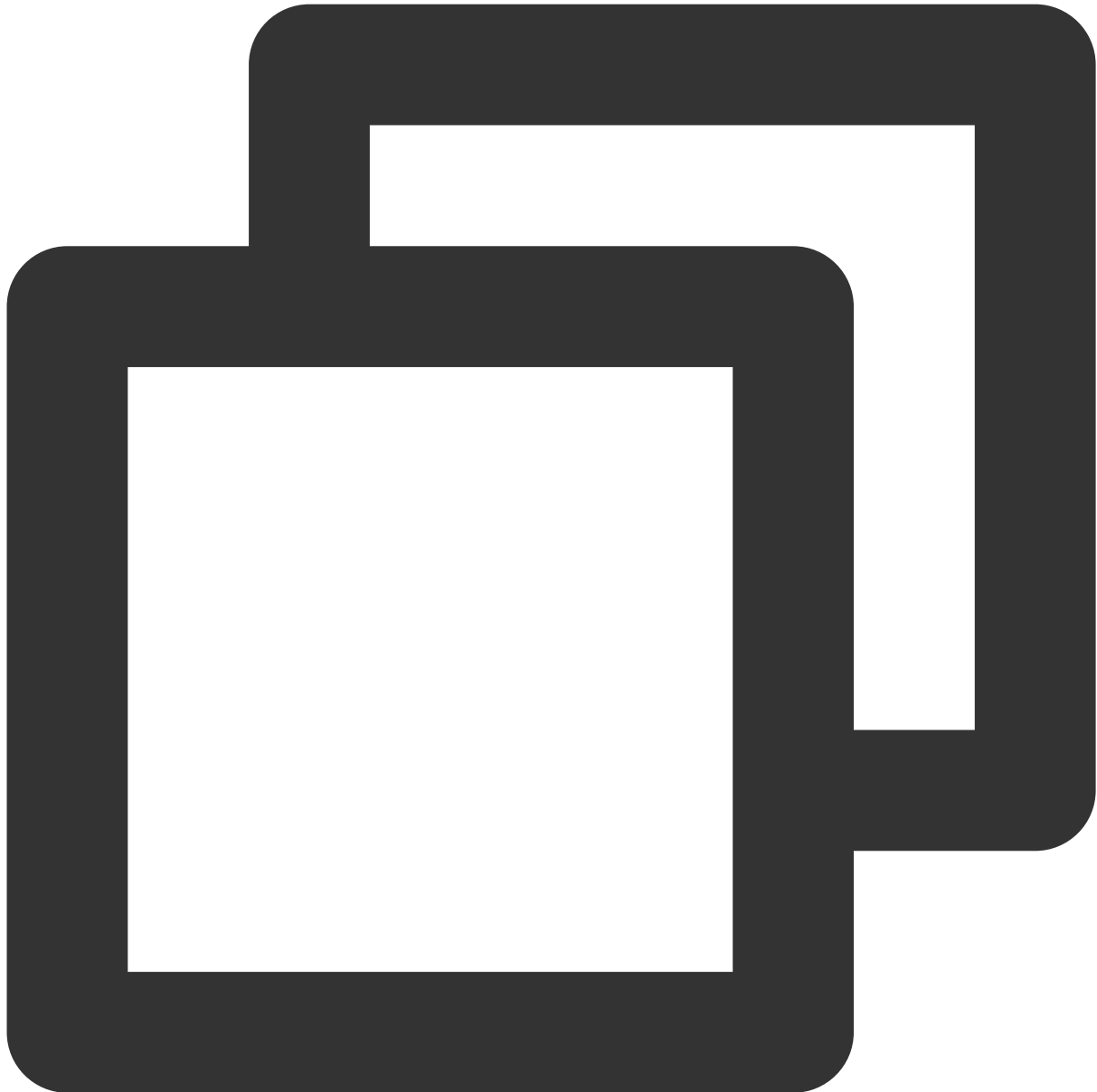




```
int totalMessagesToSend = 5;
for (int i = 0; i < totalMessagesToSend; i++) {
    Message msg = new Message(TOPIC_NAME, "Hello RocketMQ.".getBytes(StandardCharset
    msg.putUserProperty("key1", "value1");
    // Send the message
    SendResult sendResult = producer.send(message);
    System.out.println("sendResult = " + sendResult);
}
```

## Step 2. Consume messages

The main code of consuming messages is basically same as that for general messages. However, a message need to be carried with corresponding SQL expression when being subscribed to.



```
pushConsumer.subscribe(TOPIC_NAME, MessageSelector.bySql("True"));

// Subscribe to single-key SQL expression when subscribing to a topic
//pushConsumer.subscribe(TOPIC_NAME,    MessageSelector.bySql("key1 IS NOT NULL AND

//Subscribe to multiple properties
//pushConsumer.subscribe(TOPIC_NAME,    MessageSelector.bySql("key1 IS NOT NULL AND
```



```
// Register a callback implementation class to process messages pulled from the broker
pushConsumer.registerMessageListener((MessageListenerConcurrently) (msgs, context)
    // Message processing logic
    System.out.printf("%s Receive New Messages: %s %n", Thread.currentThread().getName(), msgs);
    // Mark the message as being successfully consumed and return the consumption status
    return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
});
// Start the consumer instance
pushConsumer.start();
```

**Note**

Above is a brief introduction to message publishing and subscription. For more information, see [GitHub Demo](#) or [official RocketMQ documentation](#).

# Sending and Receiving Broadcast Messages

Last updated : 2023-05-16 11:07:52

## Overview

This document describes how to use open-source SDK to send and receive broadcast messages by using the SDK for Java as an example.

## 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 here](#) or have downloaded one at the [GitHub project](#).

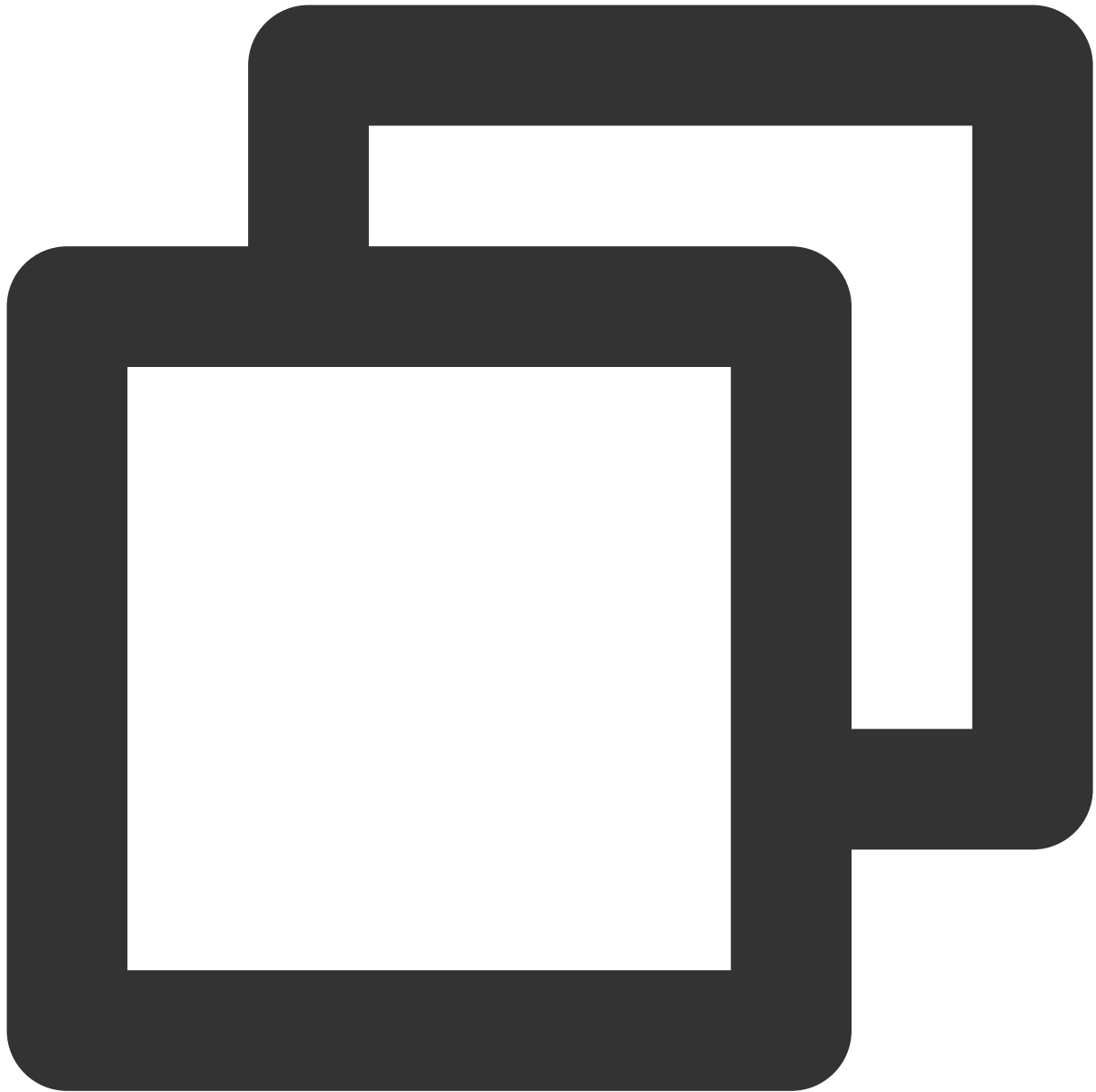
## Directions

### Step 1. Install the Java dependent library

Introduce dependencies in a Java project and add the following dependencies to the `pom.xml` file. This document uses a Maven project as an example.

#### Note

The dependency version must be v4.9.3 or later, preferably v4.9.4.



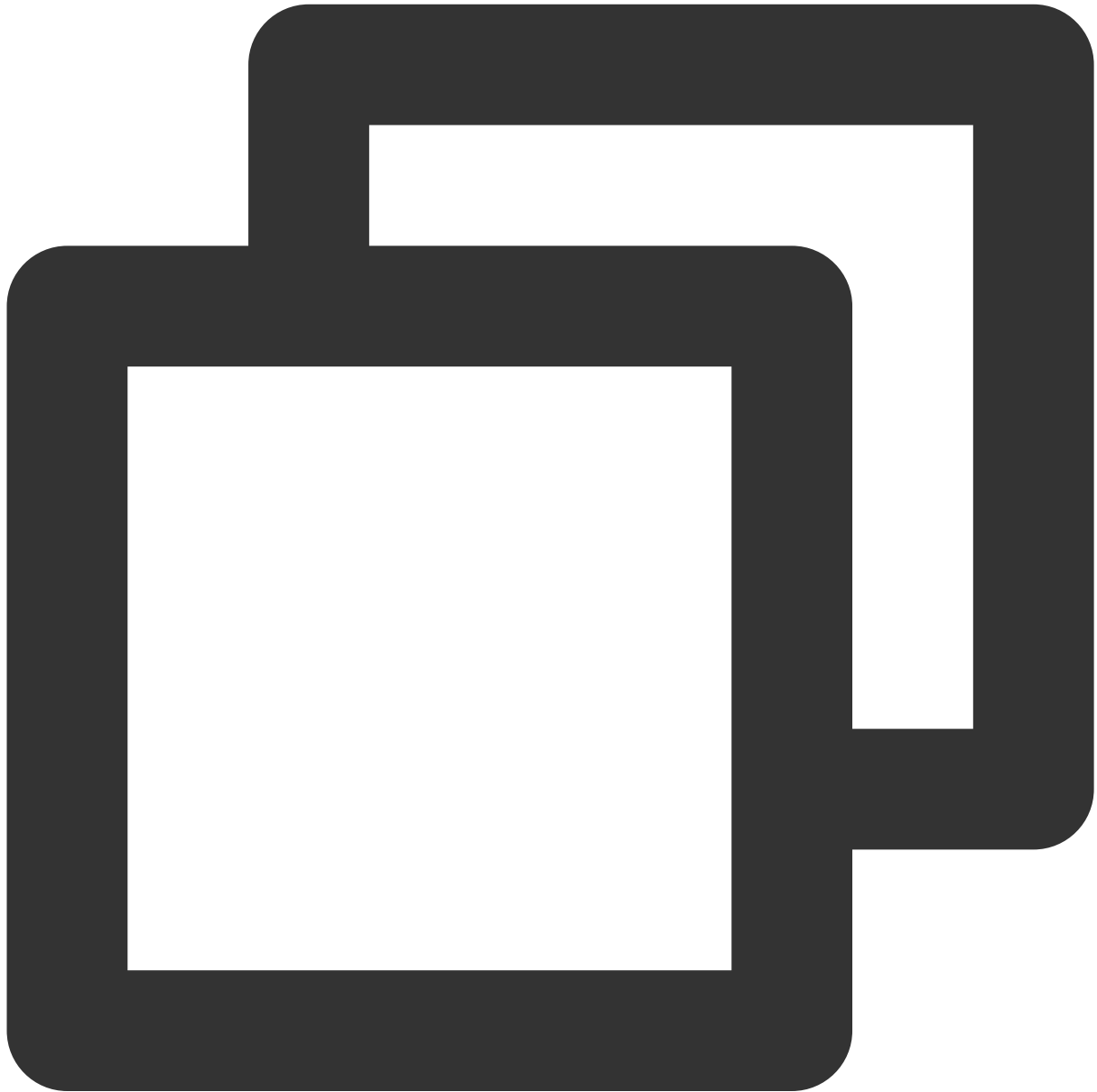
```
<!-- in your <dependencies> block -->
<dependency>
  <groupId>org.apache.rocketmq</groupId>
  <artifactId>rocketmq-client</artifactId>
  <version>4.9.4</version>
</dependency>

<dependency>
  <groupId>org.apache.rocketmq</groupId>
  <artifactId>rocketmq-acl</artifactId>
  <version>4.9.4</version>
```

```
</dependency>
```

## Step 2. Produce messages

### Creating a message producer



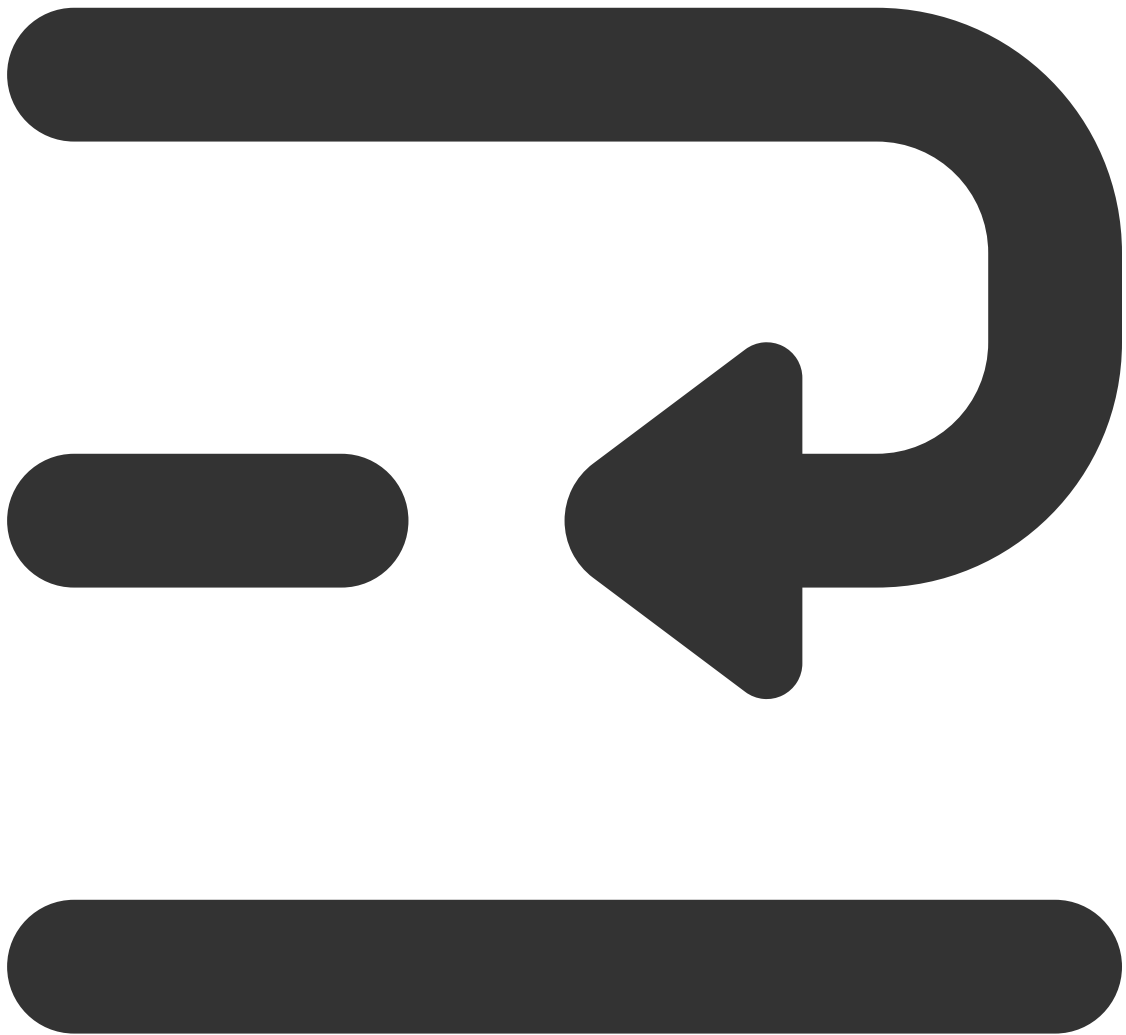
```
// Instantiate the message producer
DefaultMQProducer producer = new DefaultMQProducer(
    groupName,
    new AclClientRPCHook(new SessionCredentials(accessKey, secretKey)) // ACL pe
);
```

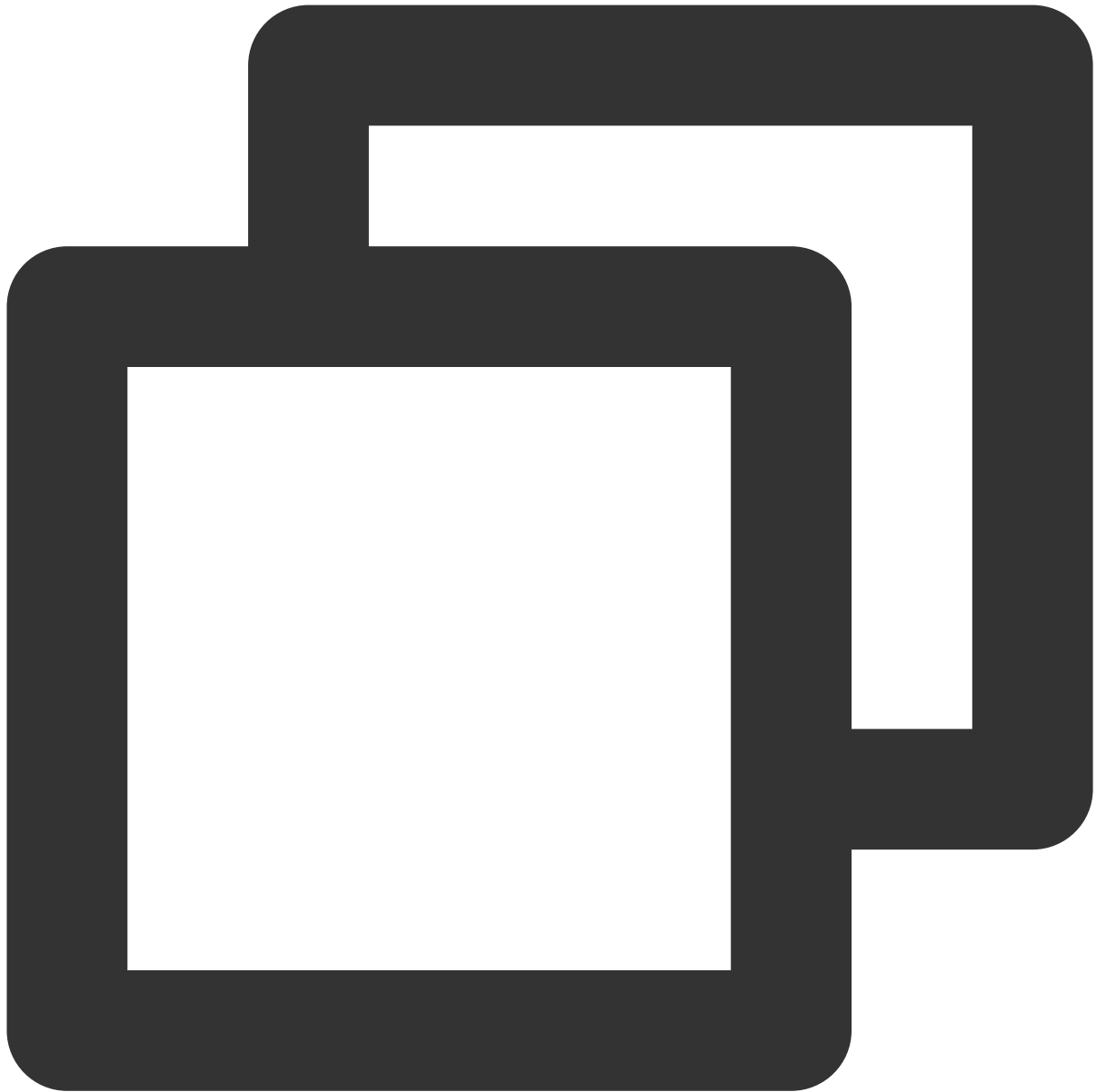
```
// Set the Nameserver address
producer.setNamesrvAddr(nameserver);
// Start the producer instance
producer.start();
```

Parameter	Description												
groupName	Producer group name. It is recommended to use the corresponding topic name.												
nameserver	Cluster access address, which can be obtained from <b>Access Address</b> in the <b>Operation</b> column or <b>Management</b> page in the console. Namespace access addresses in new virtual or exclusive clusters are in the <b>Namespace</b> list.												
secretKey	Role name, which can be copied on the <a href="#">Role Management</a> page.												
accessKey	Role token, which can be copied in the <b>Token</b> column on the <a href="#">Role Management</a> page. <div data-bbox="384 864 1517 1111" data-label="Image"> <p>The screenshot shows a table with the following data:</p> <table border="1"> <thead> <tr> <th>Name</th> <th>Token</th> <th>Description</th> <th>Creation Time</th> <th>Last Updated</th> <th>Oper</th> </tr> </thead> <tbody> <tr> <td>test</td> <td>Copy</td> <td>dai_test</td> <td>2022-03-10 16:46:15</td> <td>2022-03-10 16:46:15</td> <td>View Edit</td> </tr> </tbody> </table> <p>Total items: 1</p> <p>20 / page</p> </div>	Name	Token	Description	Creation Time	Last Updated	Oper	test	Copy	dai_test	2022-03-10 16:46:15	2022-03-10 16:46:15	View Edit
Name	Token	Description	Creation Time	Last Updated	Oper								
test	Copy	dai_test	2022-03-10 16:46:15	2022-03-10 16:46:15	View Edit								

## Sending a message

This process is the same as that of general messages. Broadcast messages reflect the behavior of consumers.



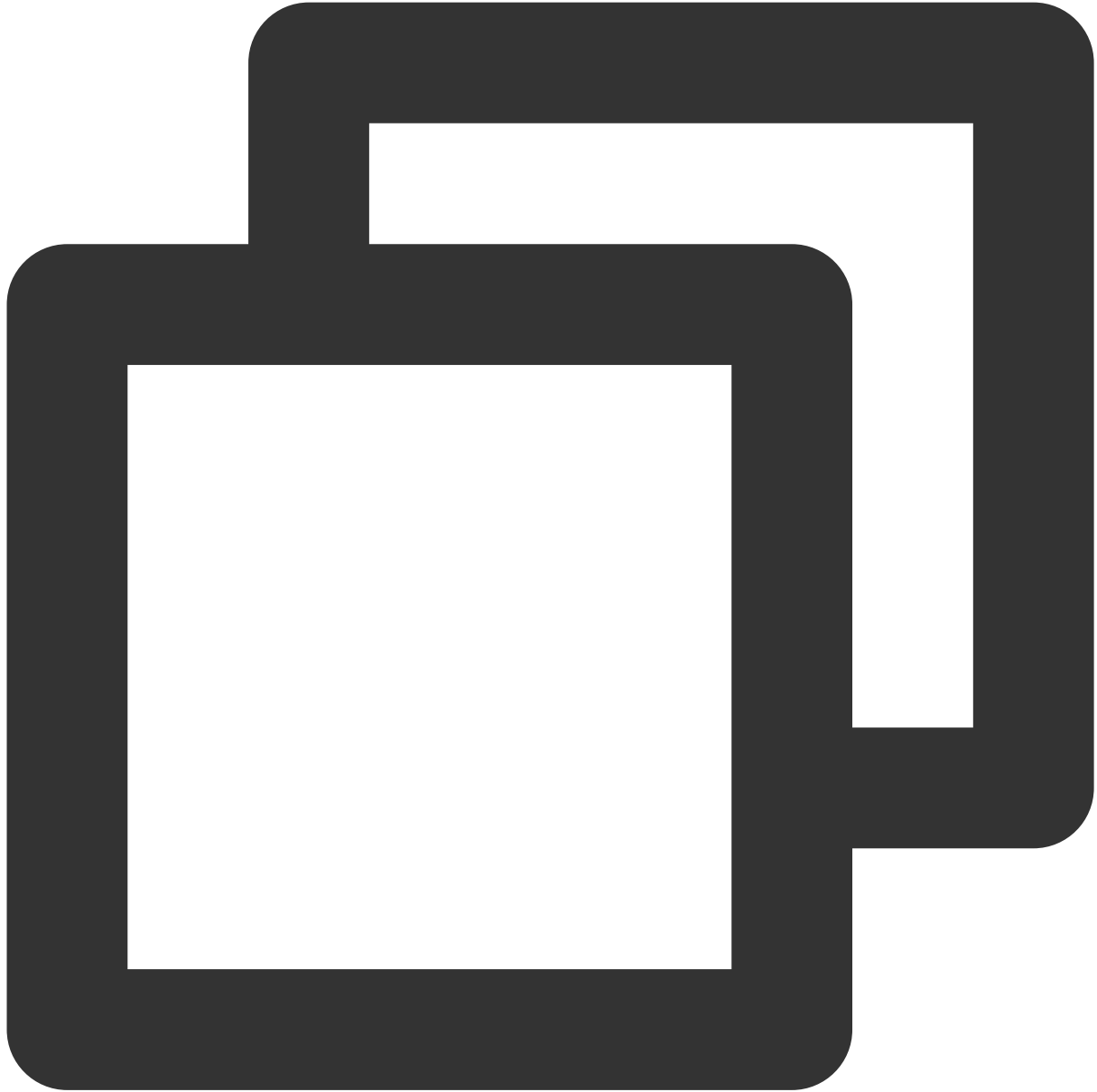


```
int totalMessagesToSend = 5;
for (int i = 0; i < totalMessagesToSend; i++) {
    Message message = new Message(TOPIC_NAME, ("Hello scheduled message " + i).getBytes());
    // Send the message
    SendResult sendResult = producer.send(message);
    System.out.println("sendResult = " + sendResult);
}
```

### Step 3. Consume messages

#### ####Creating a consumer

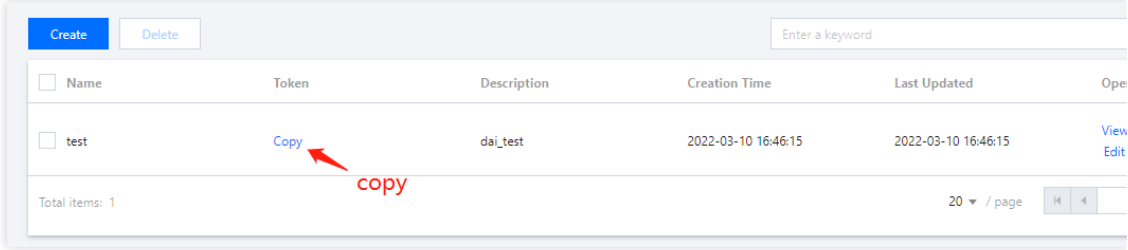
TDMQ for RocketMQ supports two consumption modes: push and pull. Push mode is recommended.



```
// Instantiate the consumer
    DefaultMQPushConsumer pushConsumer = new DefaultMQPushConsumer(
        groupName,
        new AclClientRPCHook(new SessionCredentials(accessKey, secretKey))); //
// Set the Nameserver address
pushConsumer.setNamesrvAddr(nameserver);
```

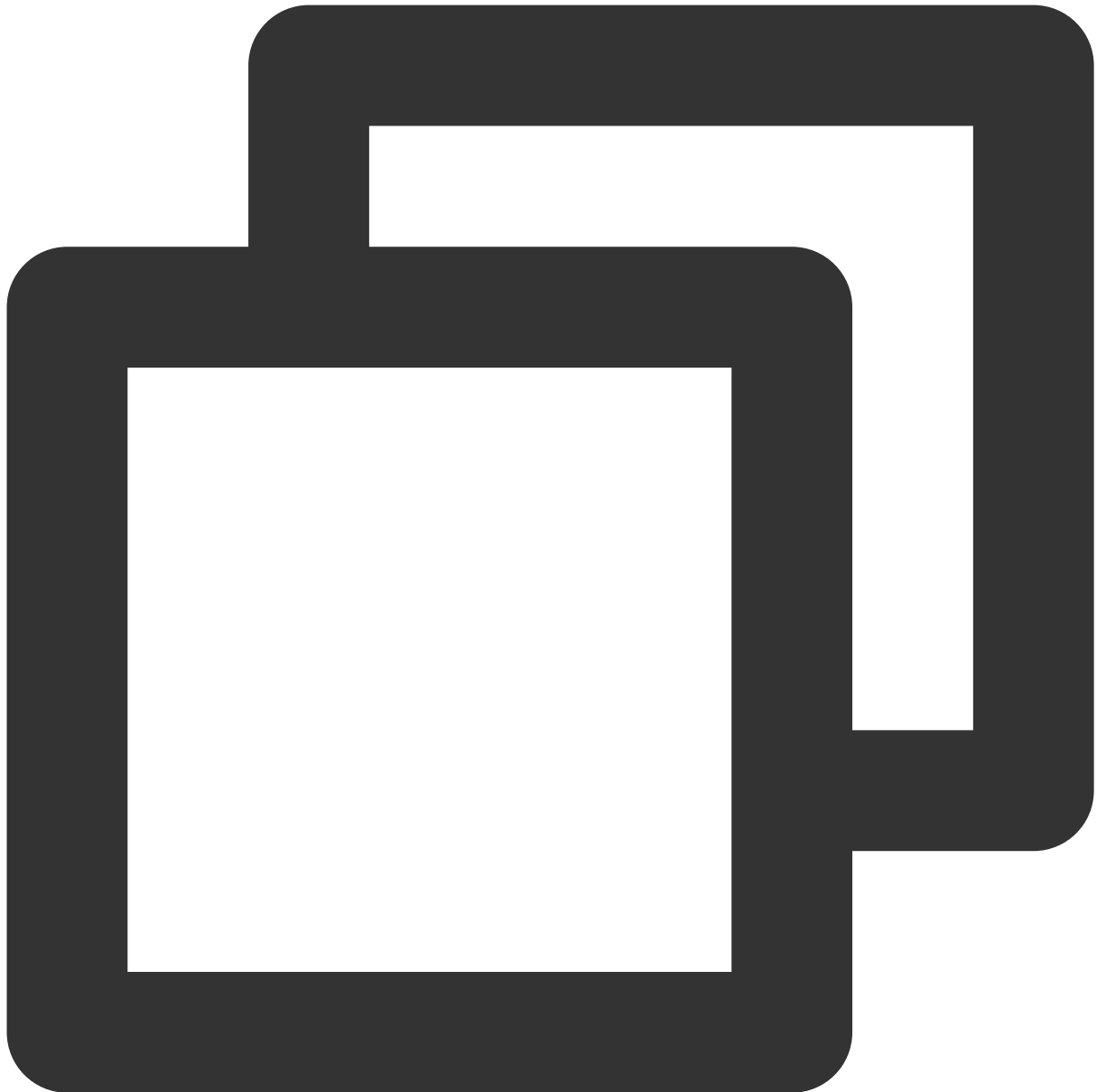
Parameter	Description
-----------	-------------



groupName	Producer group name, which can be copied under the <b>Group</b> tab on the <b>Cluster</b> page in the console
nameserver	Cluster access address, which can be obtained from <b>Access Address</b> in the <b>Operation</b> column or <b>Management</b> page in the console. Namespace access addresses in new virtual or exclusive clusters are in the <b>Namespace</b> list.
secretKey	Role name, which can be copied on the <a href="#">Role Management</a> page.
accessKey	<p>Role token, which can be copied in the <b>Token</b> column on the <a href="#">Role Management</a> page.</p> 

### Subscribing to messages

This process requires setting consumption mode.



```
// Set broadcast consumption mode
pushConsumer.setMessageModel(MessageModel.BROADCASTING);
// Subscribe to a topic
pushConsumer.subscribe(topic_name, "*");
// Register a callback implementation class to process messages pulled from the bro
pushConsumer.registerMessageListener((MessageListenerConcurrently) (msgs, context)
    // Message processing logic
    System.out.printf("%s Receive New Messages: %s %n", Thread.currentThread().getNa
    // Mark the message as being successfully consumed and return the consumption st
    return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
});
```

```
// Start the consumer instance
pushConsumer.start();
```

## Step 4. View consumption details

Log in to the [TDMQ console](#), go to the **Cluster > Group** page, and view the list of clients connected to the group.

Click **View Details** in the **Operation** column to view consumer details.

### Basic Info

Group Name	group-364733	Creation Time	2022-03-11 15:13:15
Consumption Mode	Unknown	Client Protocol	TCP
Total Heaped Messages	0	Consumer Type	Unknown

### Client Address

Subscription

Client Address	Client Language	Client Version	Message Heap ↕
No data yet			

Total items: 0

Basic Info
Namespace
Topic
Group

Current Namespace

sdaa ▼

Message Retention Period
3 days
Max TPS ⓘ
4000

Create (2/1500)

Search by keyword

Group Name	Consumer Info ↕	Consumption Mode	Descriptio
group-364733	Online Consumer 0 TPS 0 Total Heap 0 ↻	Unknown	
dasda	Online Consumer 0 TPS 0 Total Heap 0 ↻	Unknown	

Total items: 2

**Note**  
 Above is a brief introduction to message publishing and subscription. For more information, see [Demo](#) or [RocketMQ documentation](#).

# SDK for C++

Last updated : 2023-05-16 11:07:52

## 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.

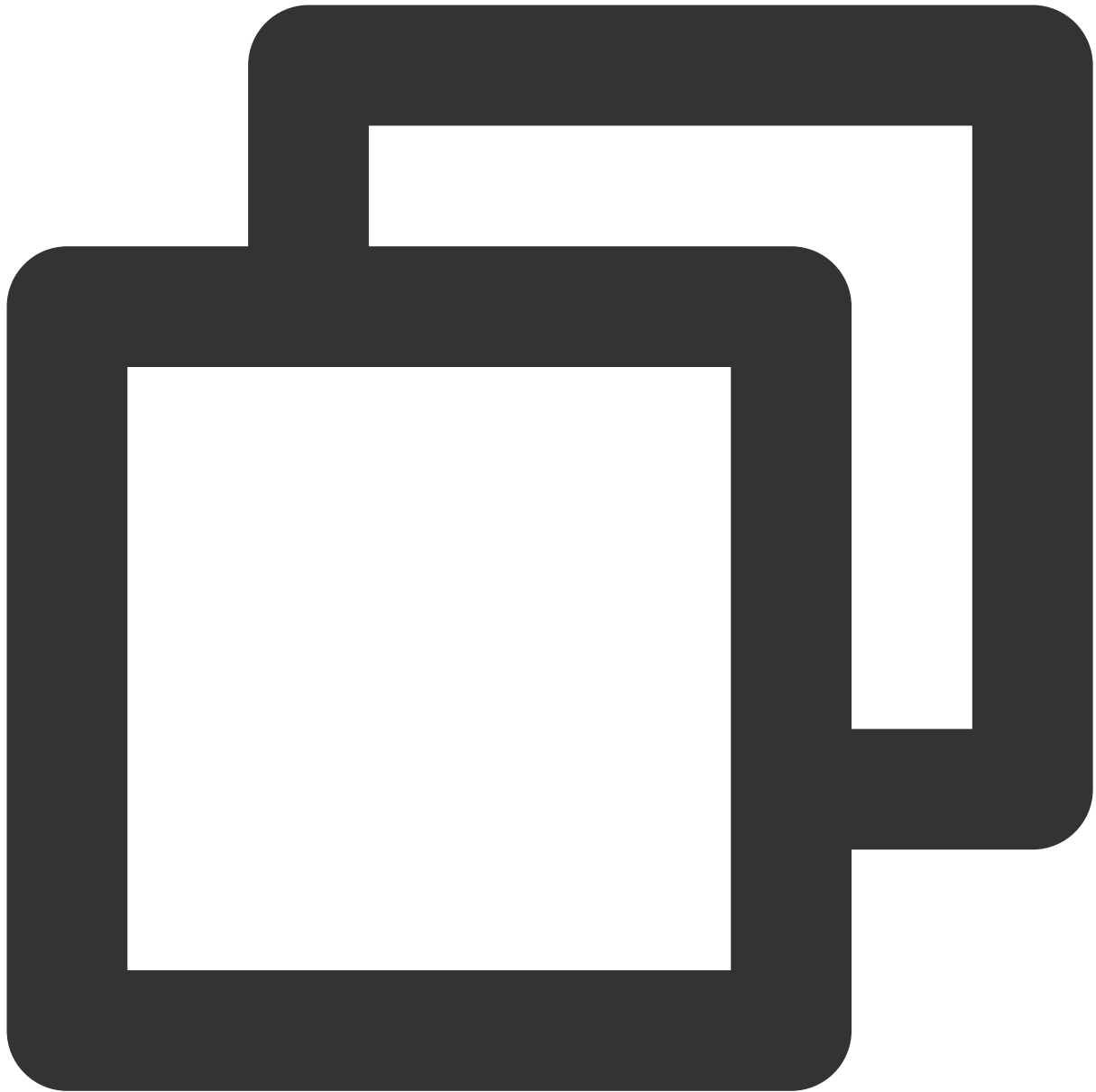
## Prerequisites

You have installed [GCC](#).

[You have downloaded the demo.](#)

## Directions

1. Prepare the environment.
  - 1.1 Install RocketMQ-Client-CPP in the client environment as instructed in the [official documentation](#). **The master branch is recommended.**
  - 1.2 Import the header files and dynamic libraries related to RocketMQ-Client-CPP to the project.
2. Instantiate the message producer.

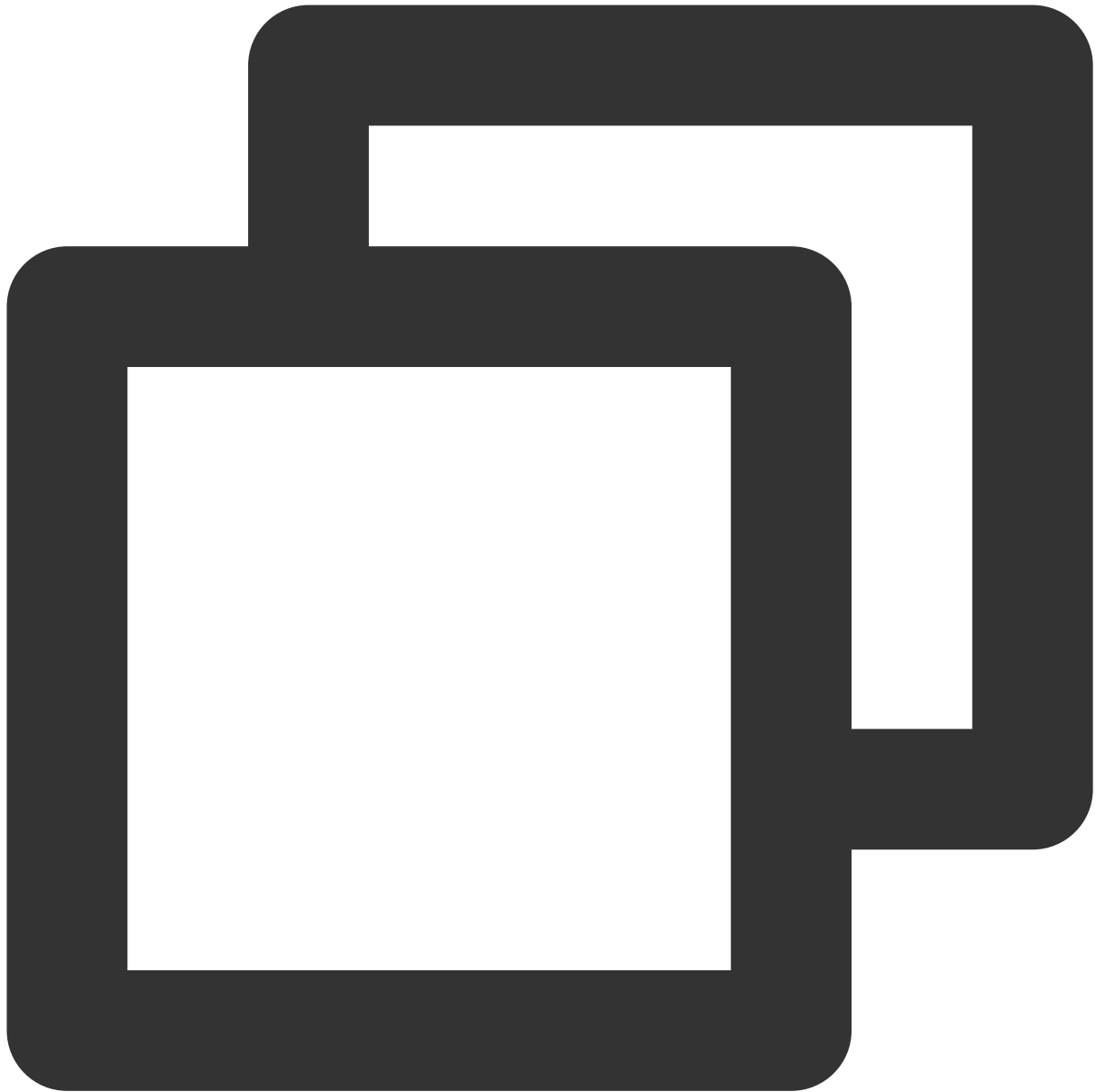


```
// Set the producer group name
DefaultMQProducer producer(groupName);
// Set the service access address
producer.setNamesrvAddr(nameserver);
// Set user permissions
producer.setSessionCredentials(
    accessKey, // Role token
    secretKey, // Role name
    "");
// Set the full namespace name
producer.setNameSpace(namespace);
```

```
// Make sure all parameters are configured before the start
producer.start();
```

Parameter	Description
groupName	Producer group name, which can be copied under the <b>Group</b> tab on the <b>Cluster</b> page in the console.
nameserver	Cluster access address, which can be obtained in the <b>Operation</b> column on the <b>Cluster Manager</b> page. Namespace access addresses in new virtual or exclusive clusters can be copied from the <b>Namespaces</b> page. <div data-bbox="384 589 1520 880" data-label="Image"> <p>The screenshot shows a table with columns: Cluster ID/Name, Topics, Group Count, Resource Tag, and Cluster Description. One cluster is listed with ID 'rocketmq-5...w9qqqr' and name 'test'. To the right, under 'API Call Address', the VPC Access Address is shown as 'rocketmq-5...w9qqqr.rocketmq.ap-gz.tencentcloudmq.com:5098'.</p> </div>
secretKey	Role name, which can be copied on the <a href="#">Role Management</a> page.
accessKey	Role token, which can be copied in the <b>Token</b> column on the <a href="#">Role Management</a> page. <div data-bbox="384 1081 1520 1227" data-label="Image"> <p>The screenshot shows a table with columns: Name, Key, Description, Creation Time, and Last Updated. A row with Name 'user' is shown, and a red arrow points to a 'Copy' button in the Key column.</p> </div>
namespace	Namespace name, which can be copied on the <b>Namespace</b> page in the console.

3. Send a message.



```
// Initialize message content
MQMessage msg(
    topicName, // Topic name
    TAGS,      // Message tag
    KEYS,      // Message key
    "Hello cpp client, this is a message." // Message content
);

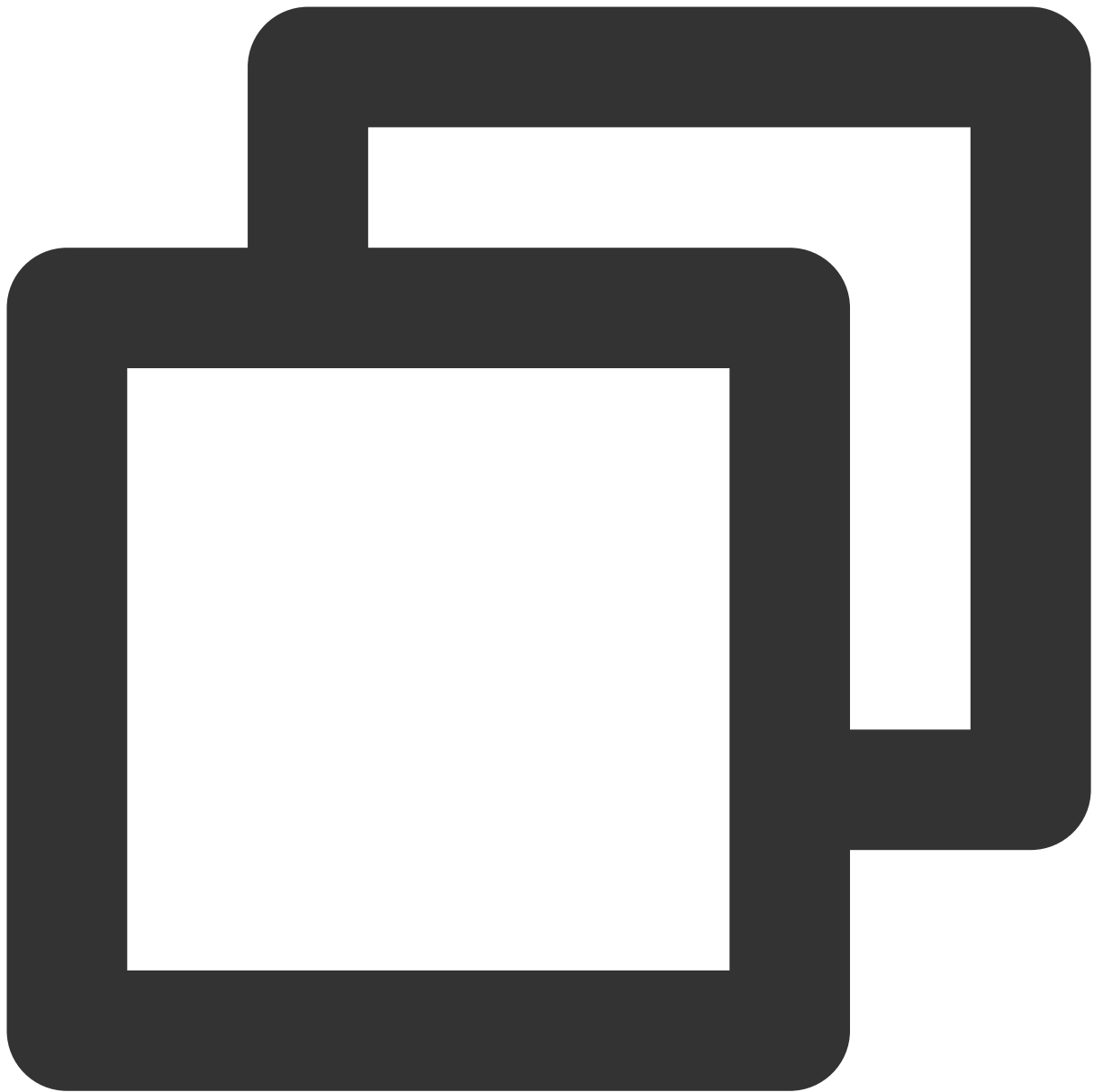
try {
    // Send the message
    SendResult sendResult = producer.send(msg);
}
```



```
std::cout << "SendResult:" << sendResult.getSendStatus() << ", Message ID: "
    << std::endl;
} catch (MQException e) {
    std::cout << "ErrorCode: " << e.GetError() << " Exception:" << e.what() << s
}
```

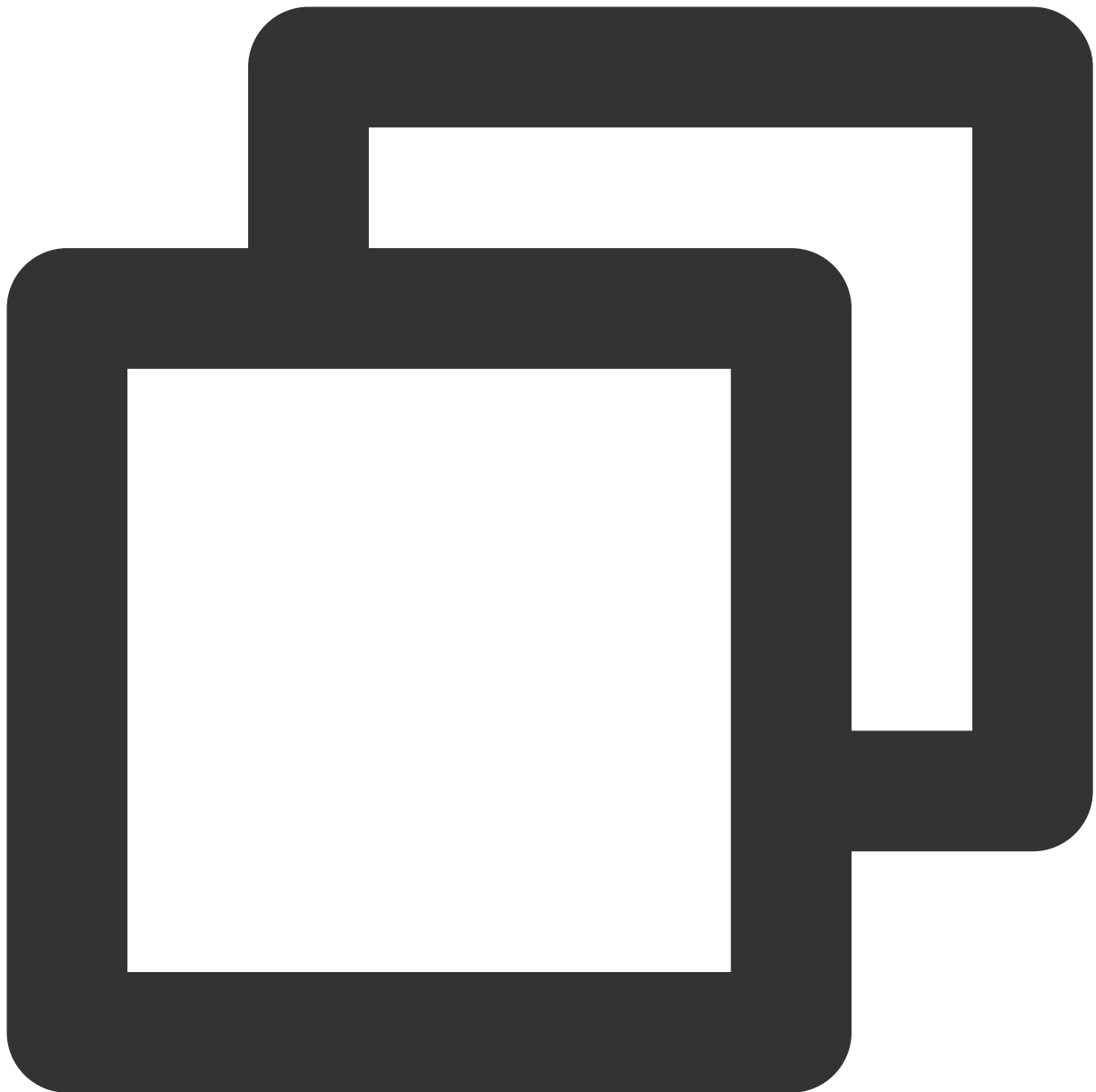
Parameter	Description
topicName	Topic name, which can be copied on the <b>Topic</b> page in the console.
TAGS	A parameter used to set the message tag.
KEYS	A parameter used to set the message key.

#### 4. Release the resource.



```
// Release resources  
producer.shutdown();
```

5. Initialize the consumer.



```
// Listen on messages
class ExampleMessageListener : public MessageListenerConcurrently {
public:
    ConsumeStatus consumeMessage(const std::vector<MQMessageExt> &msgs) {
        for (auto item = msgs.begin(); item != msgs.end(); item++) {
            // Business
            std::cout << "Received Message Topic:" << item->getTopic() << ", Msg
                << item->getTags() << ", KEYS:" << item->getKeys() << ", B
        }
        // Return CONSUME_SUCCESS if the consumption is successful
        return CONSUME_SUCCESS;
    }
};
```

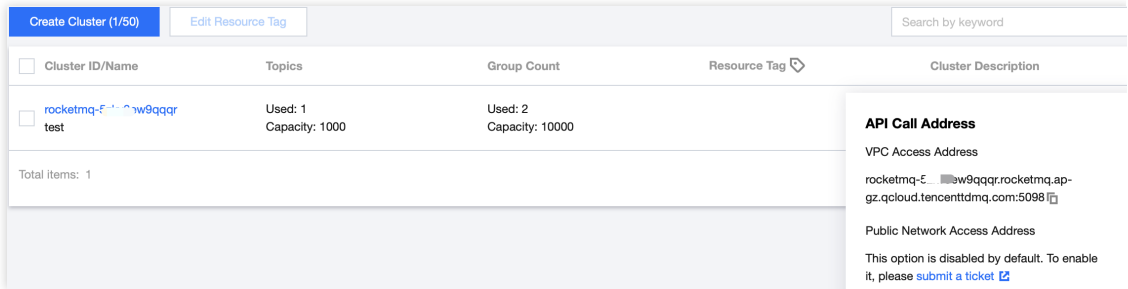
```

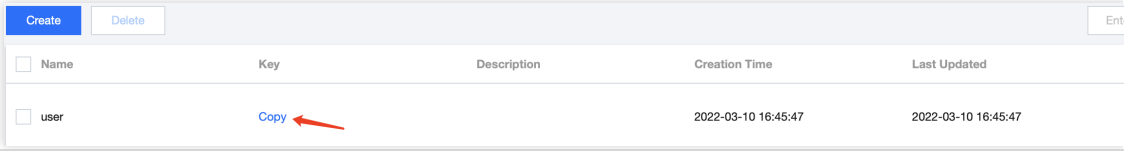
        // Return RECONSUME_LATER if the consumption failed. The message will be
        // return RECONSUME_LATER;
    }
};

// Initialize the consumer
DefaultMQPushConsumer *consumer = new DefaultMQPushConsumer(groupName);
// Set the service address
consumer->setNamesrvAddr(nameserver);
// Set user permissions
consumer->setSessionCredentials(
    accessKey,
    secretKey,
    "");
// Set the namespace
consumer->setNameSpace(namespace);
// Set the instance name
consumer->setInstanceName("CppClient");

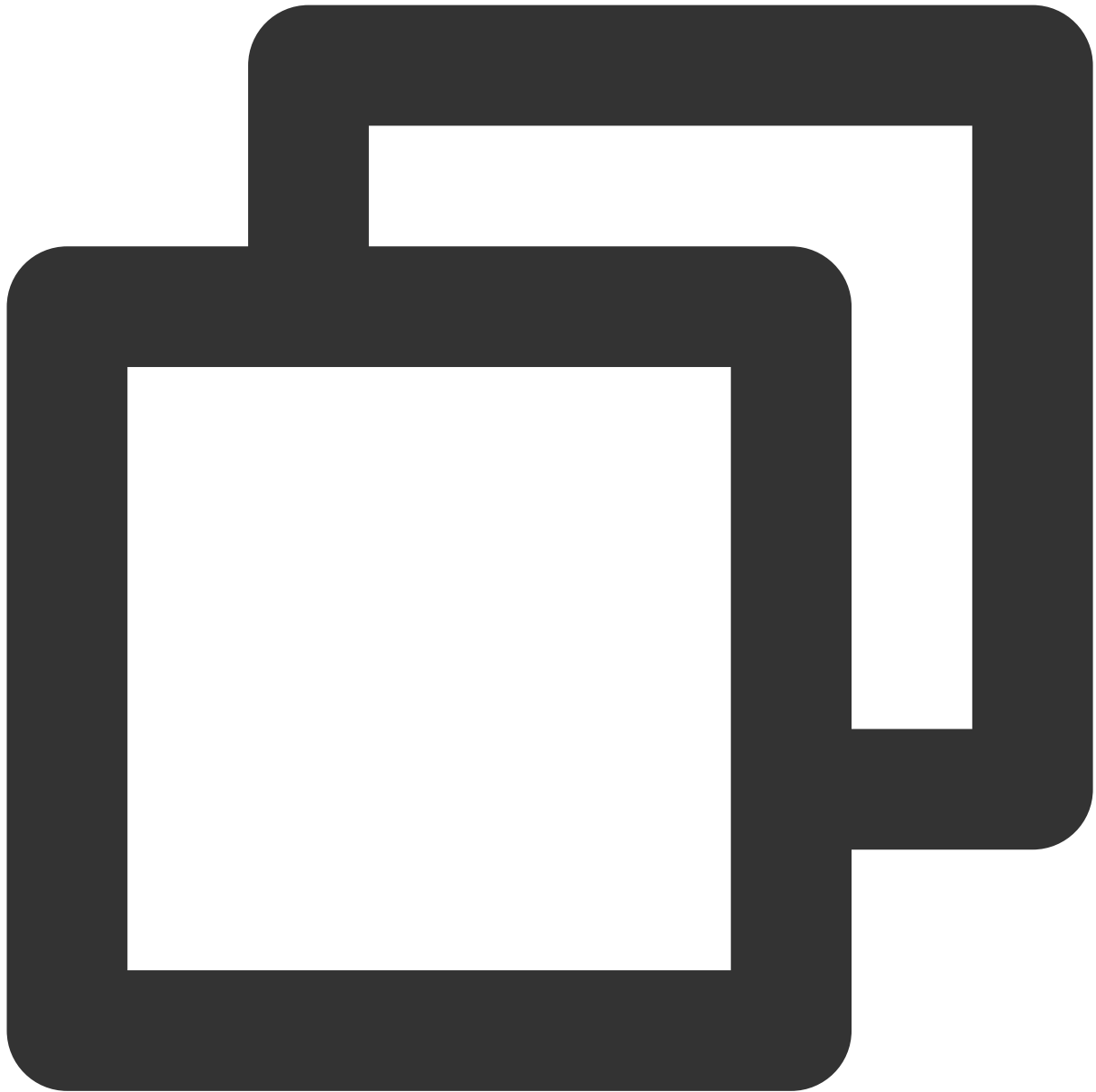
// Register a custom listener function to process the received messages and return
ExampleMessageListener *messageListener = new ExampleMessageListener();
// Subscribe to the message
consumer->subscribe(topicName, TAGS);
// Set the message listener
consumer->registerMessageListener(messageListener);

// After the preparations, you must call the start function before the consumption
consumer->start();
    
```

Parameter	Description
groupName	Consumer group name, which can be obtained under the <b>Group</b> tab on the cluster details page in the
nameserver	Cluster access address, which can be obtained in the <b>Operation</b> column on the <b>Cluster Manager</b> Namespace access addresses in new virtual or exclusive clusters can be copied from the <b>Namesp</b> 
secretKey	Role name, which can be copied on the <a href="#">Role Management</a> page.
accessKey	Role token, which can be copied in the <b>Token</b> column on the <a href="#">Role Management</a> page.

	
namespace	Namespace name, which can be copied on the <b>Namespace</b> page in the console.
topicName	Topic name, which can be copied on the <b>Topic</b> page in the console.
TAGS	A parameter used to set the message tag.

6. Release the resource.



```
// Release resources  
consumer->shutdown();
```

7. View consumer details. Log in to the [TDMQ console](#), go to the **Cluster > Group** page, and view the list of clients connected to the group. Click **View Details** in the **Operation** column to view consumer details.

Basic Info    Namespace    Topic    **Group**

Current Namespace:     Message Retention Period: 3 days    Max TPS ⓘ: 4000

[Create \(2/1500\)](#)   

Group Name	Consumer Info ↕	Consumption Mode	Description
group-364733	Online Consumer    0    TPS   0    Total Heap   0    ↻	Unknown	
dasda	Online Consumer    0    TPS   0    Total Heap   0    ↻	Unknown	

Total items: 2

**Basic Info**

Group Name	group-364733	Creation Time	2022-03-11 15:13:15
Consumption Mode	Unknown	Client Protocol	TCP
Total Heaped Messages	0	Consumer Type	Unknown

**Client Address**    Subscription

Client Address	Client Language	Client Version	Message Heap ↕
No data yet			

Total items: 0 20

**Note**

Above is a brief introduction to message publishing and subscription. Above is a brief introduction to message publishing and For more information, see [Demo](#) or [RocketMQ-Client-CPP Example](#).

# SDK for Go

Last updated : 2023-09-12 17:53:17

## 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 as instructed in [Resource Creation and Preparation](#).

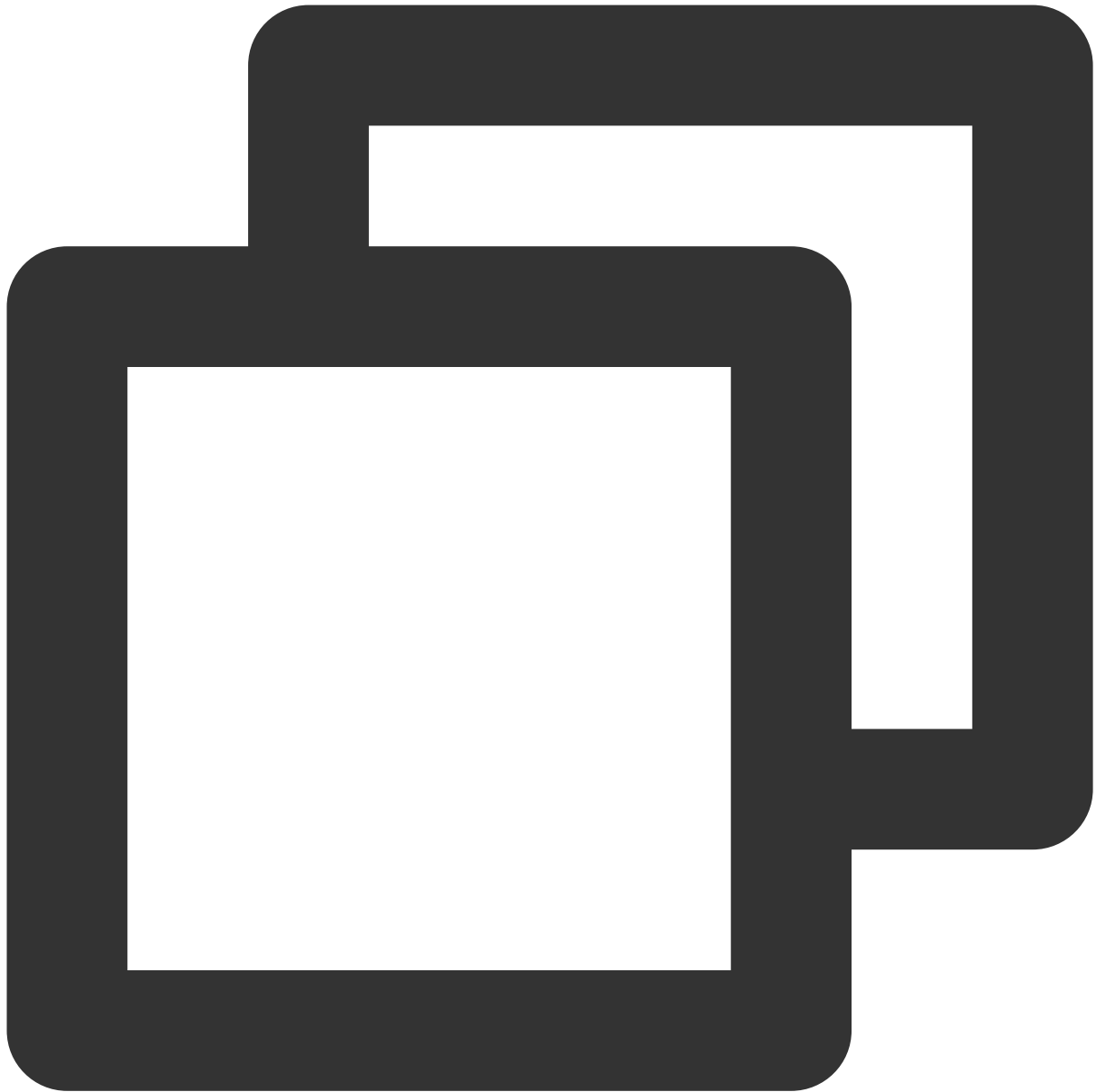
You have installed [Go](#).

[You have downloaded the demo](#).

## Directions

1. Run the following command in the client environment to RocketMQ client dependencies.





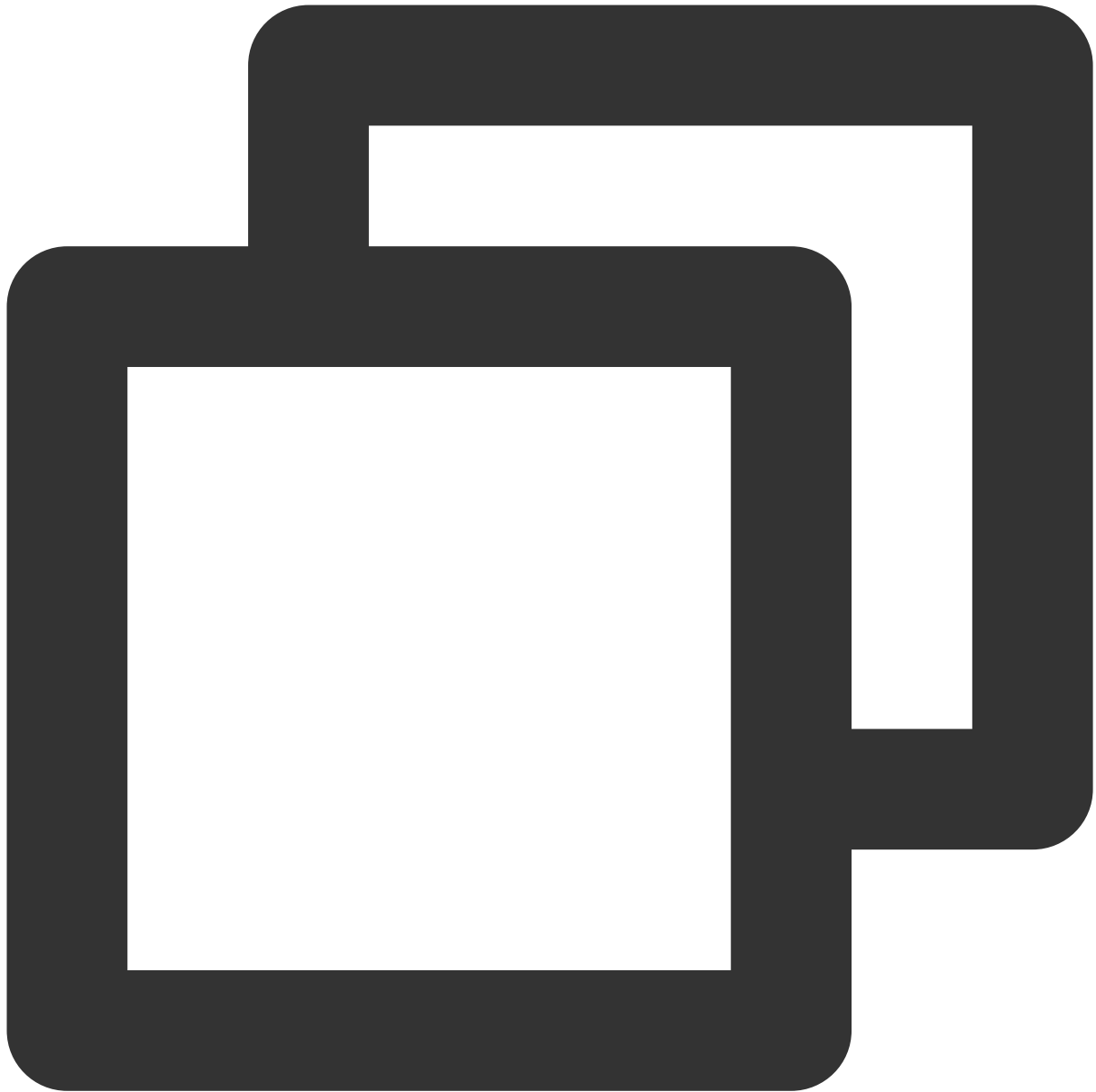
```
go get github.com/apache/rocketmq-client-go/v2
```

2. Create a producer in the corresponding method. If you need to send general messages, modify the corresponding parameters in the `syncSendMessage.go` file.

Delayed messages currently support delays of arbitrary precision without being subject to the delay level.

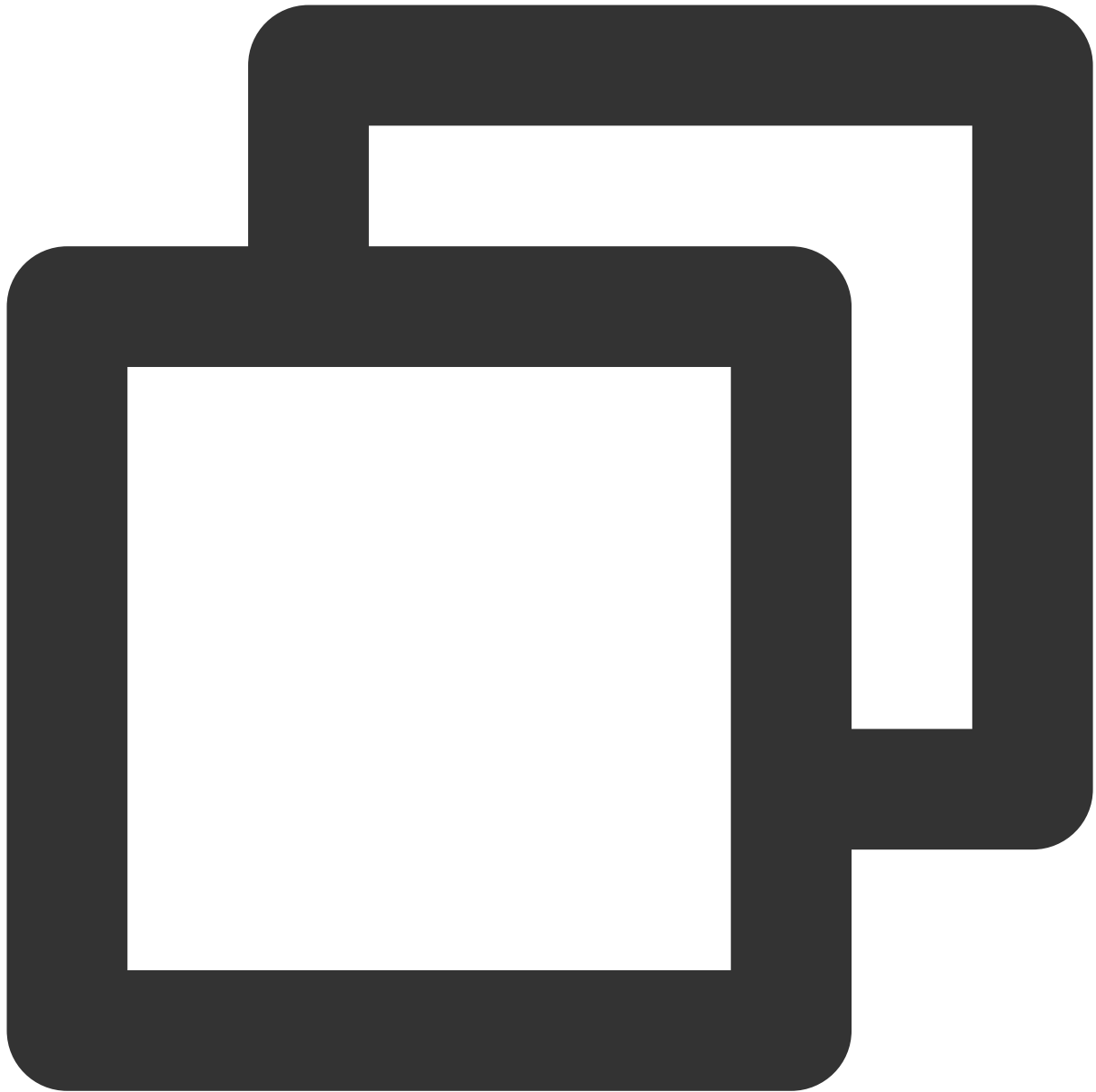
General Message

Delayed message



```
// Service access address (Note: Add "http://" or "https://" before the access address)
var serverAddress = "https://rocketmq-xxx.rocketmq.ap-bj.public.tencenttdmq.com:
// Authorize the role name
var secretKey = "admin"
// Authorize the role token
var accessKey = "eyJrZXlJZC...."
// Full namespace name
var namespace = "MQ_INST_rocketmqem4xxxx"
// Producer group name
var groupName = "group1"
// Create a message producer
```

```
p, _ := rocketmq.NewProducer(  
    // Set the service address  
    producer.WithNsResolver(primitive.NewPassthroughResolver([]string{serverAddr}  
    // Set ACL permissions  
    producer.WithCredentials(primitive.Credentials{  
        SecretKey: secretKey,  
        AccessKey: accessKey,  
    }),  
    // Set the producer group  
    producer.WithGroupName(groupName),  
    // Set the namespace name  
    producer.WithNamespace(nameSpace),  
    // Set the number of retries upon sending failures  
    producer.WithRetry(2),  
)  
// Start the producer  
err := p.Start()  
if err != nil {  
    fmt.Printf("start producer error: %s", err.Error())  
    os.Exit(1)  
}
```



```
// Topic name
var topicName = "topic1"
// Producer group name
var groupName = "group1"
// Create a message producer
p, _ := rocketmq.NewProducer(
    // Set the service address
    producer.WithNsResolver(primitive.NewPassthroughResolver([]string{"
    // Set ACL permissions
    producer.WithCredentials(primitive.Credentials{
```

```

        SecretKey: "admin",
        AccessKey: "eyJrZXlJZC.....",
    }),
    // Set the producer group
    producer.WithGroupName(groupName),
    // Set the namespace name
    producer.WithNamespace("rocketmq-xxx|namespace_go"),
    // Set the number of retries upon sending failures
    producer.WithRetry(2),
)
// Start the producer
err := p.Start()
if err != nil {
    fmt.Printf("start producer error: %s", err.Error())
    os.Exit(1)
}

for i := 0; i < 1; i++ {
    msg := primitive.NewMessage(topicName, []byte("Hello RocketMQ Go Cl
    // Set delay level
    // The relationship between the delay level and the delay time:
    // 1s, 5s, 10s, 30s, 1m, 2m, 3m, 4m, 5m, 6m, 7m, 8m, 9m, 10m, 20m,
    // 1    2    3    4    5    6    7    8    9    10    11    12    13    1
    // If you want to use the delay level, then set the following metho

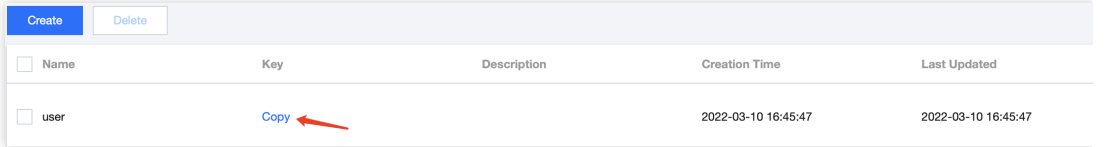
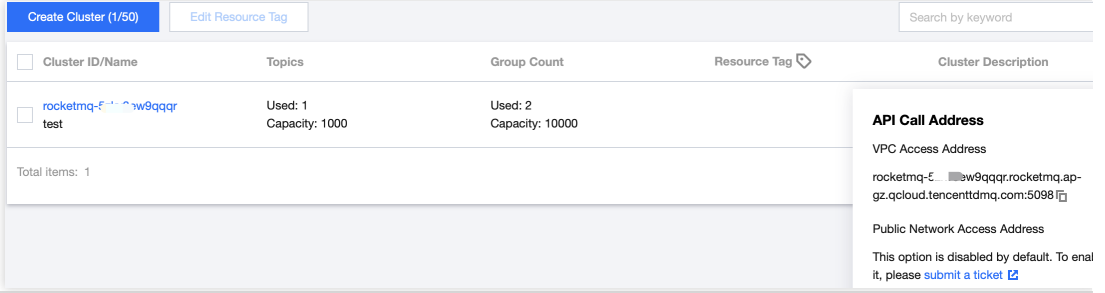
    msg.WithDelayTimeLevel(3)
    // If you want to use any delayed message, then set the following m
    delayMills := int64(10 * 1000)
    msg.WithProperty("__STARTDELIVERTIME", strconv.FormatInt(time.Now()
    // Send the message

res, err := p.SendSync(context.Background(), msg)
    if err != nil {
        fmt.Printf("send message error: %s\n", err)
    } else {
        fmt.Printf("send message success: result=%s\n", res.String
    }
}

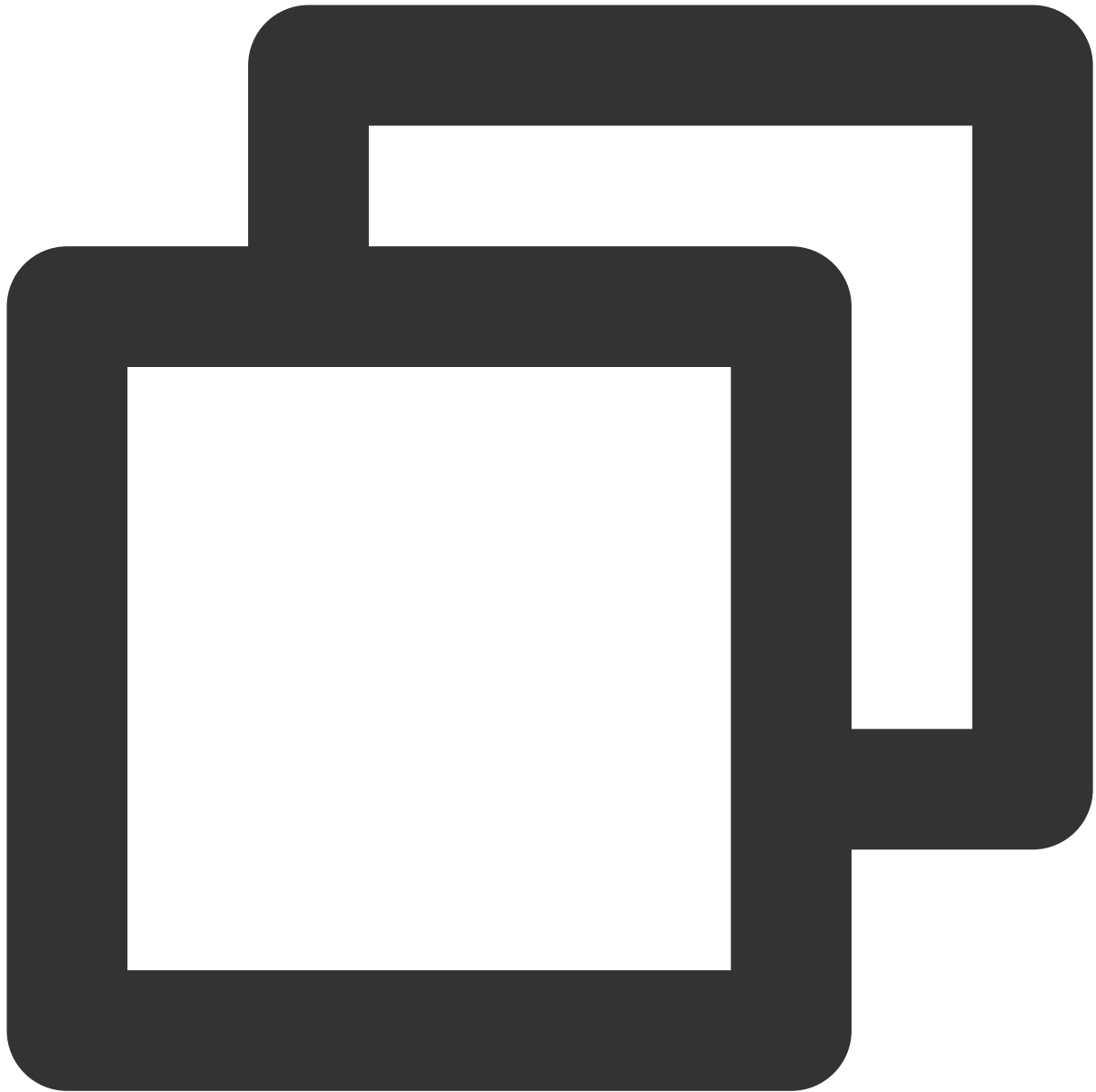
// Release resources
err = p.Shutdown()
if err != nil {
    fmt.Printf("shutdown producer error: %s", err.Error())
}

```

Parameter	Description

secretKey	Role name, which can be copied on the <a href="#">Role Management</a> page.
accessKey	<p>Role token, which can be copied in the <b>Token</b> column on the <a href="#">Role Management</a> page.</p> 
nameSpace	Namespace name, which can be copied on the <b>Namespace</b> page in the console.
serverAddress	<p>Cluster access address, which can be copied from <b>Access Address</b> in the <b>Operation</b> column of console. Namespace access addresses in new virtual or exclusive clusters can be copied from the Add <code>http://</code> or <code>https://</code> before the access address; otherwise, it cannot be resolved.</p> 
groupName	Producer group name, which can be copied under the <b>Group</b> tab in the console.

3. The process of sending messages (using sync sending as an example) is the same as above.



```
// Topic name
var topicName = "topic1"
// Configure message content
msg := &primitive.Message{
    Topic: topicName, // Set the topic name
    Body:  []byte("Hello RocketMQ Go Client! This is a new message."),
}
// Set tags
msg.WithTag("TAG")
// Set keys
msg.WithKeys([]string{"yourKey"})
```

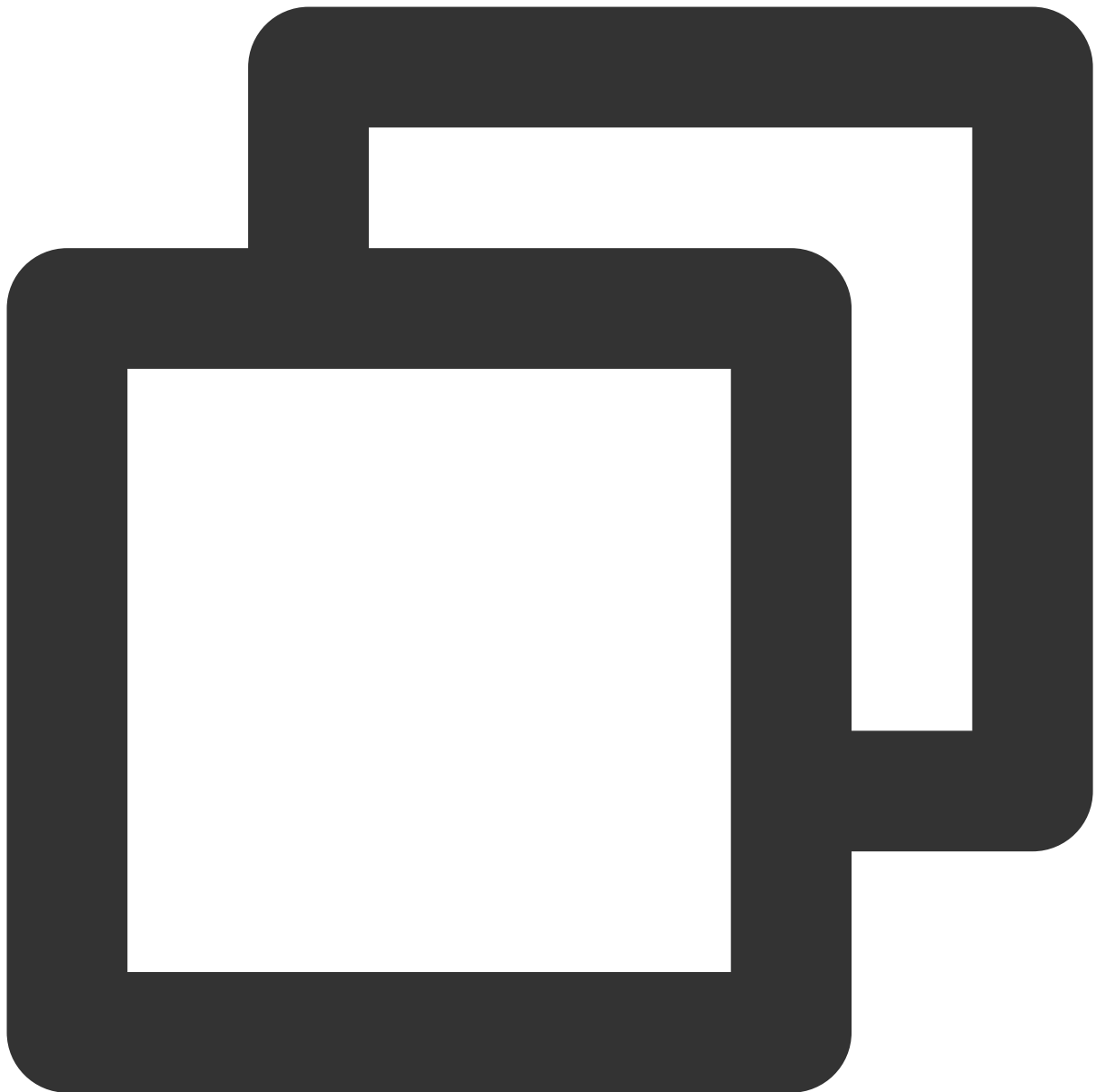
```
// Send the message
res, err := p.SendSync(context.Background(), msg)

if err != nil {
    fmt.Printf("send message error: %s\n", err)
} else {
    fmt.Printf("send message success: result=%s\n", res.String())
}
```

Parameter	Description
topicName	Topic name, which can be copied under the <b>Topic</b> tab on the cluster details page in the console.
TAG	Message tag identifier
yourKey	Message business key

Release the resource.



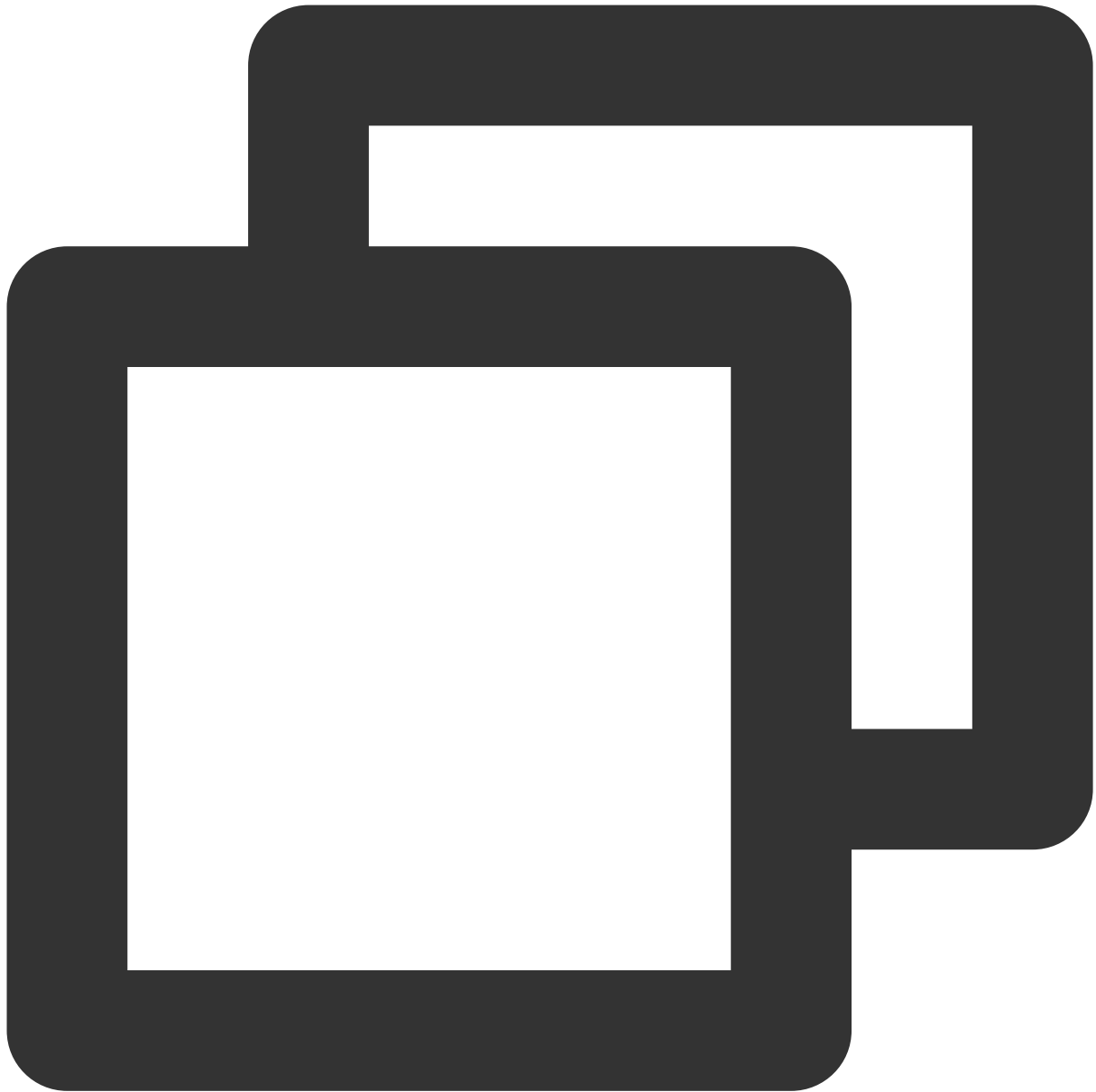


```
// Disable the producer
err = p.Shutdown()
if err != nil {
    fmt.Printf("shutdown producer error: %s", err.Error())
}
```

**Note**

For more information on async sending and one-way sending, see [Demo](#) or [RocketMQ-Client-Go Example](#).

4. Create a consumer.



```
// Service access address (Note: Add "http://" or "https://" before the access address)
var serverAddress = "https://rocketmq-xxx.rocketmq.ap-bj.public.tencenttdmq.com:
// Authorize the role name
var secretKey = "admin"
// Authorize the role token
var accessKey = "eyJrZXlJZC...."
// Full namespace name
var namespace = "rocketmq-xxx|namespace_go"
// Producer group name
var groupName = "group11"
// Create a consumer
```

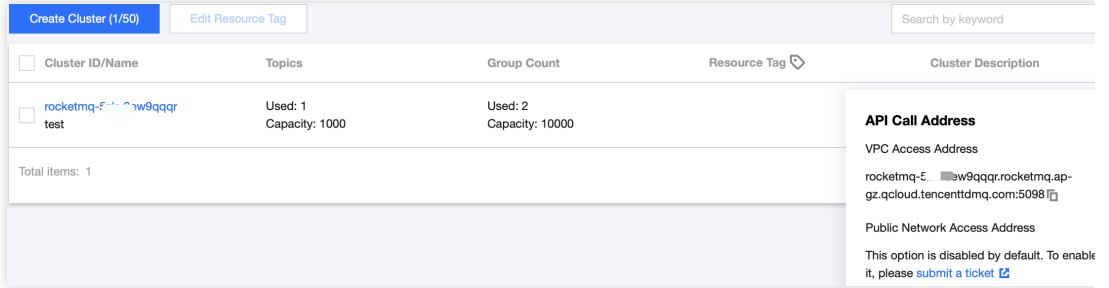
```

c, err := rocketmq.NewPushConsumer(
    // Set the consumer group
    consumer.WithGroupName(groupName),
    // Set the service address
    consumer.WithNsResolver(primitive.NewPassthroughResolver([]string{serverAddr})),
    // Set ACL permissions
    consumer.WithCredentials(primitive.Credentials{
        SecretKey: secretKey,
        AccessKey: accessKey,
    }),
    // Set the namespace name
    consumer.WithNamespace(nameSpace),
    // Set consumption from the start offset
    consumer.WithConsumeFromWhere(consumer.ConsumeFromFirstOffset),
    // Set the consumption mode (cluster consumption by default)
    consumer.WithConsumerModel(consumer.Clustering),

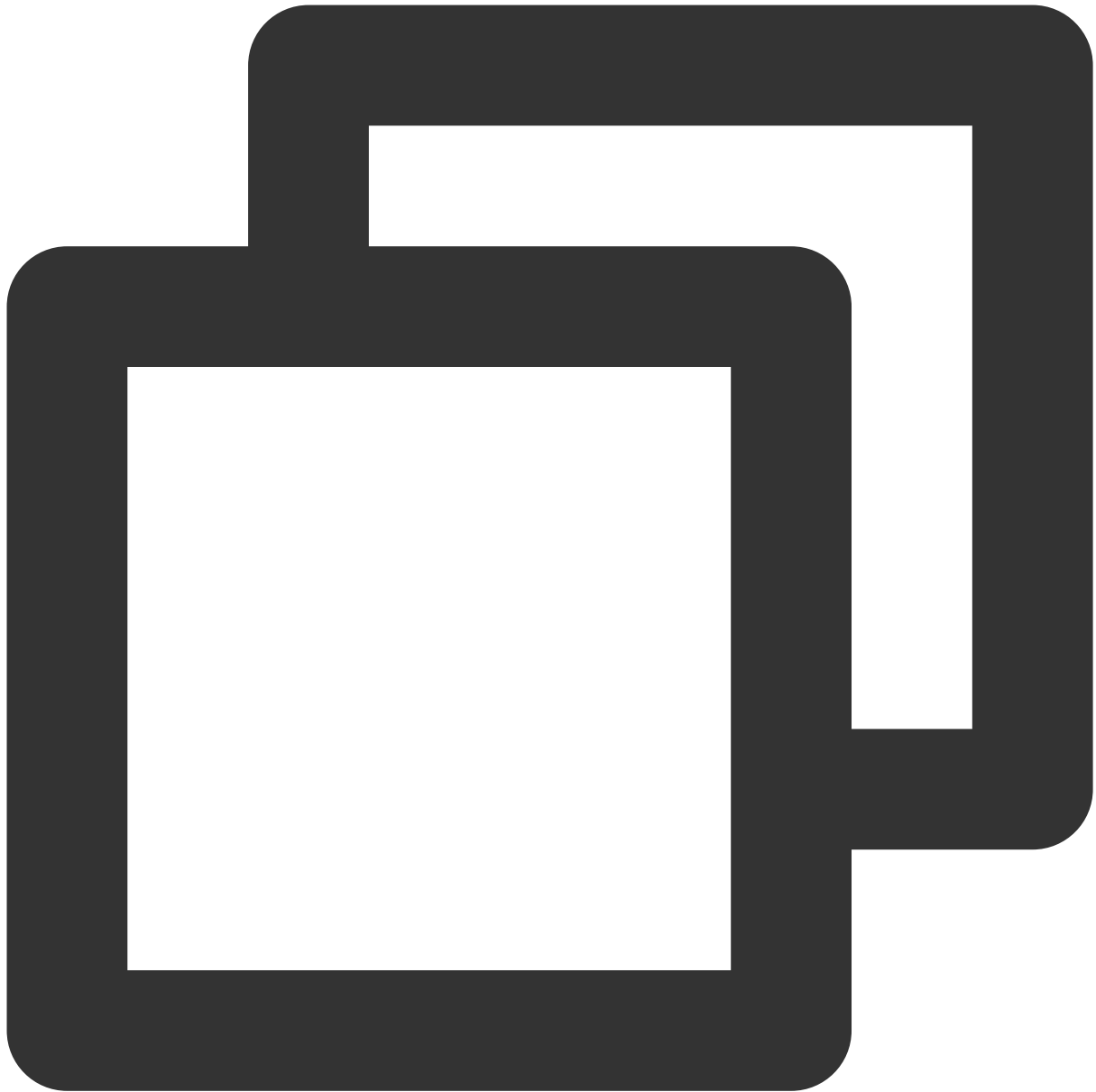
    //For broadcasting consumption, set the instance name to the system name of
    consumer.WithInstance("xxxx"),
)
if err != nil {
    fmt.Println("init consumer2 error: " + err.Error())
    os.Exit(0)
}

```

Parameter	Description										
secretKey	Role name, which can be copied on the <a href="#">Role Management</a> page.										
accessKey	Role token, which can be copied in the <b>Token</b> column on the <a href="#">Role Management</a> page. <div data-bbox="416 1435 1516 1585" data-label="Image"> <table border="1"> <thead> <tr> <th>Name</th> <th>Key</th> <th>Description</th> <th>Creation Time</th> <th>Last Updated</th> </tr> </thead> <tbody> <tr> <td><input type="checkbox"/> user</td> <td>Copy</td> <td></td> <td>2022-03-10 16:45:47</td> <td>2022-03-10 16:45:47</td> </tr> </tbody> </table> </div>	Name	Key	Description	Creation Time	Last Updated	<input type="checkbox"/> user	Copy		2022-03-10 16:45:47	2022-03-10 16:45:47
Name	Key	Description	Creation Time	Last Updated							
<input type="checkbox"/> user	Copy		2022-03-10 16:45:47	2022-03-10 16:45:47							
nameSpace	The full namespace name can be copied under the <b>Topic</b> tab on the <b>Cluster</b> page in the console cluster ID +   + namespace.										
serverAddress	Cluster access address, which can be copied from <b>Access Address</b> in the <b>Operation</b> column on the console. Namespace access addresses in new virtual or exclusive clusters can be copied from <b>Note</b> : Add <code>http://</code> or <code>https://</code> before the access address; otherwise, it cannot be resolved.										

	
<p>groupName</p>	<p>Producer group name, which can be copied under the <b>Group</b> tab in the console.</p>

5. Consume a message.



```
// Topic name
var topicName = "topic1"
// Set the tag of messages that are subscribed to
selector := consumer.MessageSelector{
    Type:      consumer.TAG,
    Expression: "TagA || TagC",
}
// Set the delay level of consumption retry. A total of 18 levels can be set. Be
// 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
// 1s, 5s, 10s, 30s, 1m, 2m, 3m, 4m, 5m, 6m, 7m, 8m, 9m, 10m, 20m, 30m, 1h, 2h
delayLevel := 1
```

```

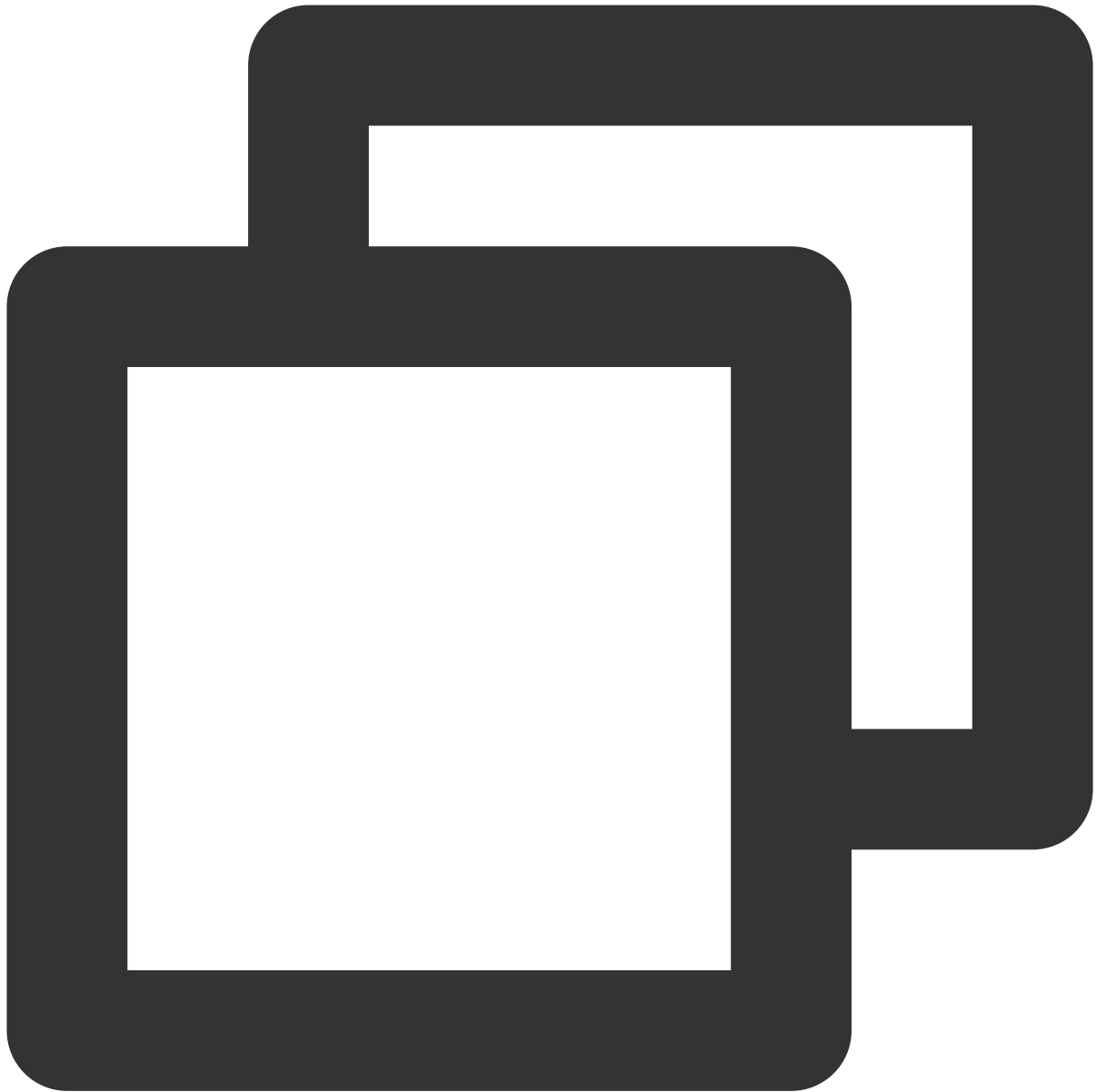
err = c.Subscribe(topicName, selector, func(ctx context.Context,
                                                    msgs ...*primitive
                                                    fmt.Printf("subscribe callback len: %d \n", len(msgs))
// Set the delay level for the next consumption
concurrentCtx, _ := primitive.GetConcurrentlyCtx(ctx)
concurrentCtx.DelayLevelWhenNextConsume = delayLevel // only run when return

for _, msg := range msgs {
    // Simulate a successful consumption after three retries
    if msg.ReconsumeTimes > 3 {
        fmt.Printf("msg ReconsumeTimes > 3. msg: %v", msg)
        return consumer.ConsumeSuccess, nil
    } else {
        fmt.Printf("subscribe callback: %v \n", msg)
    }
}
// Simulate a consumption failure. Retry is required.
return consumer.ConsumeRetryLater, nil
}))
if err != nil {
    fmt.Println(err.Error())
}

```

Parameter	Description
topicName	Topic name, which can be copied on the <b>Topic</b> page in the console.
Expression	Message tag identifier
delayLevel	A parameter used to set the delay level of consumption retry. A total of 18 delay levels are supported.

6. Consume messages (the consumer can consume messages only after the messages are subscribed to).



```
// Start consumption
err = c.Start()
if err != nil {
    fmt.Println(err.Error())
    os.Exit(-1)
}
time.Sleep(time.Hour)
// Release resources
err = c.Shutdown()
if err != nil {
    fmt.Printf("shundown Consumer error: %s", err.Error())
}
```

```
}
```

7. View consumption details. Log in to the [TDMQ console](#), go to the **Cluster > Group** page, and view the list of clients connected to the group. Click **View Details** in the **Operation** column to view consumer details.

The screenshot shows the 'Group' page in the TDMQ console. At the top, there are tabs for 'Basic Info', 'Namespace', 'Topic', and 'Group'. Below the tabs, there is a 'Current Namespace' dropdown set to 'sdaa', a 'Message Retention Period' of '3 days', and a 'Max TPS' of '4000'. A blue 'Create (2/1500)' button is on the left, and a search bar is on the right. The main content is a table with columns: 'Group Name', 'Consumer Info', 'Consumption Mode', and 'Description'. Two rows are visible, both for 'group-364733' and 'dasda', showing 'Online Consumer' status with 0 TPS and 0 Total Heap. At the bottom, it says 'Total items: 2' and '20 / page'.

The screenshot shows the 'Basic Info' and 'Client Address' sections. The 'Basic Info' section has a table with the following data:

Group Name	group-364733	Creation Time	2022-03-11 15:13:15
Consumption Mode	Unknown	Client Protocol	TCP
Total Heaped Messages	0	Consumer Type	Unknown

The 'Client Address' section has a sub-tab 'Subscription' and a table with columns: 'Client Address', 'Client Language', 'Client Version', and 'Message Heap'. The table is empty with the text 'No data yet'. At the bottom, it says 'Total items: 0' and '20 / page'.

**Note**  
 Above is a brief introduction to how to send and receive messages with the Go client. For more information, see [Demo](#) or [Rocketmq-Client-Go Example](#).



# SDK for Python

Last updated : 2023-09-12 17:53:17

## Overview

This document describes how to use open-source SDK to send and receive messages by 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 as instructed in [Resource Creation and Preparation](#).

[You have installed Python](#).

[You have installed pip](#).

[You have downloaded the demo](#).

## Directions

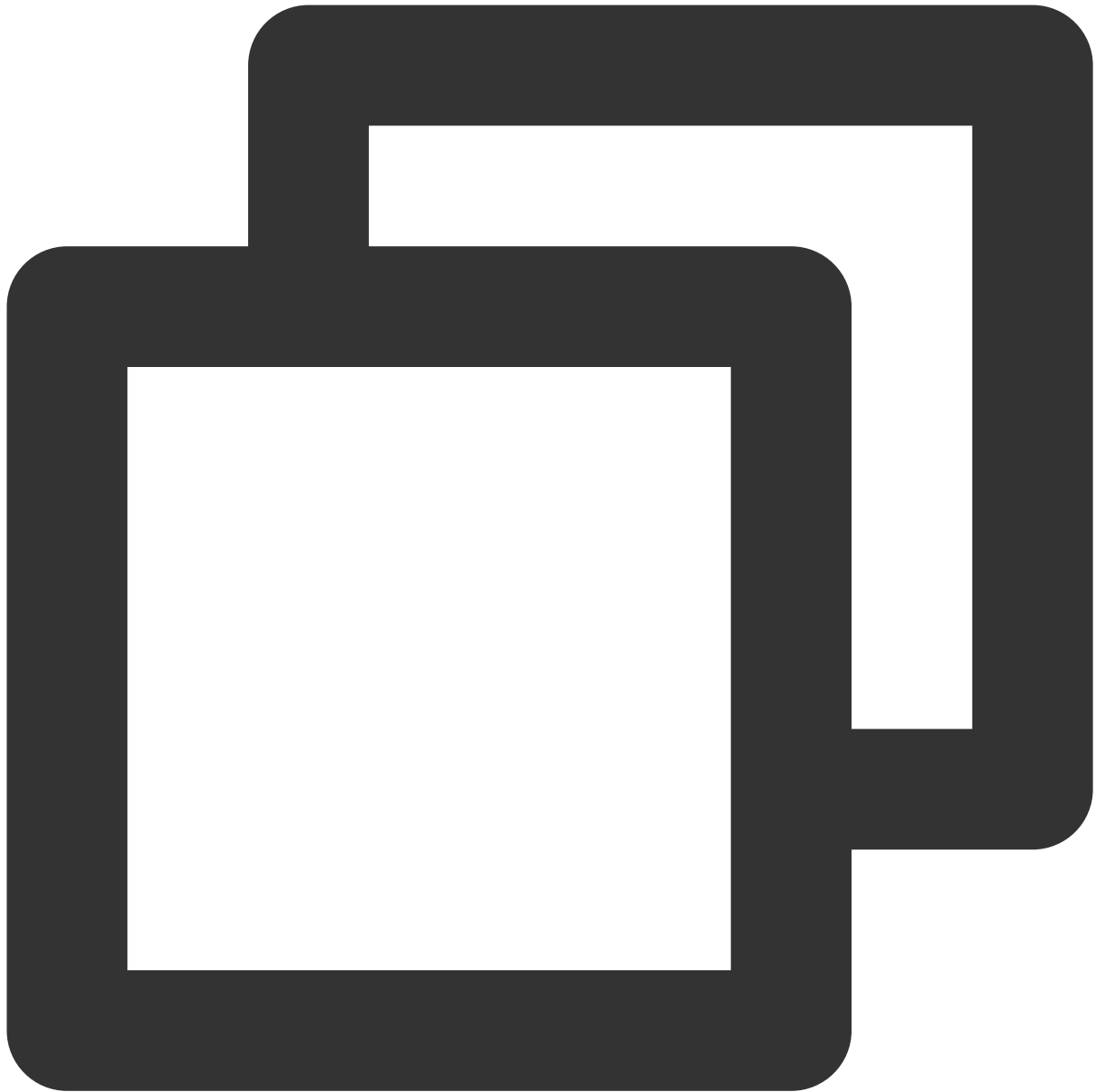
### Step 1. Prepare the environment

As RocketMQ-client Python is lightweight wrapper around [rocketmq-client-cpp](#), you need to install `librocketmq` first.

#### Note

Currently, the Python client only supports Linux and macOS operating systems. It doesn't support Windows systems.

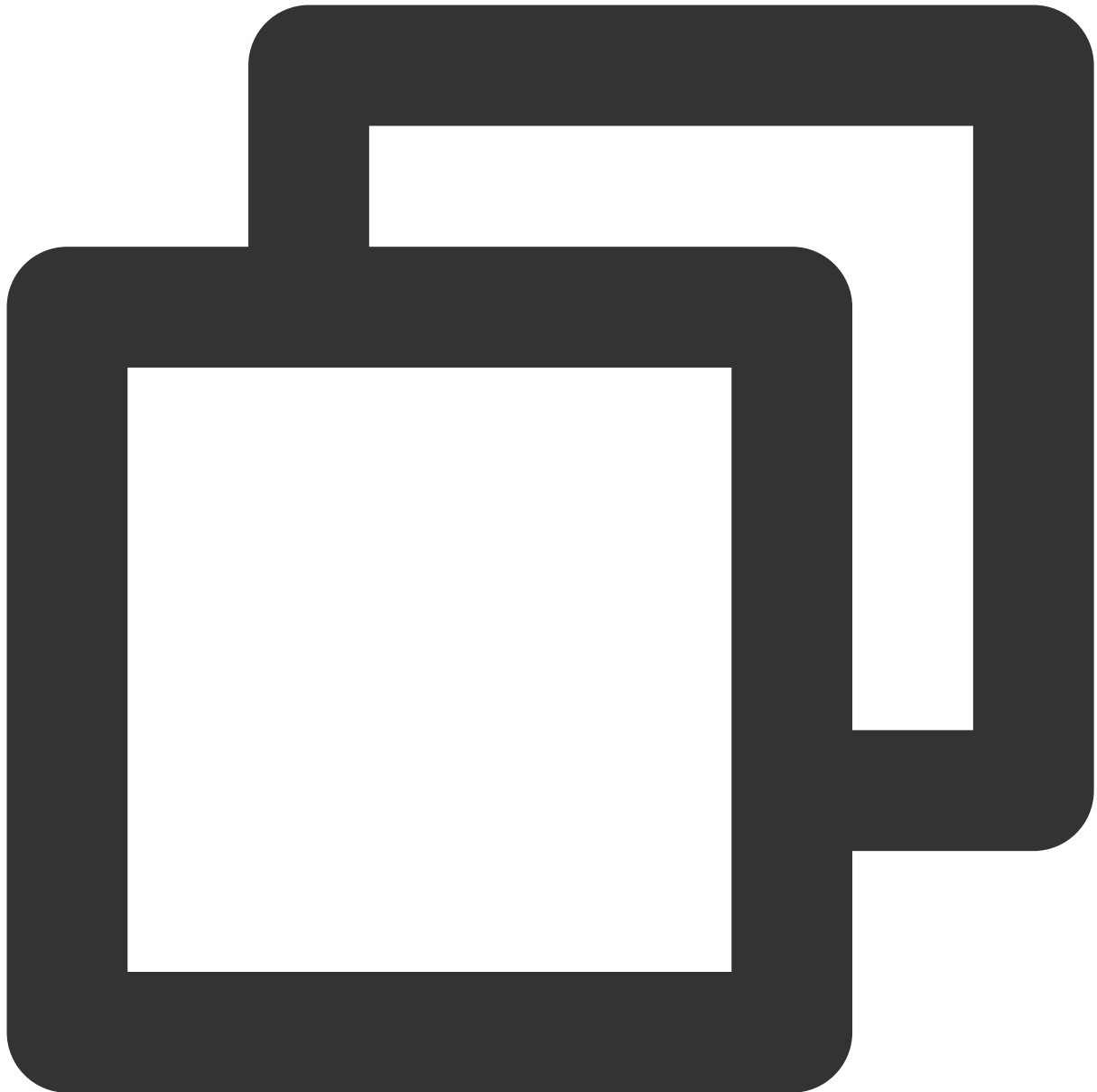
1. Install `librocketmq` 2.0.0 or later as instructed in [Install librocketmq](#).
2. Run the following command to install `rocketmq-client-python` .



```
pip install rocketmq-client-python
```

## Step 2. Produce messages

Create, compile, and run a message production program.



```
from rocketmq.client import Producer, Message

# Initialize the producer and set the producer group information. Be sure to use
producer = Producer(groupName)
# Set the service address
producer.set_name_server_address(nameserver)
# Set permissions (role name and token)
producer.set_session_credentials(
    accessKey, # Role token
    secretKey, # Role name
    ''
```

```

)
# Start the producer
producer.start()

# Assemble messages. The topic name can be copied on the **Topic** page in the console
msg = Message(topicName)
# Set keys
msg.set_keys(TAGS)
# Set tags
msg.set_tags(KEYS)
# Message content
msg.set_body('This is a new message.')
```

```

# Send messages in sync mode
ret = producer.send_sync(msg)
print(ret.status, ret.msg_id, ret.offset)
# Release resources
producer.shutdown()
```

Parameter	Description
groupName	Producer group name, which can be obtained under the <b>Group</b> tab on the cluster details page in the console.
nameserver	Cluster access address, which can be copied from <b>Access Address</b> in the <b>Operation</b> column on the <b>Cluster</b> page in the console. Namespace access addresses in new virtual or exclusive clusters can be copied from the <b>Namespace</b> list. <div data-bbox="384 1317 1517 1574" data-label="Image"> <p>The screenshot shows a table with columns: Cluster ID/Name, Topics, Group Count, Resource Tag, Cluster Description, and Operation. A cluster named 'test' is selected, and its 'Operation' column is expanded to show 'API Call Address' (rocketmq-f...w9qqq) and 'Public Network Access Address' (This option is disabled by default. To enable it, please submit a ticket).</p> </div>
secretKey	Role name, which can be copied on the <a href="#">Role Management</a> page.
accessKey	Role token, which can be copied in the <b>Token</b> column on the <a href="#">Role Management</a> page. <div data-bbox="384 1771 1517 1899" data-label="Image"> <p>The screenshot shows a table with columns: Name, Key, Description, Creation Time, Last Updated, and Operation. A role named 'user' is selected, and a red arrow points to the 'Copy' button in the 'Key' column.</p> </div>
topicName	Topic name, which can be copied on the <b>Topic</b> page in the console.
TAGS	A parameter used to set the message tag.

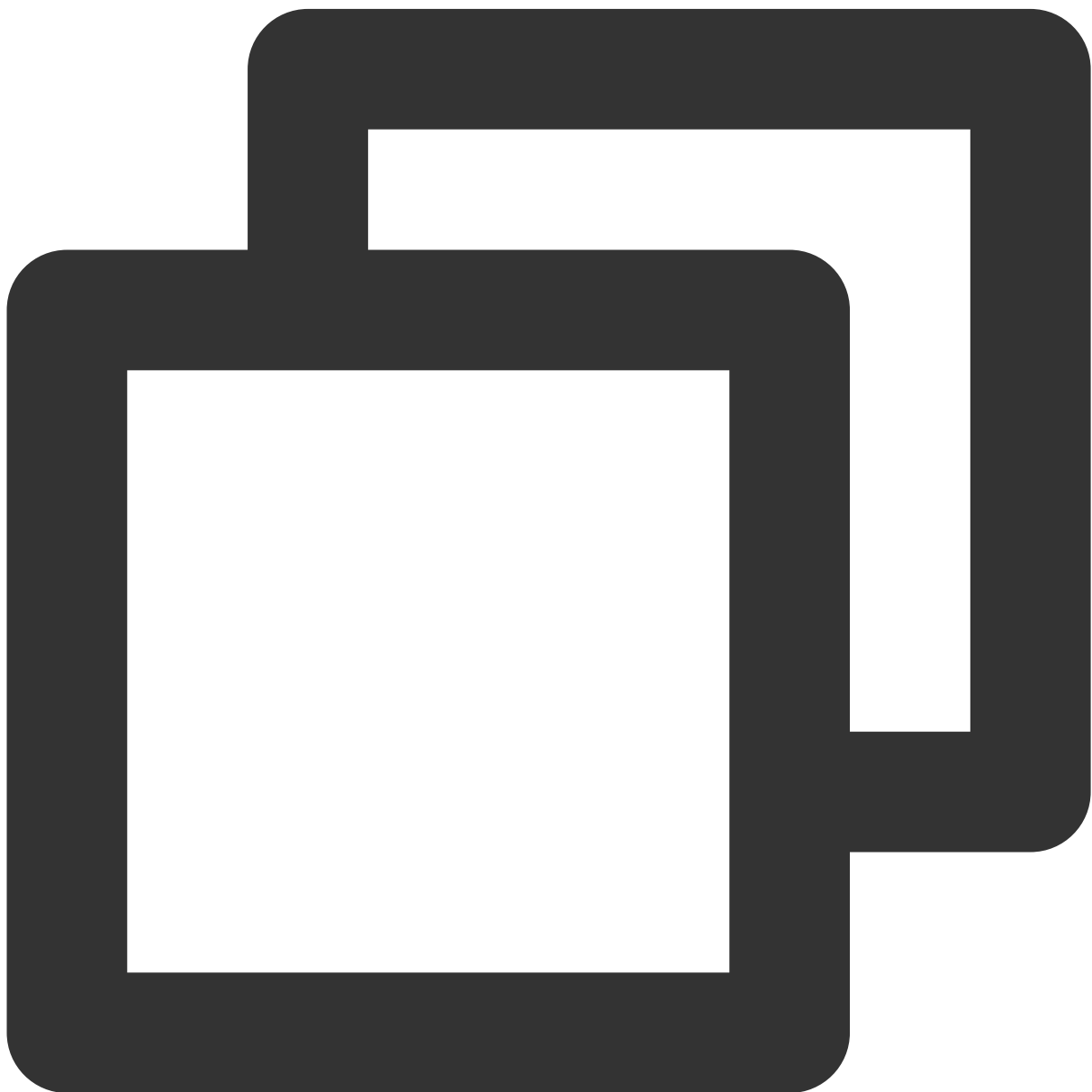
KEYS

A parameter used to set the message key.

There are certain defects in the message production of the open-source Python client, causing uneven load among different queues of the same Topic. For more information, see [RocketMQ document] (<https://github.com/apache/rocketmq-client-python/issues/128!cac28b204e4c02765f18ecd741ed1628>).

### Step 3. Consume messages

Create, compile, and run a message consumption program.



```
import time
```

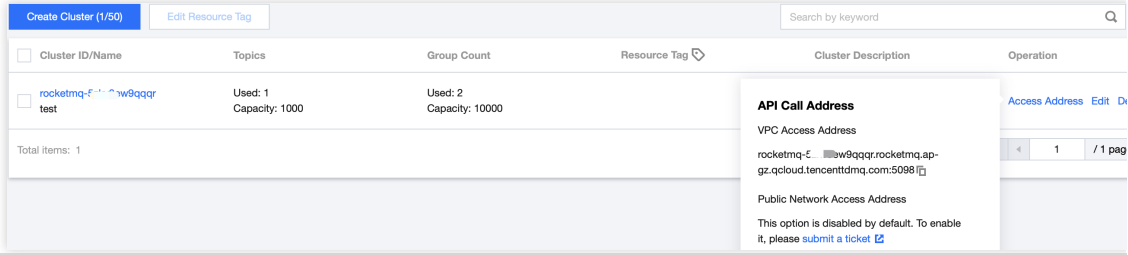
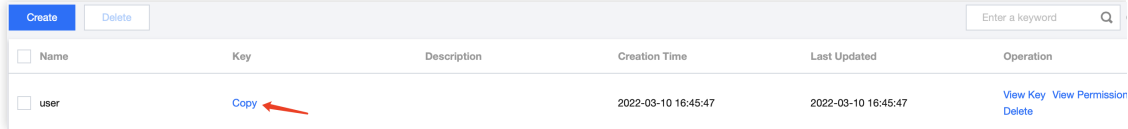
```
from rocketmq.client import PushConsumer, ConsumeStatus

# Message processing callback
def callback(msg):
    # Simulate the business processing logic
    print('Received message. messageId: ', msg.id, ' body: ', msg.body)
    # Return CONSUME_SUCCESS if the consumption is successful
    return ConsumeStatus.CONSUME_SUCCESS
    # Return the consumption status if the consumption is successful
    # return ConsumeStatus.RECONSUME_LATER

# Initialize the consumer and set the consumer group information
consumer = PushConsumer(groupName)
# Set the service address
consumer.set_name_server_address(nameserver)
# Set permissions (role name and token)
consumer.set_session_credentials(
    accessKey,      # Role token
    secretKey,     # Role name
    ''
)
# Subscribe to a topic
consumer.subscribe(topicName, callback, TAGS)
print(' [Consumer] Waiting for messages.')
# Start the consumer
consumer.start()

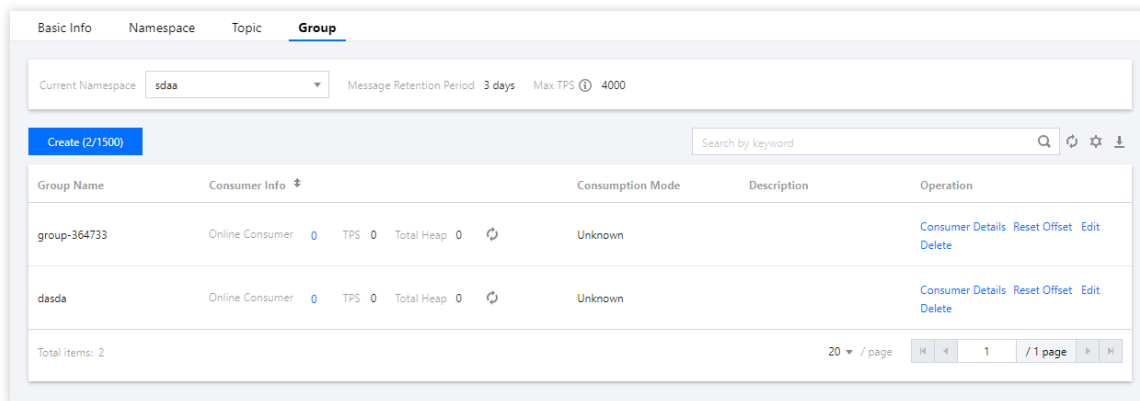
while True:
    time.sleep(3600)
# Release resources
consumer.shutdown()
```

Parameter	Description
groupName	Consumer group name, which can be copied under the <b>Group</b> tab on the cluster details page.
nameserver	Cluster access address, which can be copied from <b>Access Address</b> in the <b>Operation</b> column on the <b>Cluster</b> page in the console. Namespace access addresses in new virtual or exclusive clusters can be copied from the <b>Namespace</b> list.

	
secretKey	Role name, which can be copied on the <a href="#">Role Management</a> page.
accessKey	Role token, which can be copied in the <b>Token</b> column on the <a href="#">Role Management</a> page.
	
topicName	Topic name, which can be copied on the <b>Topic</b> page in the console.
TAGS	A parameter used to set the tag of messages that are subscribed to. The default value is set to <code>*</code> , means subscribing to all messages.

### Step 4. View consumption details

Log in to the [TDMQ console](#), go to the **Cluster > Group** page, and view the list of clients connected to the group. Click **View Details** in the **Operation** column to view consumer details.



### Basic Info

Group Name	group-364733	Creation Time	2022-03-11 15:13:15
Consumption Mode	Unknown	Client Protocol	TCP
Total Heaped Messages	0	Consumer Type	Unknown

### Client Address

Subscription

Client Address	Client Language	Client Version	Message Heap	Operation
No data yet				

Total items: 0

20 / page

## Note

Above is a brief introduction to message publishing and subscription. For more information, see [Demo](#) or [RocketMQ-Client-Python Sample](#).



# Access over HTTP

Last updated : 2023-05-16 11:07:52

## Overview

TDMQ for RocketMQ can be accessed over the HTTP protocol from the private or public network. It is compatible with [HTTP SDKs](#) for multiple programming languages in the community.

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

### Note

Currently, transactional message and sequential message cannot be implemented over HTTP.

When creating a consumer group, you need to specify the type (TCP or HTTP, as described in [Group Management](#)); therefore, a consumer group does not support simultaneous consumption by TCP and HTTP clients.

## 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 imported dependencies through Maven and added SDK dependencies of the corresponding programming language in the pom.xml file.

For more examples, see the [demos](#) in the open-source community.

## Retry Mechanism

Every message consumed over HTTP will have an **invisibility time** of 5 minutes.

If the client acknowledges a message within the invisibility time, the consumption is successful and will not be retried.

If the client does not acknowledge a message after the invisibility time elapses, the message will become visible again, that is, the client will consume the message again subsequently.

Note that after the invisibility time of a message elapses during one consumption, the message handler will become invalid, and the message can no longer be acknowledged.

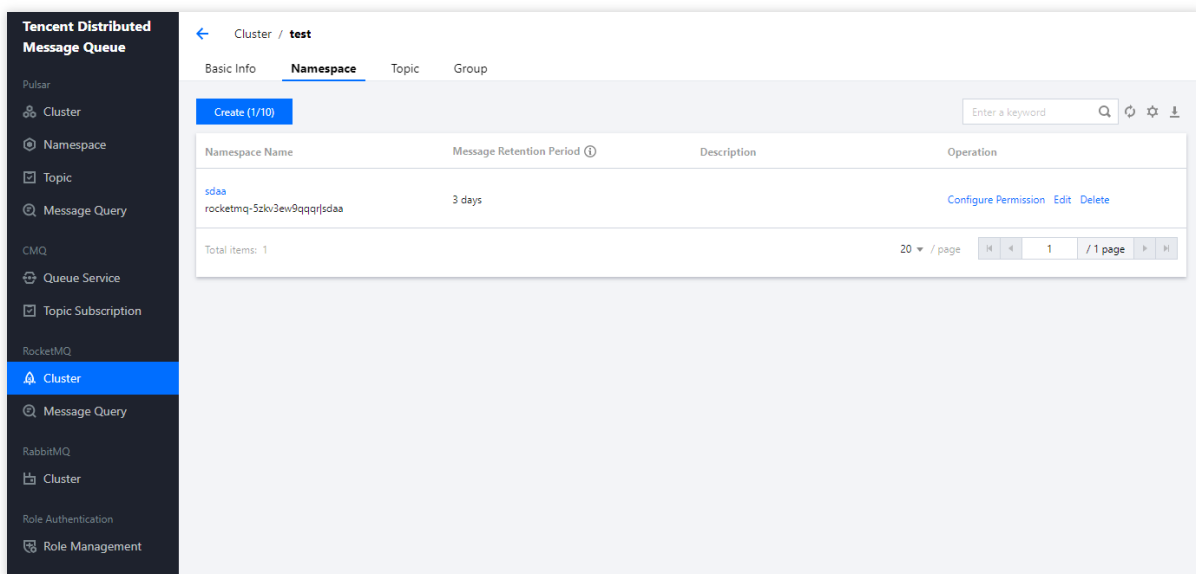
## Directions

## Step 1. Import dependencies

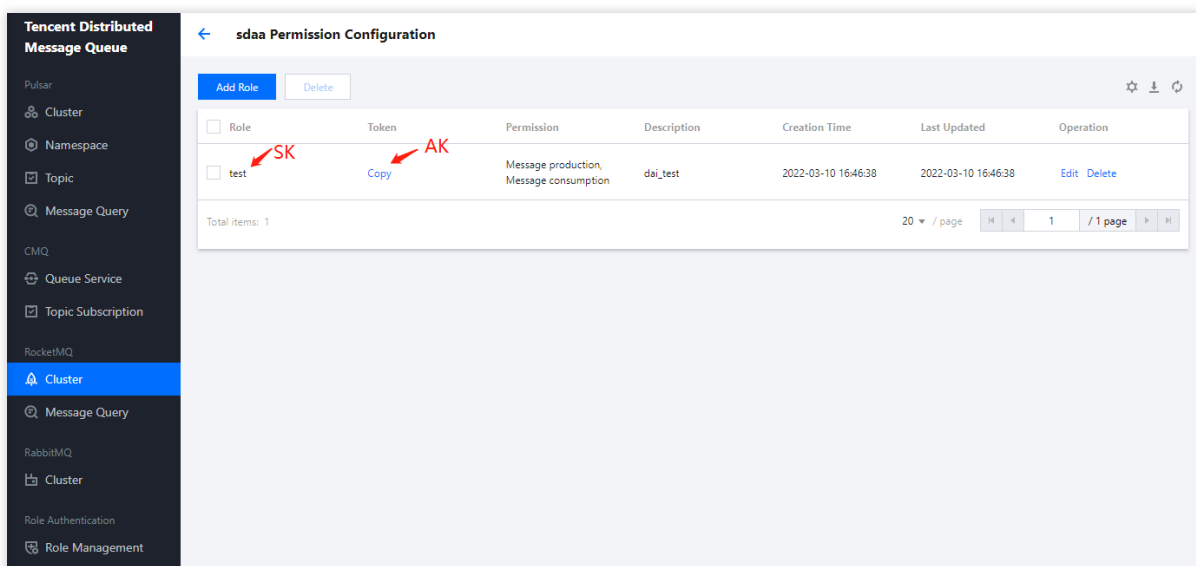
Import the SDK dependencies of the corresponding programming language into the pom.xml file of the project.

## Step 2. Get parameters

1. Log in to the TDMQ console, select the target cluster, and click the cluster name to enter the cluster details page.
2. Select the **Namespace** tab at the top and click **Configure Permission** on the right to enter the permission configuration page. If the role list is empty, click **Create** to create a role. For more information, see [Resource Creation and Preparation](#).



3. Copy the AK and SK on the page for use in next steps.

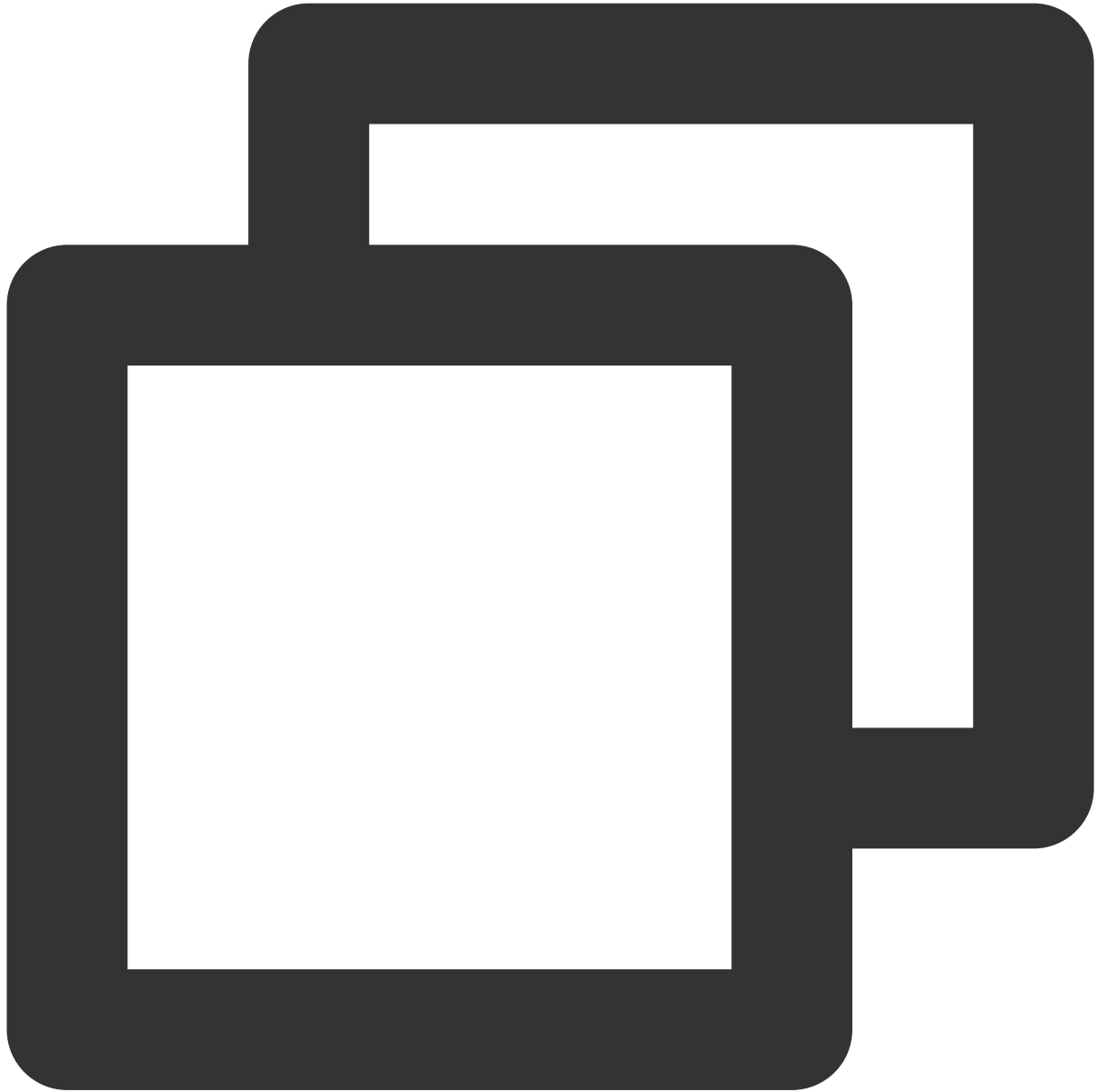


## Step 3. Initialize the producer client

JAVA

PHP

NodeJS



```
import com.aliyun.mq.http.MQClient;
import com.aliyun.mq.http.MQProducer;

public class Producer {

    public static void main(String[] args) {
        MQClient mqClient = new MQClient(
            // HTTP access point
            "${HTTP_ENDPOINT}",
```

```
        // Access key, which can be created and obtained in the TDMQ for Ro
        "${ACCESS_KEY}",
        // Role name, which can be created and obtained in the TDMQ for Roc
        "${SECRET_KEY}"
    );

    // The topic used for sending messages, which is required and can be obtain
    final String topic = "${TOPIC}";
    // The namespace of the topic, which is required and can be obtained in the
    final String instanceId = "${INSTANCE_ID}";

    // Create a producer
    MQProducer producer = mqClient.getProducer(instanceId, topic);

    // Send the message

    mqClient.close();
}
}
```



```
require "vendor/autoload.php";

use MQ\MQClient;

class ProducerTest
{
    private $client;
    private $producer;

    public function __construct()
    {
```

```
$this->client = new MQClient(  
    // HTTP access point  
    "${HTTP_ENDPOINT}",  
    // Access key, which can be created and obtained in the TDMQ for Rocket  
    "${ACCESS_KEY}",  
    // Role name, which can be created and obtained in the TDMQ for RocketM  
    "${SECRET_KEY}"  
);  
  
// The topic used for sending messages, which is required and can be obtain  
$topic = "${TOPIC}";  
// The namespace of the topic, which is required and can be obtained in the  
$instanceId = "${INSTANCE_ID}";  
  
$this->producer = $this->client->getProducer($instanceId, $topic);  
}  
  
public function run()  
{  
    // Send the message  
}  
}  
  
$instance = new ProducerTest();  
$instance->run();
```



```
const {
  MQClient,
  MessageProperties
} = require('@aliyunmq/mq-http-sdk');

// Set HTTP access endpoints
const endpoint = "{Endpoint}";
// AccessKey
const accessKeyId = "{Accesskey}";
// SecretKey
const accessKeySecret = "rop";
```

```
var client = new MQClient(endpoint, accessKeyId, accessKeySecret);

// Its Topic
const topic = "TopicA";
// ID of the instance to which the topic belongs
const instanceId = "MQ_INST_XXXXX";

const producer = client.getProducer(instanceId, topic);

(async function(){
  try {
    // Send 4 messages in a loop
    for(var i = 0; i < 4; i++) {
      let res;
      if (i % 2 == 0) {
        msgProps = new MessageProperties();
        // Set attributes
        msgProps.putProperty("key", i);
        // Set keys
        msgProps.messageKey("MessageKey");
        res = await producer.publishMessage("hello mq.", "", msgProps);
      } else {
        msgProps = new MessageProperties();
        // Set attributes
        msgProps.putProperty("key", i);
        // Timed message, with the time being 10s later
        msgProps.startDeliverTime(Date.now() + 10 * 1000);
        res = await producer.publishMessage("hello mq. timer msg!", "TagA", msgProp
      }
      console.log("Publish message: MessageID:%s,BodyMD5:%s", res.body.MessageId, r
    }
  } catch(e) {
    // The message failed to be sent and needs to be retried. You can resend this m
    console.log(e)
  }
}
```

#### Step 4. Initialize the consumer client

JAVA

PHP

NodeJS





```
import com.aliyun.mq.http.MQClient;
import com.aliyun.mq.http.MQConsumer;

public class Consumer {

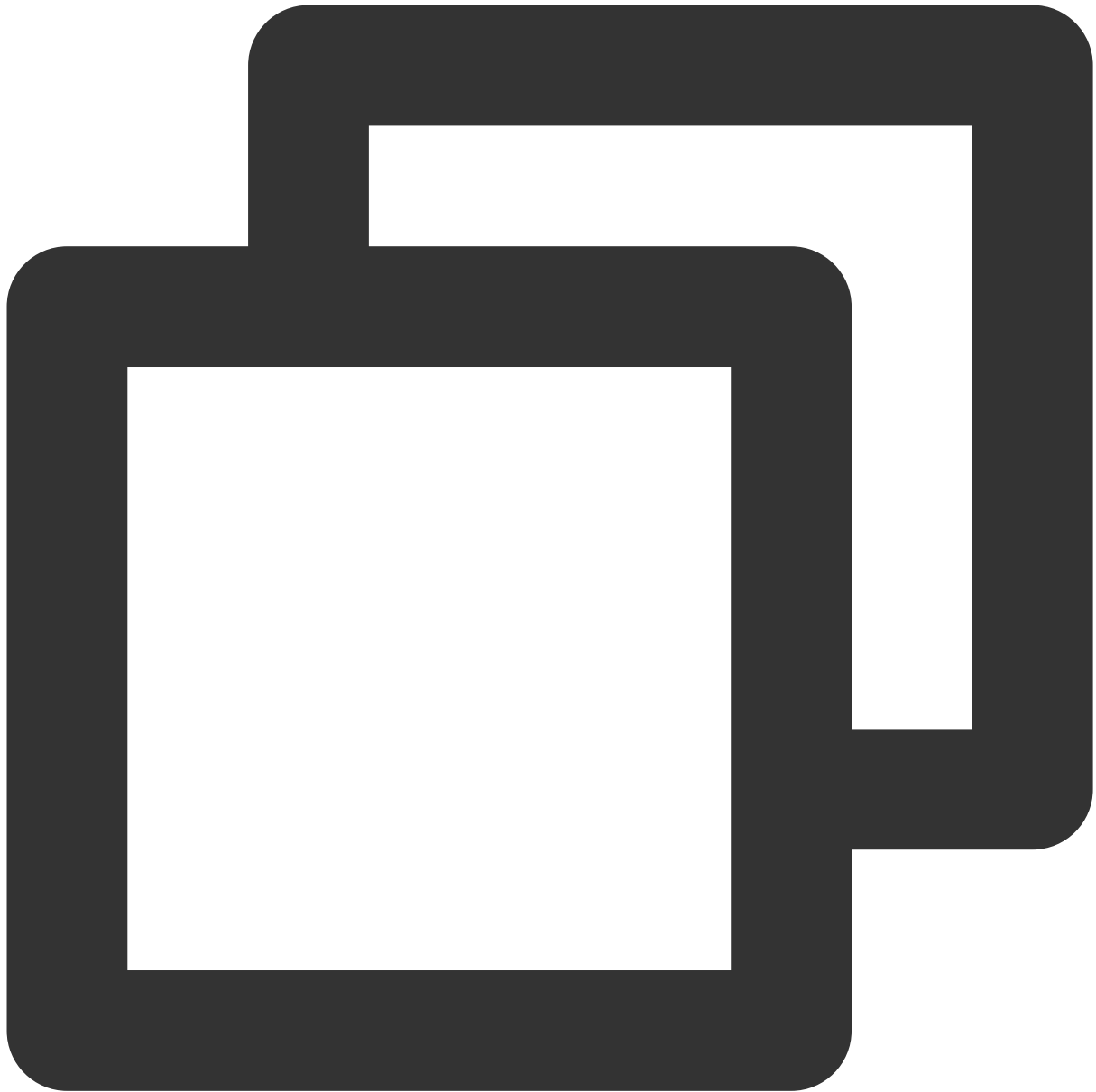
    public static void main(String[] args) {
        MQClient mqClient = new MQClient(
            // HTTP access point
            "${HTTP_ENDPOINT}",
            // Access key, which can be created and obtained in the TDMQ for Ro
            "${ACCESS_KEY}",
```

```
        // Role name, which can be created and obtained in the TDMQ for Roc
        "${SECRET_KEY}"
    );

    // The topic used for consuming messages, which is required and can be obta
    final String topic = "${TOPIC}";
    // Consumer group name, which is required and can be obtained in the TDMQ c
    final String groupId = "${GROUP_ID}";
    // The namespace of the topic, which is required and can be obtained in the
    final String instanceId = "${INSTANCE_ID}";

    final MQConsumer consumer = mqClient.getConsumer(instanceId, topic, groupId

    do {
        // Consume a message
    } while (true);
}
}
```



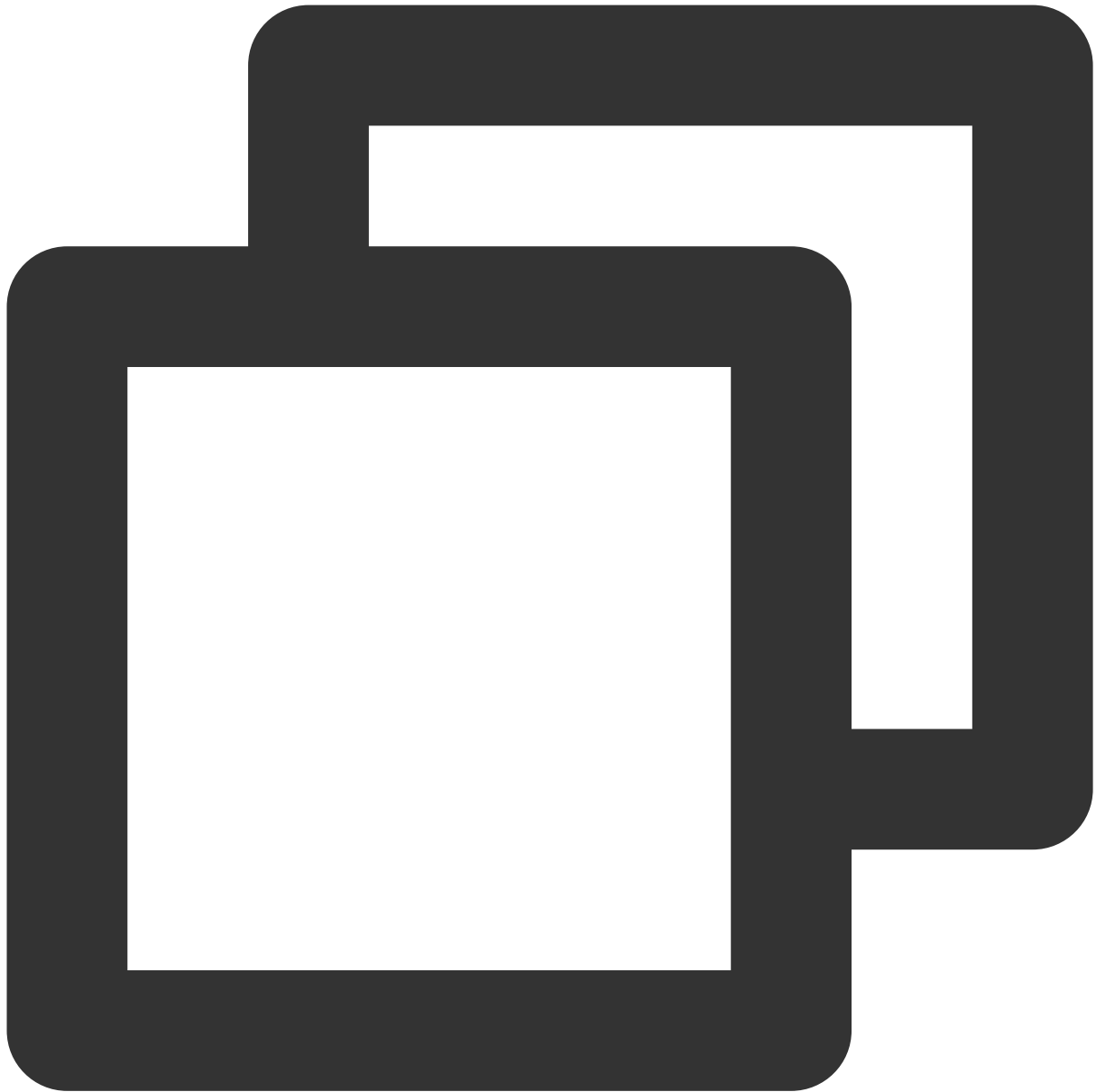
```
require "vendor/autoload.php";

use MQ\MQClient;

class ConsumerTest
{
    private $client;
    private $consumer;

    public function __construct()
    {
```

```
$this->client = new MQClient(  
    // HTTP access point  
    "${HTTP_ENDPOINT}",  
    // Access key, which can be created and obtained in the TDMQ for Rocket  
    "${ACCESS_KEY}",  
    // Role name, which can be created and obtained in the TDMQ for RocketM  
    "${SECRET_KEY}"  
);  
  
// The topic used for consuming messages, which is required and can be obta  
$topic = "${TOPIC}";  
// Consumer group name, which is required and can be obtained in the TDMQ c  
$groupId = "${GROUP_ID}";  
// The namespace of the topic, which is required and can be obtained in the  
$instanceId = "${INSTANCE_ID}";  
  
$this->consumer = $this->client->getConsumer($instanceId, $topic, $groupId)  
}  
  
public function run()  
{  
    while (True) {  
        // Consume a message  
    }  
}  
}  
  
$instance = new ConsumerTest();  
$instance->run();
```



```
const {
  MQClient
} = require('@aliyunmq/mq-http-sdk');

// Set HTTP access endpoints
const endpoint = "{Endpoint}";
// AccessKey
const accessKeyId = "{Accesskey}";
// SecretKey
const accessKeySecret = "rop";
```

```
var client = new MQClient(endpoint, accessKeyId, accessKeySecret);

// Its Topic
const topic = "TopicA";
// ID of the instance to which the topic belongs
const instanceId = "MQ_INST_xxxxx";
// The consumer group you created in the console
const groupId = "GID_xxx";

const consumer = client.getConsumer(instanceId, topic, groupId);

(async function(){
  // Consume messages in loop
  while(true) {
    try {
      // long polling of consumption messages
      // Long polling means that if the topic has no messages, the request will han
      res = await consumer.consumeMessage(
        3, // This indicates a maximum of 3 messages can be consumed at a time. U
        3 // Long polling lasts 3 seconds, which can be set up to 30 seconds.
      );

      if (res.code == 200) {
        // Consume messages based on business processing logic
        console.log("Consume Messages, requestId:%s", res.requestId);
        const handles = res.body.map((message) => {
          console.log("\t\tMessageId:%s, Tag:%s, PublishTime:%d, NextConsumeTime:%d, Fir
            ", Props:%j, MessageKey:%s, Prop-A:%s",
              message.MessageId, message.MessageTag, message.PublishTime, message.N
              message.MessageBody, message.Properties, message.MessageKey, message.Pro
          return message.ReceiptHandle;
        });

        // If a message is not acked for successful consumption before `message.Nex
        // The message handle has a timestamp that changes each time the same messa
        res = await consumer.ackMessage(handles);
        if (res.code != 204) {
          // The handle of some messages may time out, which will cause the acknowl
          console.log("Ack Message Fail:");
          const failHandles = res.body.map((error)=>{
            console.log("\t\tErrorHandle:%s, Code:%s, Reason:%s\n", error.ReceiptHa
            return error.ReceiptHandle;
          });
          handles.forEach((handle)=>{
            if (failHandles.indexOf(handle) < 0) {
              console.log("\t\tSucHandle:%s\n", handle);
            }
          }
        }
      }
    }
  }
})();
```

```
    });  
  } else {  
    // The message is acked for successful consumption  
    console.log("Ack Message suc, RequestId:%s\\n\\t", res.requestId, handles  
  }  
}  
} catch(e) {  
  if (e.Code.indexOf("MessageNotExist") > -1) {  
    // If there is no message, long polling will continue on the server.  
    console.log("Consume Message: no new message, RequestId:%s, Code:%s", e.Req  
  } else {  
    console.log(e);  
  }  
}  
}  
}  
}) ();
```

# Access over HTTP

## SDK for Java

### Sending and Receiving General Messages

Last updated : 2023-09-13 11:36:59

## Overview

TDMQ for RocketMQ can be accessed over the HTTP protocol from the private or public network. It is compatible with [HTTP SDKs](#) for multiple programming languages in the community.

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

### Note

Currently, transactional message cannot be implemented over HTTP.

As a consumer group does not support simultaneous consumption by TCP and HTTP clients, you need to specify the type (TCP or HTTP) when creating a consumer group. For more information, see [Group Management](#).

## 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 imported dependencies through Maven and added SDK dependencies of the corresponding programming language in the pom.xml file.

For more examples, see the [demos](#) in the open-source community.

## Retry Mechanism

A fixed retry interval is used in HTTP, which can't be customized currently.

Message Type	Retry Interval	Maximum Number of Retries
General Message	5 minutes	288
Sequential message	1 minute	288



**Note**

If the client acknowledges a message within the retry interval, the message consumption is successful and will not be retried.

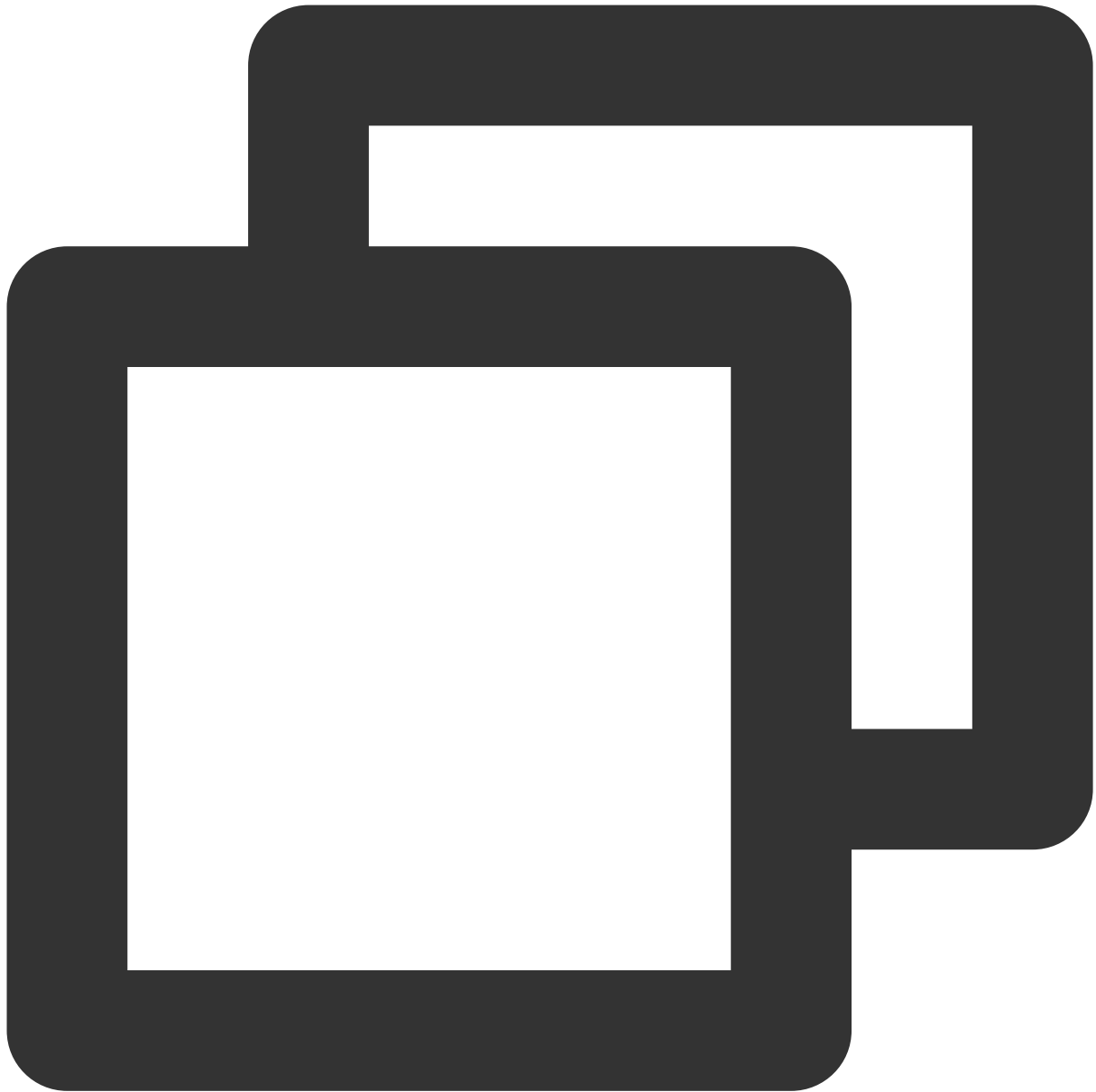
If the client doesn't acknowledge a message after the retry interval has expired, the message will become visible again, and the client will consume it again.

The message handle consumed each time is only valid within the retry interval, and become invalid after that time period.

## Directions

**Step 1. Install the Java dependent library**

Introduce dependencies in a Java project and add the following dependencies to the `pom.xml` file. This document uses a Maven project as an example.

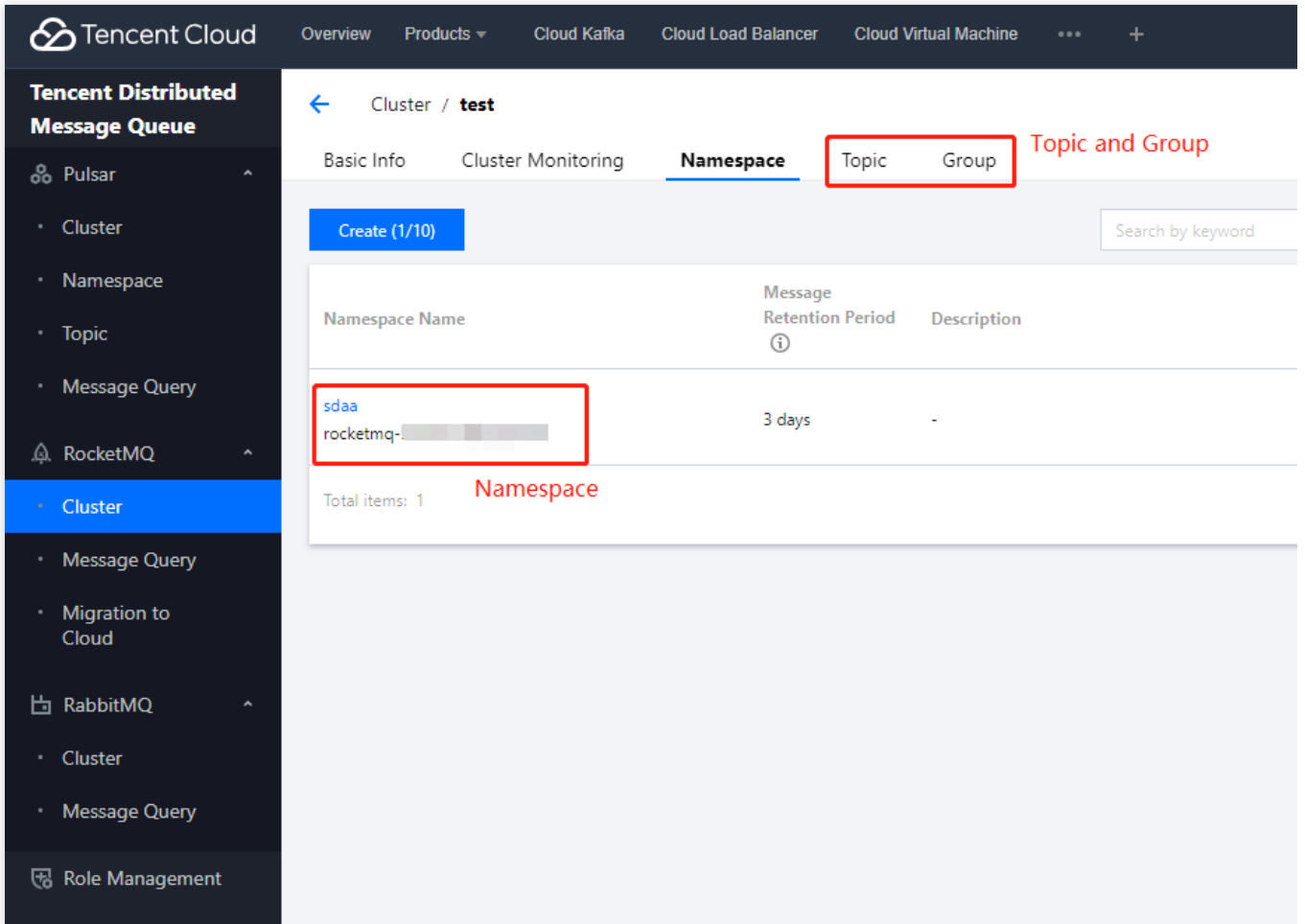


```
<!-- in your <dependencies> block -->
<dependency>
  <groupId>com.aliyun.mq</groupId>
  <artifactId>mq-http-sdk</artifactId>
  <version>1.0.3</version>
</dependency>
```

## Step 2. Get parameters




1. Log in to the TDMQ console, select the target cluster, and click the cluster name to enter the cluster details page.

2. Select the **Namespace** tab at the top and click **Configure Permission** on the right to enter the permission configuration page. If the role list is empty, click **Create** to create a role. For more information, see [Resource Creation and Preparation](#).



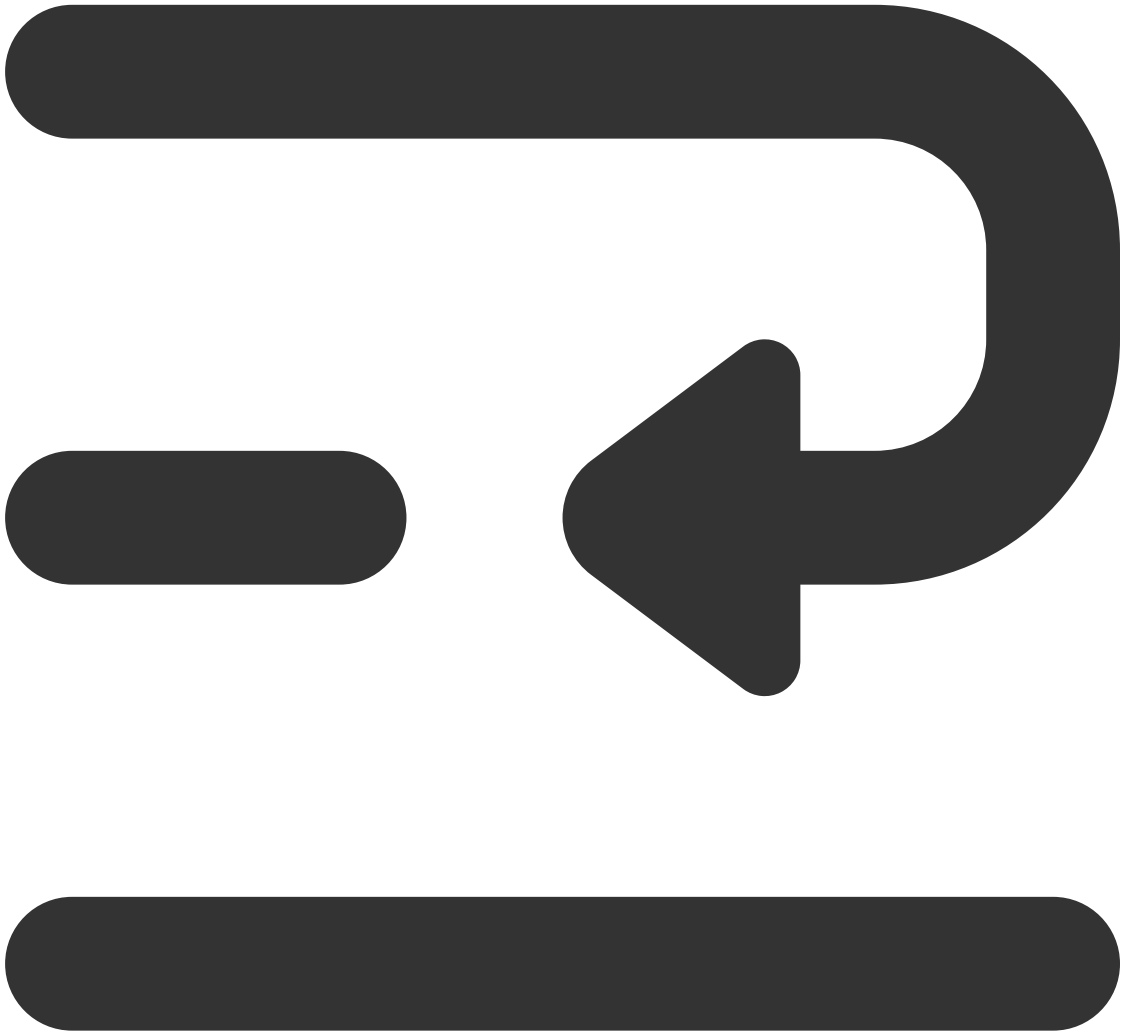
3. Copy the AK and SK on the page for use in next steps.

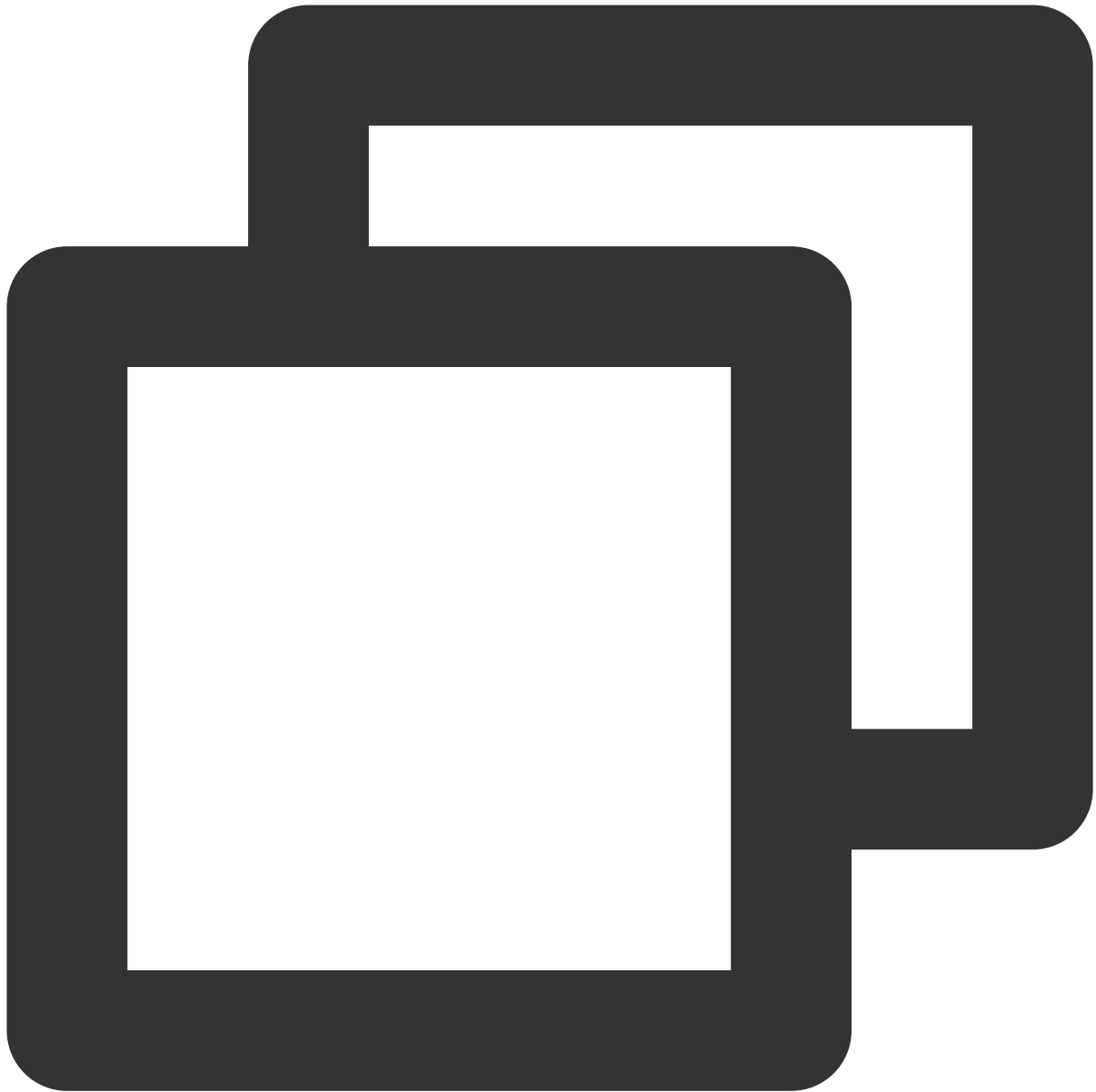
The screenshot shows the 'sdaa Permission Configuration' page in the Tencent Cloud console. The left sidebar is expanded to 'RocketMQ' > 'Cluster'. The main content area displays a table of roles with columns for Role(secretKey), Token(accessKey), Permission, Description, and Creation Time. There are three roles listed. The first role has a red arrow pointing to 'SK' and another pointing to 'AK' in the Token column. The second role has 'Copy' in the Token column. The third role has 'Copy' in the Token column and 'dai\_testa' in the Description column. Below the table, it says 'Total items: 3'. There are 'Add Role' and 'Delete' buttons at the top of the table area.

<input type="checkbox"/>	Role(secretKey)	Token(accessKey)	Permission	Description	Creation Time
<input type="checkbox"/>		Copy <i>AK</i>	Message production, Message consumption	-	2023-06-28 13
<input type="checkbox"/>		Copy	Message production, Message consumption	-	2023-04-26 16
<input type="checkbox"/>		Copy	Message production, Message consumption	dai_testa	2022-03-10 16

### Step 3. Produce messages

#### Creating a message producer

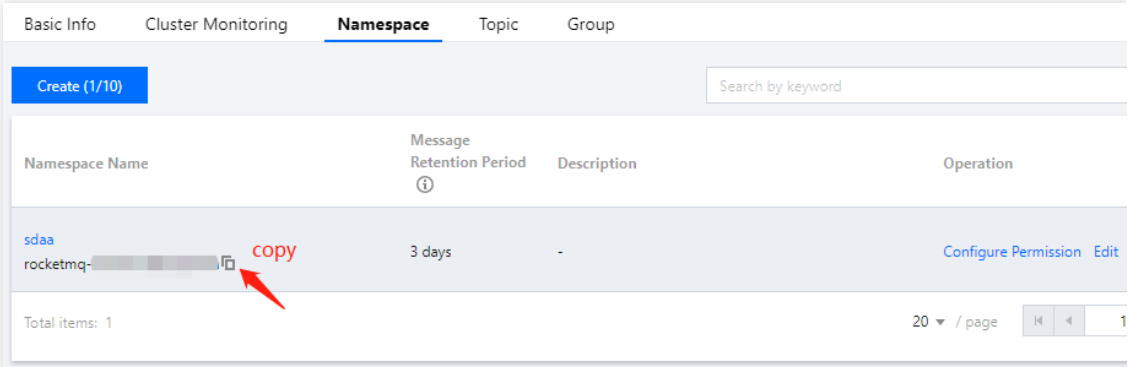
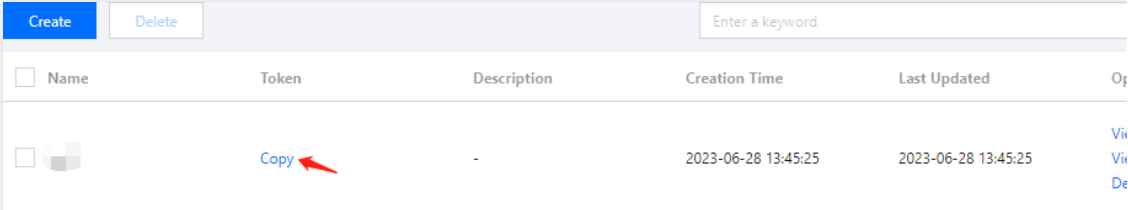




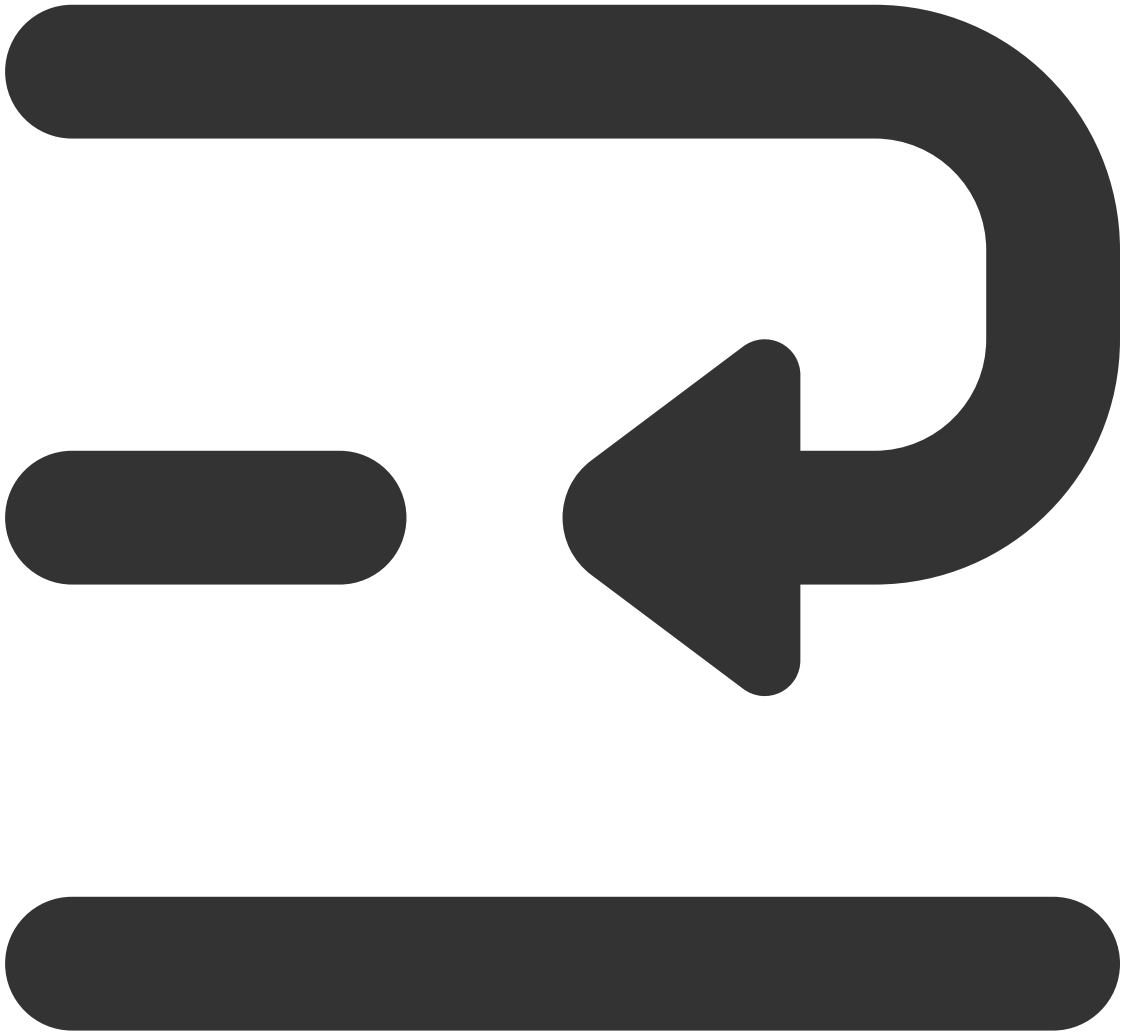
```
// Get the client
MQClient mqClient = new MQClient(endpoint, accessKey, secretKey);

// Get the topic producer
MQProducer producer = mqClient.getProducer(namespace, topicName);
```

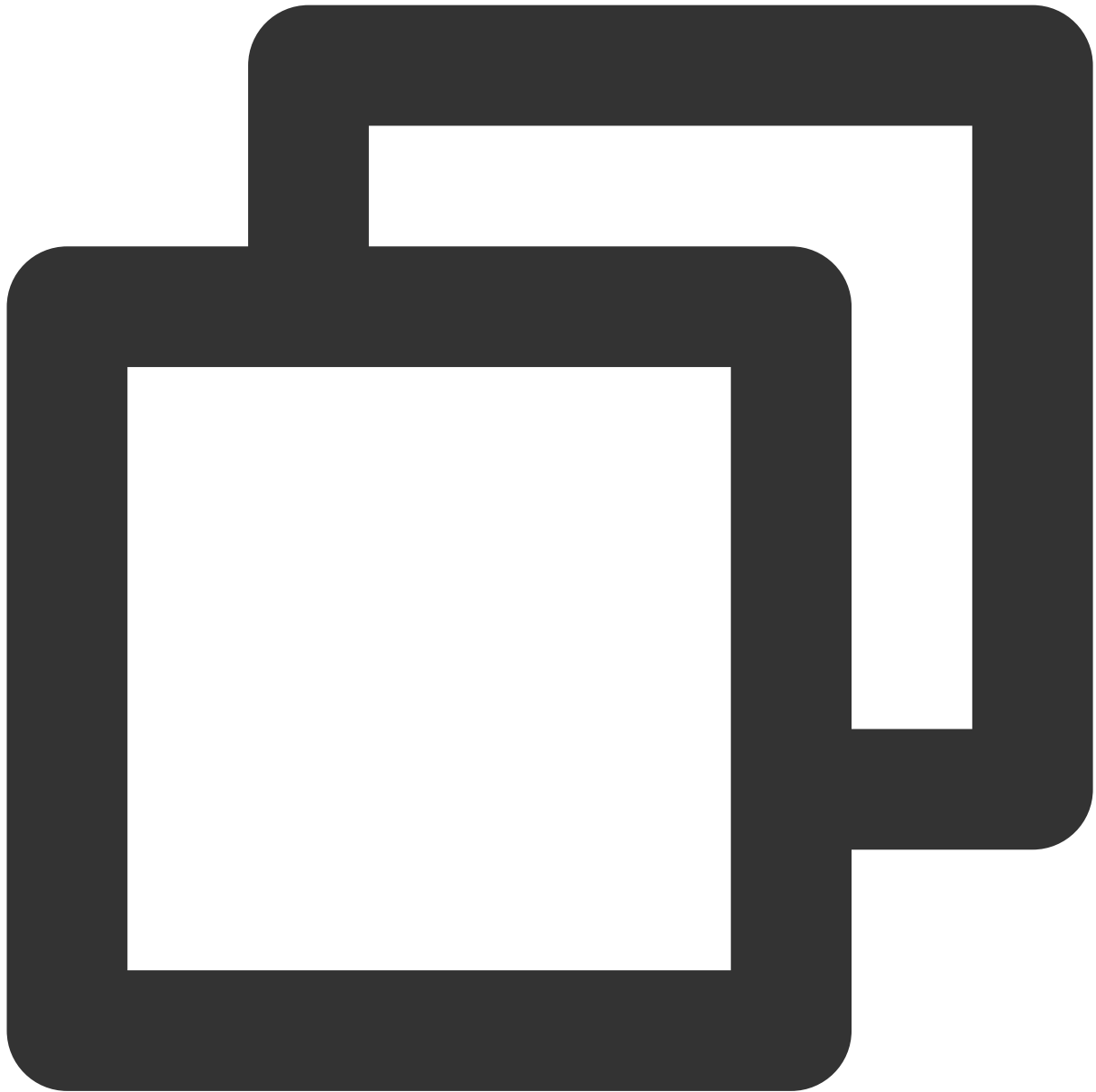
Parameter	Description
topicName	Topic name, which can be copied under the <b>Topic</b> tab on the <b>Cluster</b> page in the console.
namespace	Namespace name, which can be copied under the <b>Namespace</b> tab on the <b>Cluster</b> page in the console.

	
<p>endpoint</p>	<p>Cluster access address over HTTP, which can be obtained from <b>Access Address</b> in the <b>Operation Cluster</b> page in the console.</p>
<p>secretKey</p>	<p>Role name, which can be copied on the <a href="#">Role Management</a> page.</p>
<p>accessKey</p>	<p>Role token, which can be copied in the <b>Token</b> column on the <a href="#">Role Management</a> page.</p> 

**Sending a message**







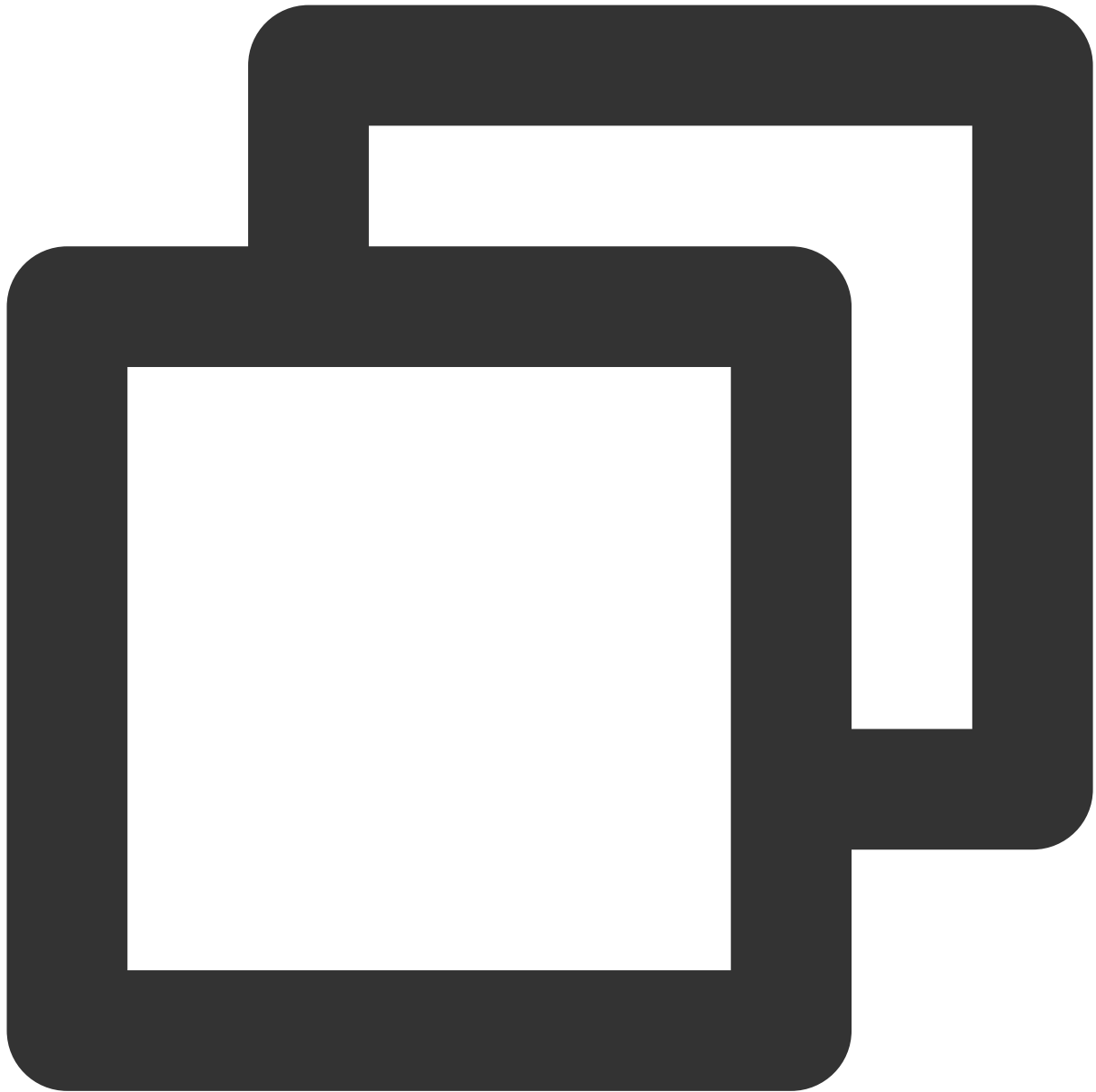
```
try {
    for (int i = 0; i < 10; i++) {
        TopicMessage pubMsg;
        pubMsg = new TopicMessage(
            ("Hello RocketMQ " + i).getBytes(),
            "TAG"
        );
        TopicMessage pubResultMsg = producer.publishMessage(pubMsg);
        System.out.println("Send mq message success. MsgId is: " + pubResultMsg.get
    }
} catch (Throwable e) {
```

```
System.out.println("Send mq message failed.");  
e.printStackTrace();  
}
```

Parameter	Description
TAG	Set the message tag.

## Step 4. Consume messages

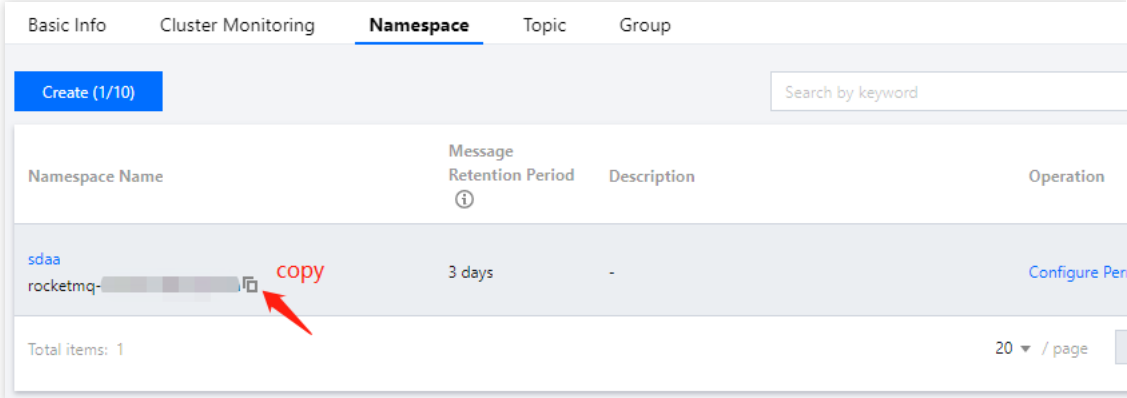
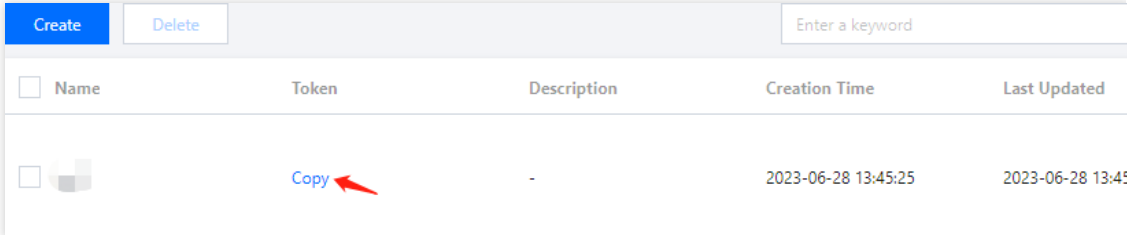
### Creating a consumer



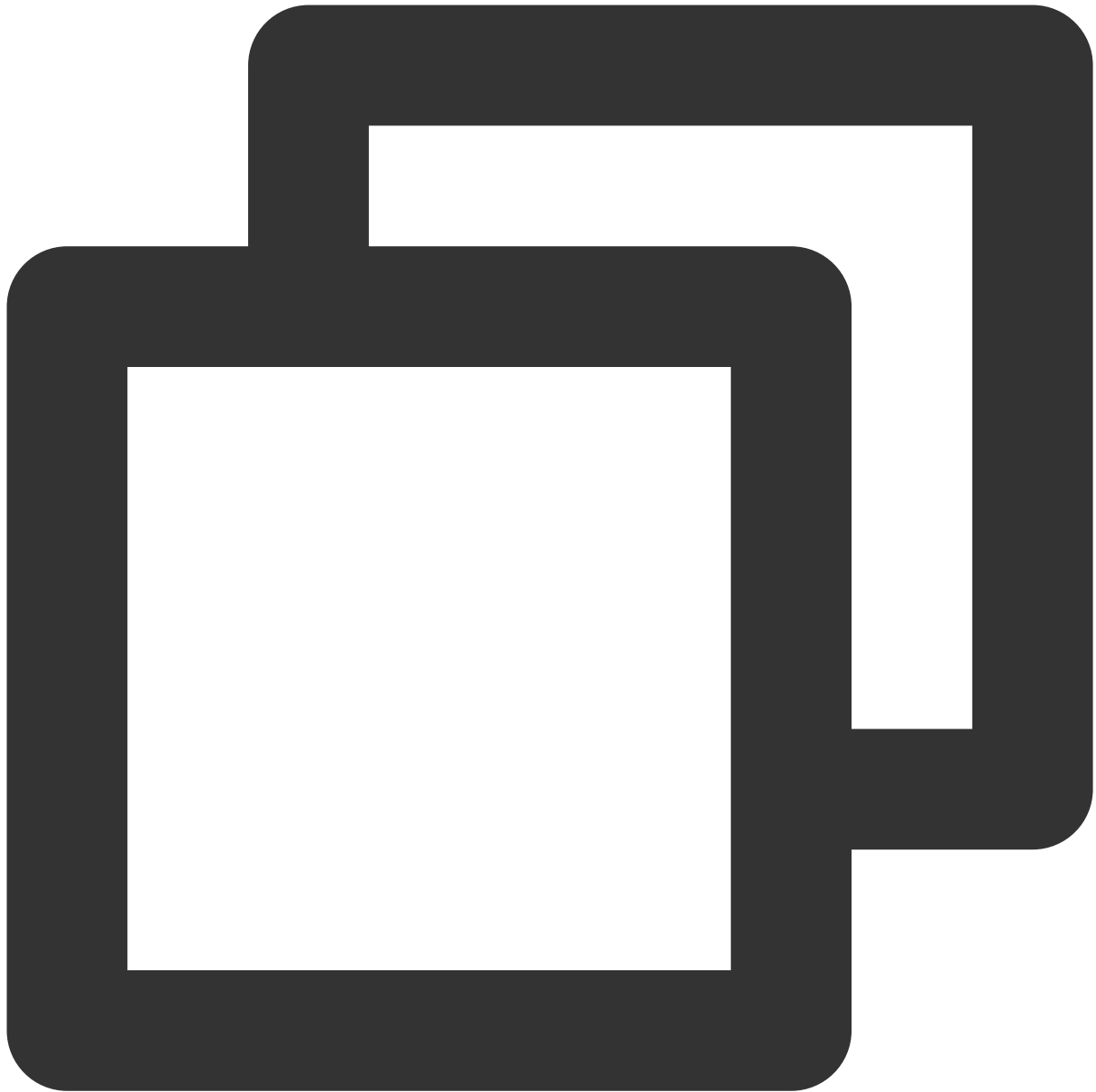
```
// Get the client
MQClient mqClient = new MQClient(endpoint, accessKey, secretKey);

// Get the topic consumer
MQConsumer consumer = mqClient.getConsumer(namespace, topicName, groupName, "TAG
```

Parameter	Description
topicName	Topic name, which can be copied under the <b>Topic</b> tab on the <b>Cluster</b> page in the console.
groupName	Producer group name, which can be copied under the <b>Group</b> tab on the <b>Cluster</b> page in the console.

<p>namespace</p>	<p>Namespace name, which can be copied under the <b>Namespace</b> tab on the <b>Cluster</b> page in the console.</p> 
<p>TAG</p>	<p>Subscribed tag.</p>
<p>endpoint</p>	<p>Cluster access address over HTTP, which can be obtained from <b>Access Address</b> in the <b>Operation</b> page in the console.</p>
<p>secretKey</p>	<p>Role name, which can be copied on the <a href="#">Role Management</a> page.</p>
<p>accessKey</p>	<p>Role token, which can be copied in the <b>Token</b> column on the <a href="#">Role Management</a> page.</p> 

**Subscribing to messages**



```
do {
    List<Message> messages = null;

    try {
        // long polling of consumption messages
        // Long polling means that if the topic has no messages, the request will w
        messages = consumer.consumeMessage(
            Integer.parseInt(batchSize),
            Integer.parseInt(waitSeconds)
        );
    } catch (Throwable e) {
```

```
e.printStackTrace();
}
if (messages == null || messages.isEmpty()) {
    System.out.println(Thread.currentThread().getName() + ": no new message, continue;");
}

for (Message message : messages) {
    System.out.println("Receive message: " + message);
}

{
    List<String> handles = new ArrayList<String>();
    for (Message message : messages) {
        handles.add(message.getReceiptHandle());
    }

    try {
        consumer.ackMessage(handles);
    } catch (Throwable e) {
        if (e instanceof AckMessageException) {
            AckMessageException errors = (AckMessageException) e;
            System.out.println("Ack message fail, requestId is:" + errors.getRequestId());
            if (errors.getErrorMessages() != null) {
                for (String errorHandle : errors.getErrorMessages().keySet()) {
                    System.out.println("Handle:" + errorHandle + ", ErrorCode:"
                        + errors.getErrorMessages().get(errorHandle)
                        + ", ErrorMsg:" + errors.getErrorMessages().get(errorHandle));
                }
            }
            continue;
        }
        e.printStackTrace();
    }
}
} while (true);
```

Parameter	Description
batchSize	The number of messages pulled at a time. Maximum value: 16.
waitSeconds	The polling waiting time for a message pull. Maximum value: 30 seconds.

# Sending and Receiving Sequential Messages

Last updated : 2023-09-13 11:37:45

## Overview

TDMQ for RocketMQ can be accessed over the HTTP protocol from the private or public network. It is compatible with [HTTP SDKs](#) for multiple programming languages in the community.

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

### Note

Currently, transactional message cannot be implemented over HTTP.

As a consumer group does not support simultaneous consumption by TCP and HTTP clients, you need to specify the type (TCP or HTTP) when creating a consumer group. For more information, see [Group Management](#).

## 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 imported dependencies through Maven and added SDK dependencies of the corresponding programming language in the pom.xml file.

For more examples, see the [demos](#) in the open-source community.

## Retry Mechanism

A fixed retry interval is used in HTTP, which can't be customized currently.

Message Type	Retry Interval	Maximum Number of Retries
General Message	5 minutes	288
Sequential message	1 minute	288

### Note

If the client acknowledges a message within the retry interval, the message consumption is successful and will not be retried.

If the client doesn't acknowledge a message after the retry interval has expired, the message will become visible again, and the client will consume it again.

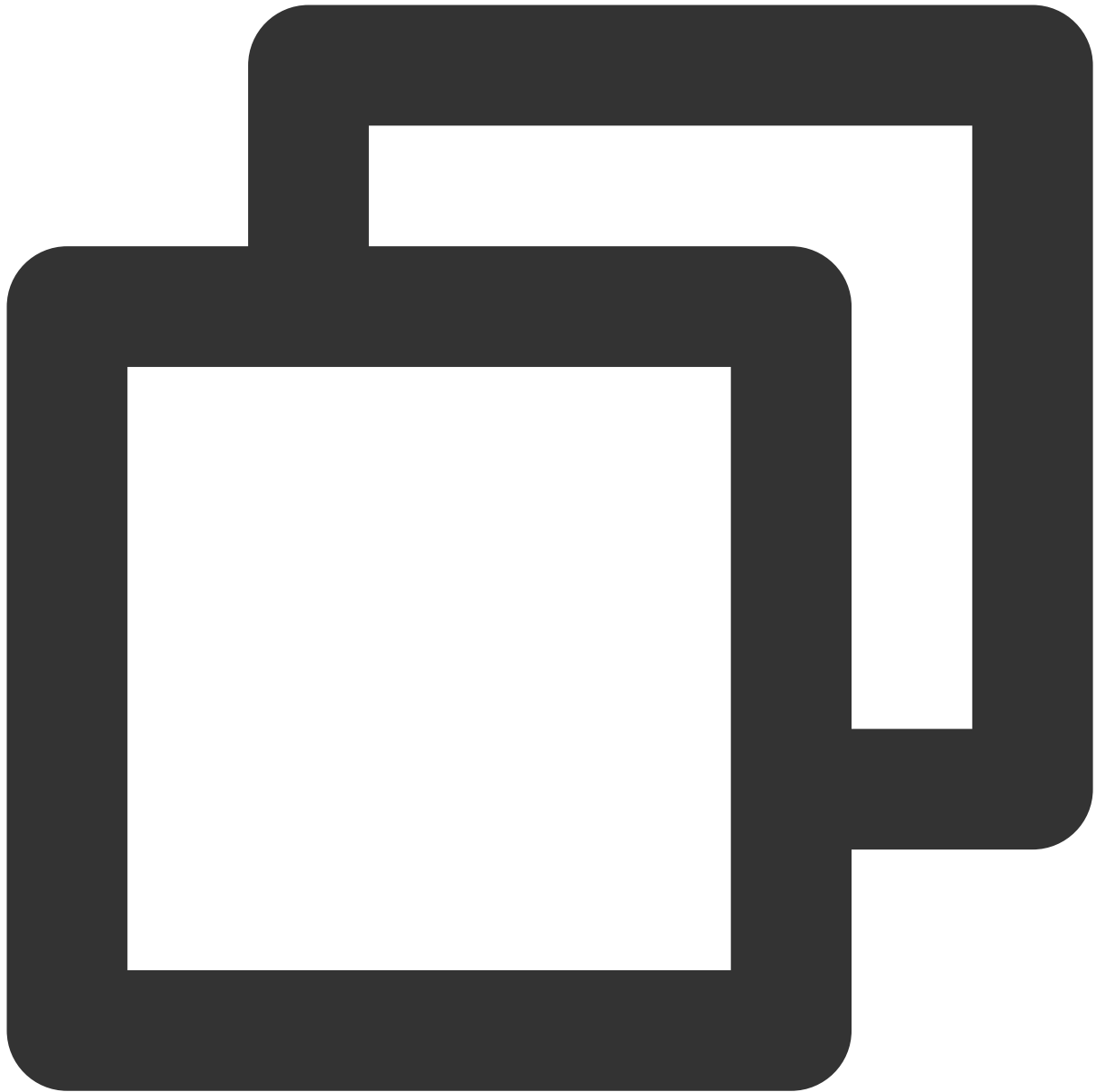
The message handle consumed each time is only valid within the retry interval, and become invalid after that time period.

## Directions

### Step 1. Install the Java dependent library

Introduce dependencies in a Java project and add the following dependencies to the `pom.xml` file. This document uses a Maven project as an example.



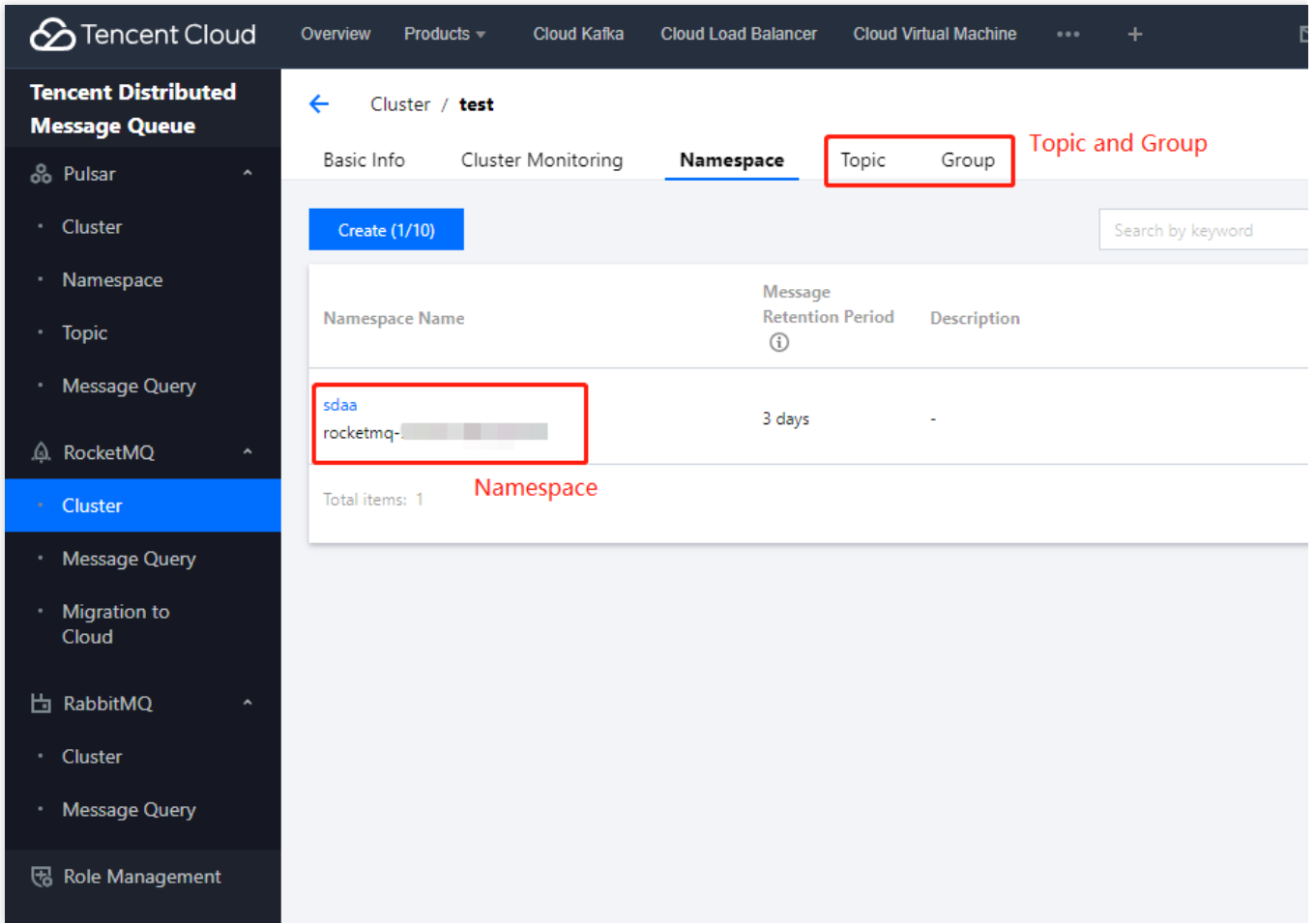


```
<!-- in your <dependencies> block -->
<dependency>
  <groupId>com.aliyun.mq</groupId>
  <artifactId>mq-http-sdk</artifactId>
  <version>1.0.3</version>
</dependency>
```

## Step 2. Get parameters




1. Log in to the TDMQ console, select the target cluster, and click the cluster name to enter the cluster details page.

2. Select the **Namespace** tab at the top and click **Configure Permission** on the right to enter the permission configuration page. If the role list is empty, click **Create** to create a role. For more information, see [Resource Creation and Preparation](#).



3. Copy the AK and SK on the page for use in next steps.

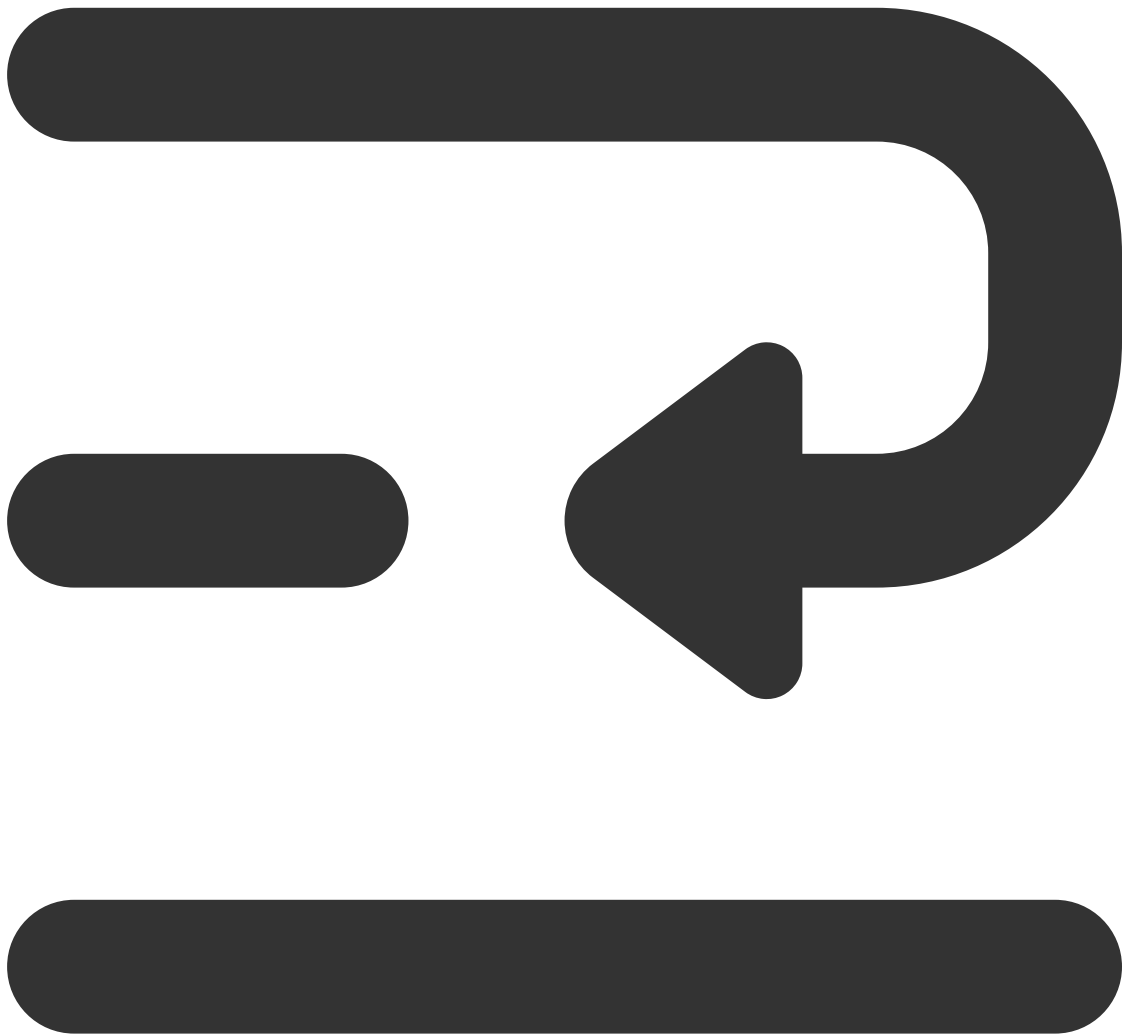
The screenshot shows the 'sdaa Permission Configuration' page in the Tencent Cloud console. The left sidebar is expanded to 'RocketMQ' > 'Cluster'. The main content area displays a table of roles with columns for Role(secretKey), Token(accessKey), Permission, Description, and Creation Time. There are three roles listed. Red arrows point to 'SK' and 'AK' labels above the first row's secret key and access key fields. The 'Add Role' and 'Delete' buttons are visible at the top of the table.

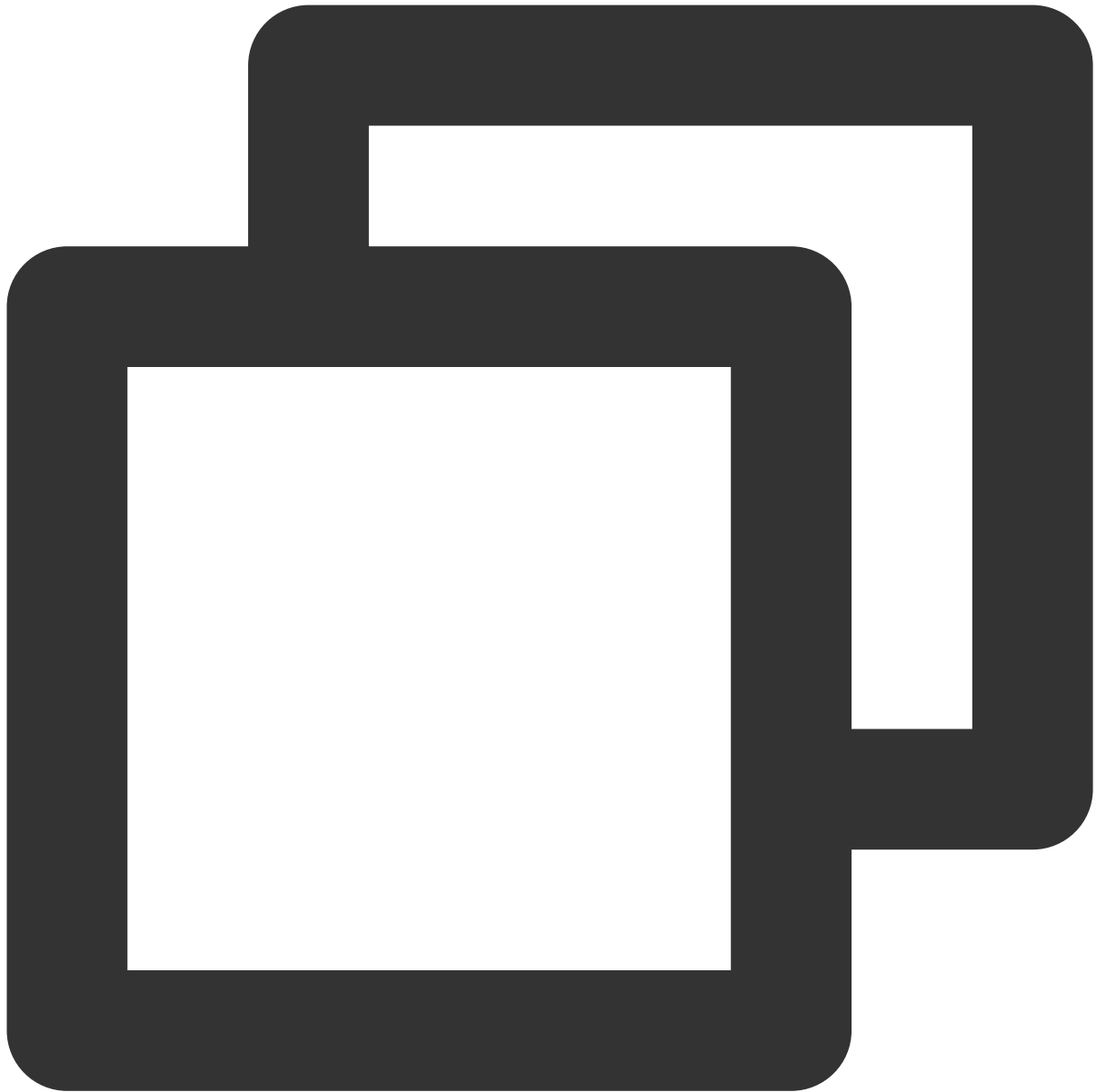
<input type="checkbox"/>	Role(secretKey)	Token(accessKey)	Permission	Description	Creation Time
<input type="checkbox"/>	 <b>SK</b>	<a href="#">Copy</a> <b>AK</b>	Message production, Message consumption	-	2023-06-28 13
<input type="checkbox"/>		<a href="#">Copy</a>	Message production, Message consumption	-	2023-04-26 16
<input type="checkbox"/>		<a href="#">Copy</a>	Message production, Message consumption	dai_testa	2022-03-10 16

Total items: 3

### Step 3. Produce messages

#### Creating a message producer

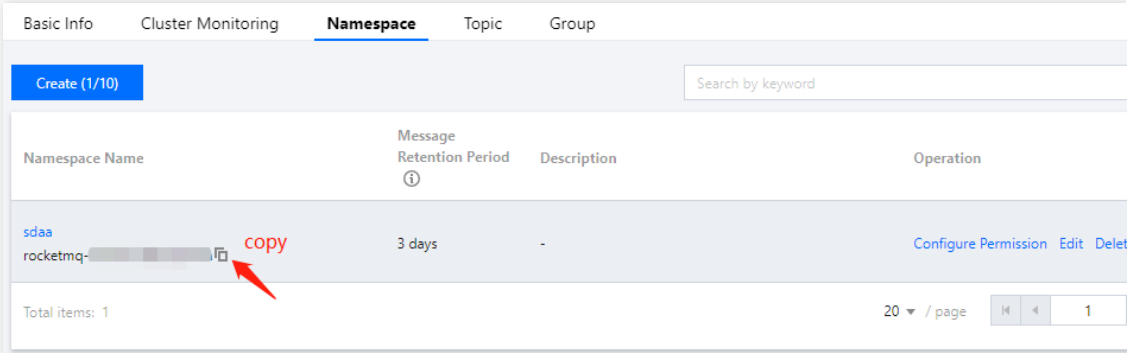
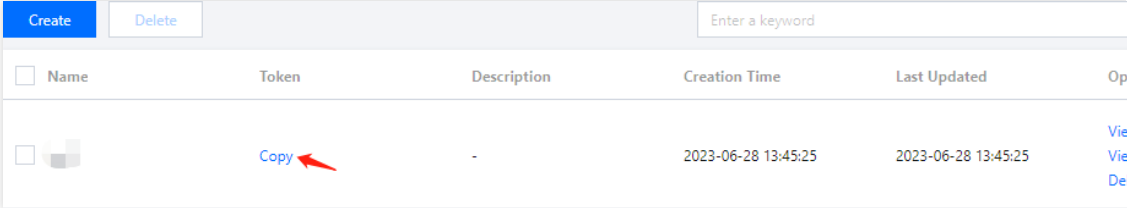




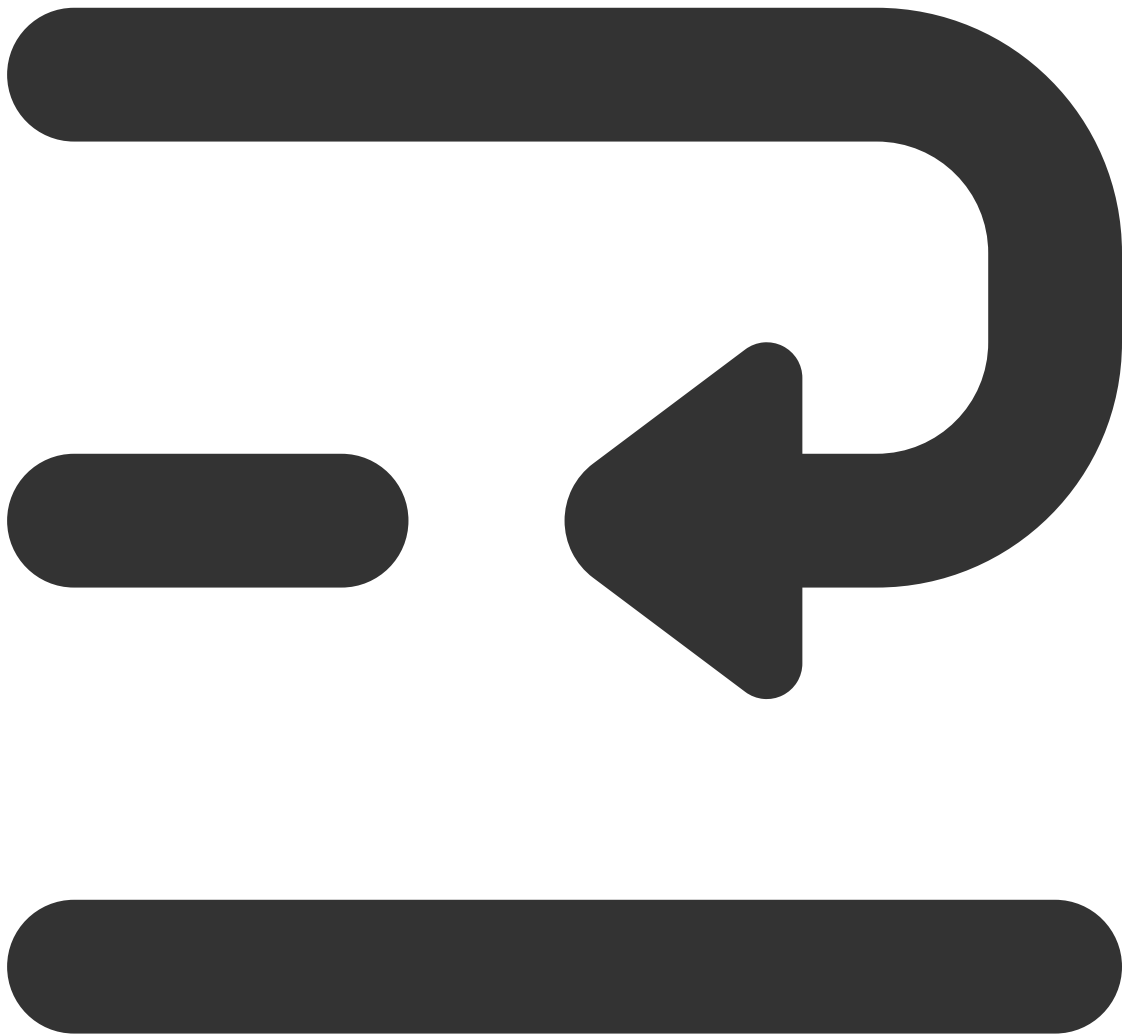
```
// Get the client
MQClient mqClient = new MQClient(endpoint, accessKey, secretKey);

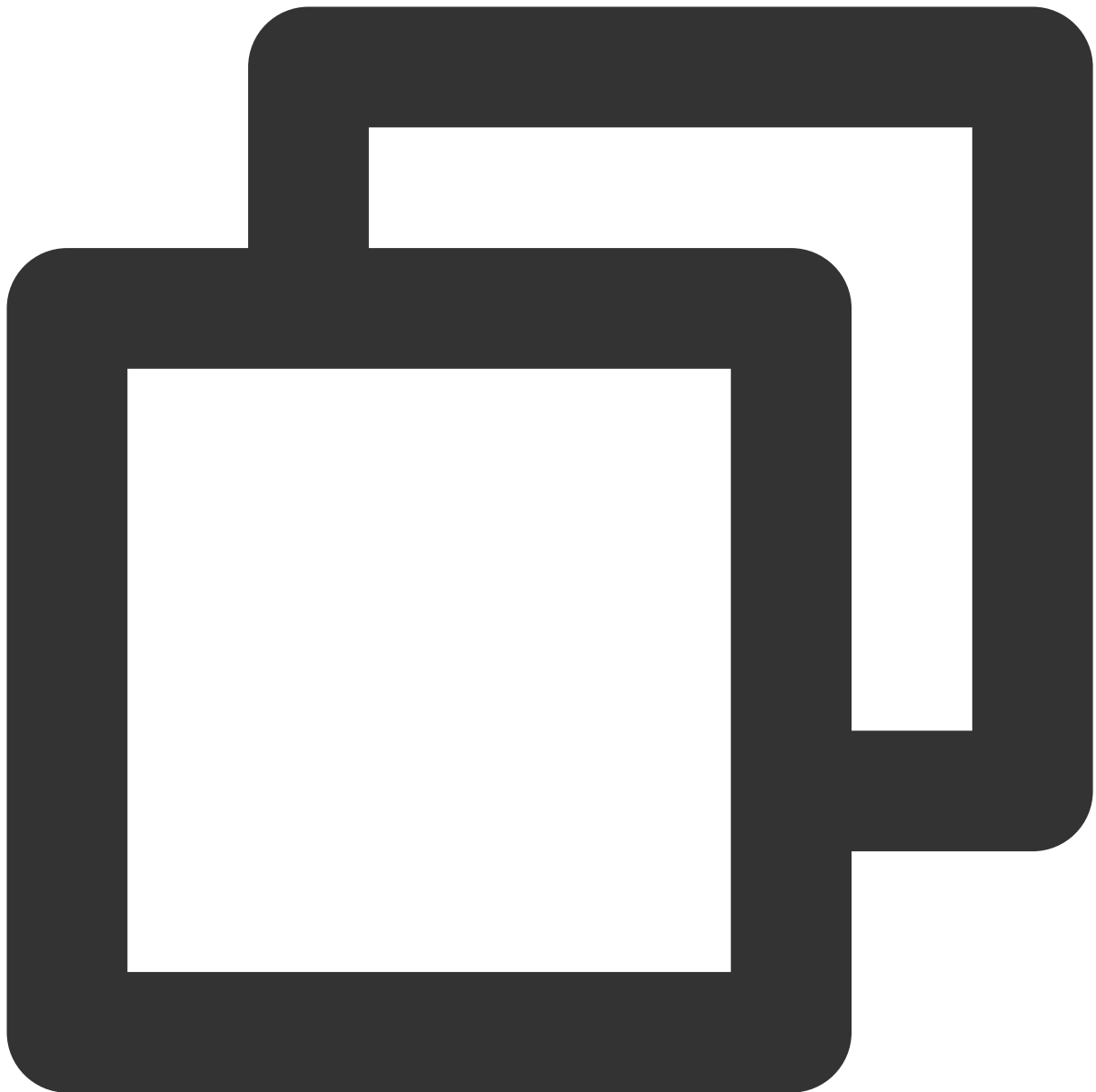
// Get the topic producer
MQProducer producer = mqClient.getProducer(namespace, topicName);
```

Parameter	Description
topicName	Topic name, which can be copied under the <b>Topic</b> tab on the <b>Cluster</b> page in the console.
namespace	Namespace name, which can be copied under the <b>Namespace</b> tab on the <b>Cluster</b> page in the console.

	
<p>endpoint</p>	<p>Cluster access address over HTTP, which can be obtained from <b>Access Address</b> in the <b>Operation Cluster</b> page in the console.</p>
<p>secretKey</p>	<p>Role name, which can be copied on the <a href="#">Role Management</a> page.</p>
<p>accessKey</p>	<p>Role token, which can be copied in the <b>Token</b> column on the <a href="#">Role Management</a> page.</p> 

**Sending a message**





```
try {
    for (int i = 0; i < 10; i++) {
        TopicMessage pubMsg;
        pubMsg = new TopicMessage(
            ("Hello RocketMQ " + i).getBytes(),
            "TAG"
        );
        // Set the ShardingKey of the partitionally sequential message
        pubMsg.setShardingKey(i % 3);
        TopicMessage pubResultMsg = producer.publishMessage(pubMsg);
        System.out.println("Send mq message success. MsgId is: " + pubResultMsg.get
```

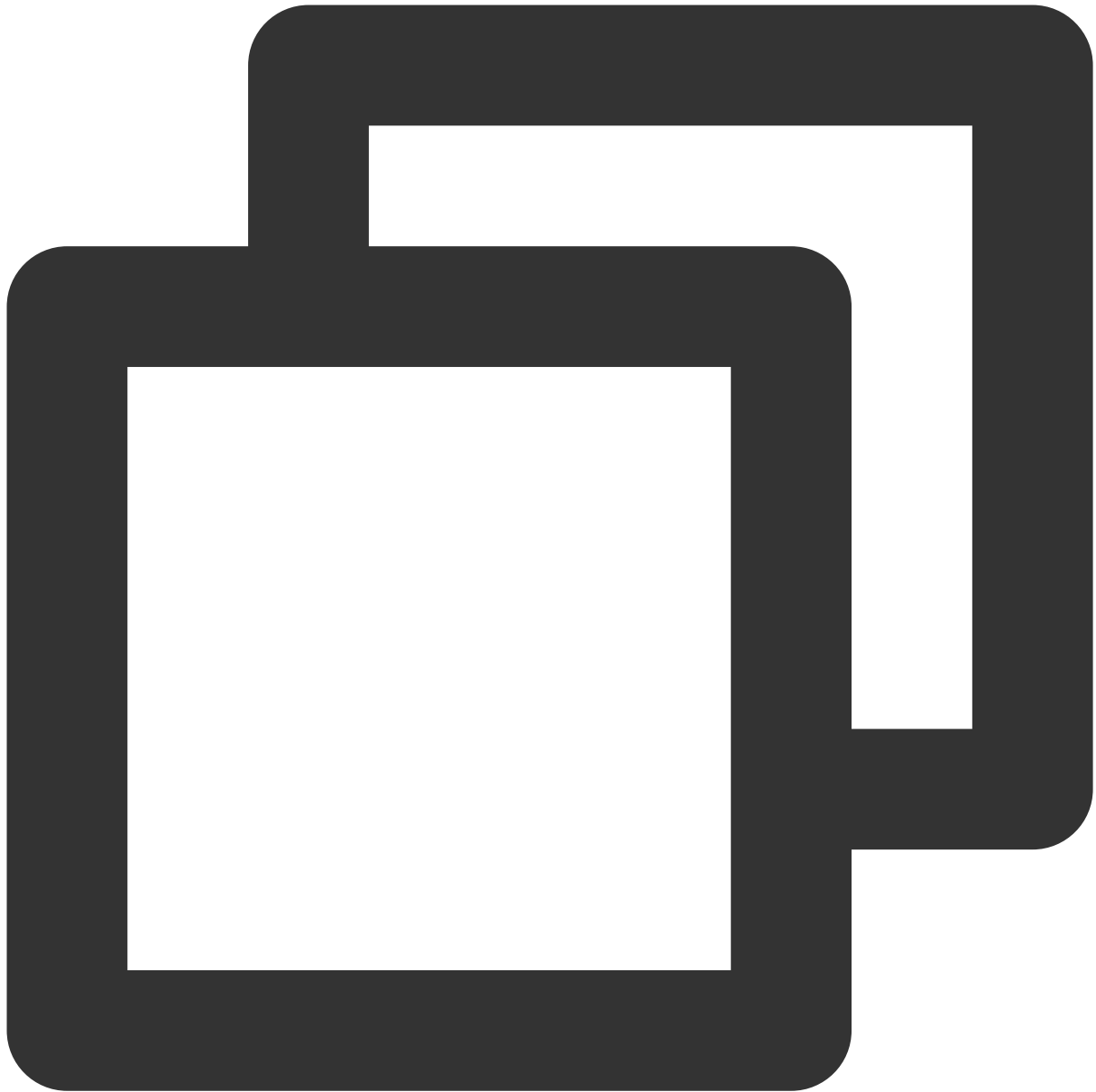


```
    }  
    } catch (Throwable e) {  
        System.out.println("Send mq message failed.");  
        e.printStackTrace();  
    }  
}
```

Parameter	Description
TAG	Set the message tag.
ShardingKey	A partition field of sequential messages. Messages with the same ShardingKey will be sent to the same partition.

## Step 4. Consume messages

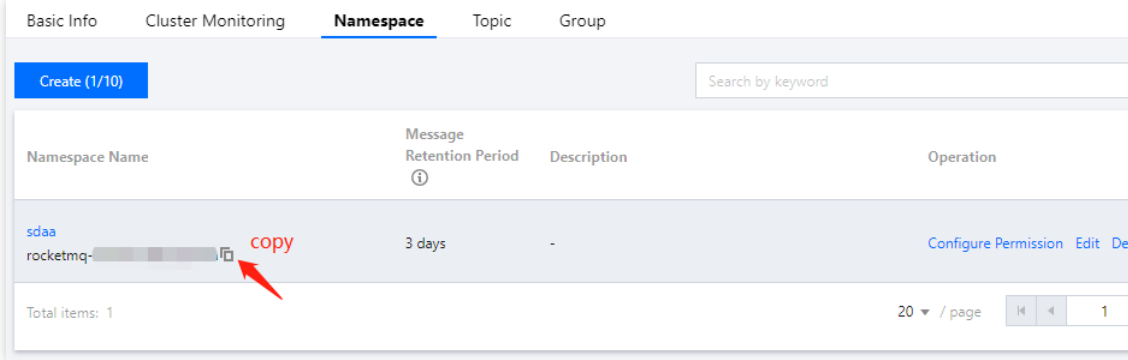
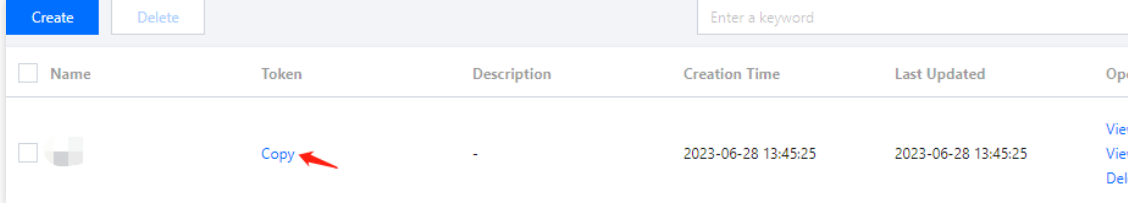
### Creating a consumer



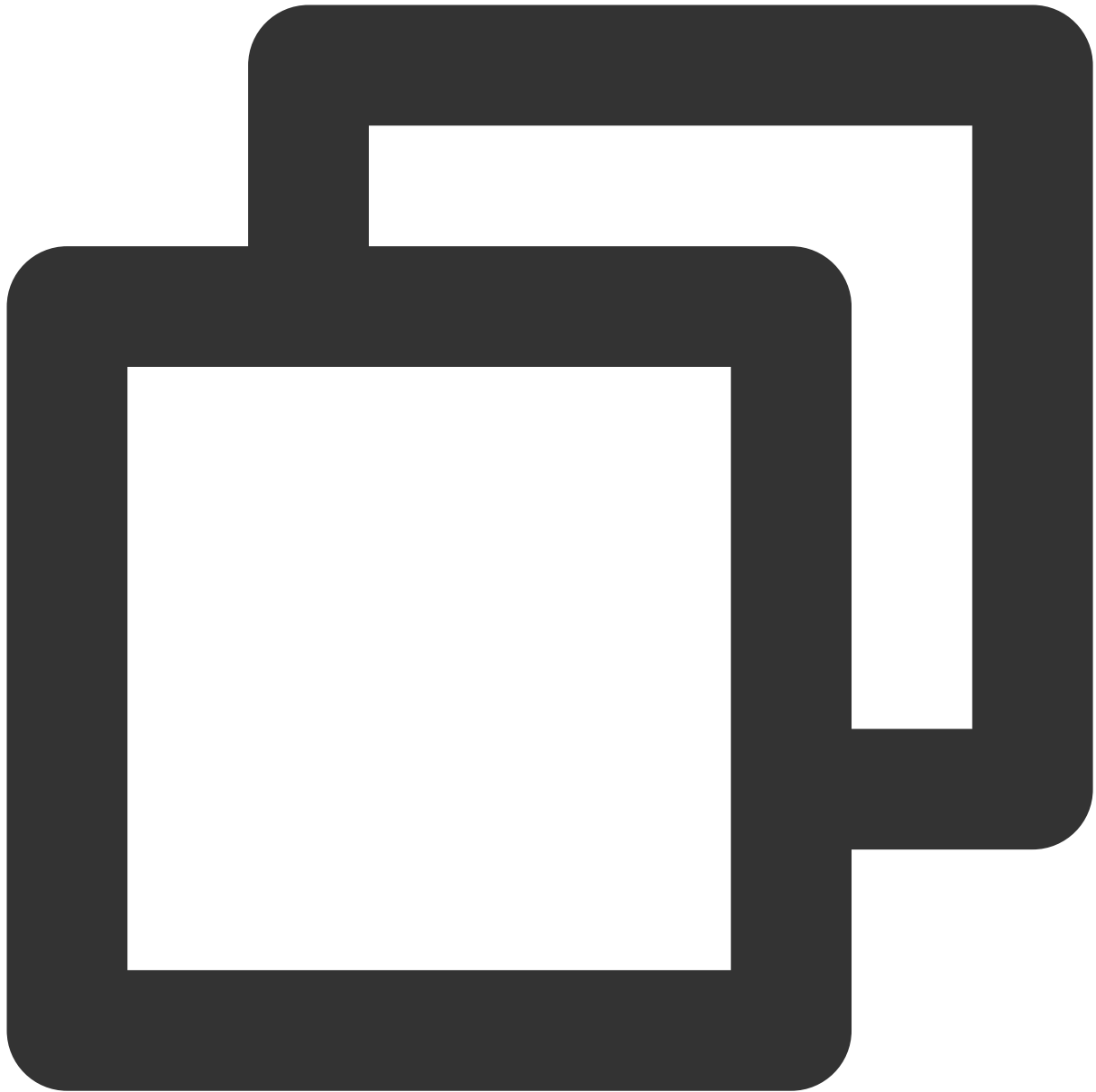
```
// Get the client
MQClient mqClient = new MQClient(endpoint, accessKey, secretKey);

// Get the topic consumer
MQProducer consumer = mqClient.getConsumer(namespace, topicName, groupName, "TAG
```

Parameter	Description
topicName	Topic name, which can be copied under the <b>Topic</b> tab on the <b>Cluster</b> page in the console.
groupName	Producer group name, which can be copied under the <b>Group</b> tab on the <b>Cluster</b> page in the console.

<p>namespace</p>	<p>Namespace name, which can be copied under the <b>Namespace</b> tab on the <b>Cluster</b> page in the console.</p> 
<p>TAG</p>	<p>Subscribed tag.</p>
<p>endpoint</p>	<p>Cluster access address over HTTP, which can be obtained from <b>Access Address</b> in the <b>Operation</b> <b>Cluster</b> page in the console.</p>
<p>secretKey</p>	<p>Role name, which can be copied on the <a href="#">Role Management</a> page.</p>
<p>accessKey</p>	<p>Role token, which can be copied in the <b>Token</b> column on the <a href="#">Role Management</a> page.</p> 

**Subscribing to messages**



```
do {
    List<Message> messages = null;

    try {
        // Long polling consumes messages sequentially. Although the messages obtai
        // For sequential consumption, as long as a message in a partition hasn't b
        // For a partition, the next batch of messages can only be consumed after a
        messages = consumer.consumeMessageOrderly(
            Integer.parseInt(batchSize),
            Integer.parseInt(waitSeconds)
        );
    }
}
```

```

    } catch (Throwable e) {
        e.printStackTrace();
    }
    if (messages == null || messages.isEmpty()) {
        System.out.println(Thread.currentThread().getName() + ": no new message, co
        continue;
    }

    for (Message message : messages) {
        System.out.println("Receive message: " + message);
    }

    {
        List<String> handles = new ArrayList<String>();
        for (Message message : messages) {
            handles.add(message.getReceiptHandle());
        }

        try {
            consumer.ackMessage(handles);
        } catch (Throwable e) {
            if (e instanceof AckMessageException) {
                AckMessageException errors = (AckMessageException) e;
                System.out.println("Ack message fail, requestId is:" + errors.getRe
                if (errors.getErrorMessages() != null) {
                    for (String errorHandle : errors.getErrorMessages().keySet()) {
                        System.out.println("Handle:" + errorHandle + ", ErrorCode:"
                            + ", ErrorMsg:" + errors.getErrorMessages().get(err
                    }
                }
                continue;
            }
            e.printStackTrace();
        }
    }
} while (true);

```

Parameter	Description
batchSize	The number of messages pulled at a time. Maximum value: 16.
waitSeconds	The polling waiting time for a message pull. Maximum value: 30 seconds.