TDMQ for RocketMQ

RocketMQ 5.x

Product Documentation

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 5.x

Introduction to RocketMQ 5.x

Overview

Last updated：2024-01-17 16:40:26

TDMQ for RocketMQ 5.x is a distributed message middleware developed by Tencent Cloud based on the latest Apache RocketMQ 5.0. It is fully compatible with the use and concepts of Apache RocketMQ 5.0 and supports connection to the open-source community version client without modifications.

In addition to retaining the characteristics of low latency, high performance, reliability, and scalability, and the capacity to process trillions of messages of its preceding versions, TDMQ for RocketMQ 5.x takes full advantage of the infrastructure and technologies in the cloud native era to enhance resource usage and elasticity.

Strengths

Last updated：2024-01-17 16:40:38

In addition to retaining the characteristics of low latency, high performance, reliability, and scalability, and the capacity to process trillions of messages of its preceding versions, TDMQ for RocketMQ 5.x takes full advantage of the infrastructure and ecological technologies in the cloud native era to enhance resource usage and elasticity.
The 5.x series possesses the following strengths compared to self-built RocketMQ and the TDMQ for RocketMQ 4.x series:

Elastic Storage and Computing

The 5.x series of TDMQ for RocketMQ uses an architecture featuring separation of storage and computation, significantly enhancing both resource usage and elasticity. The storage operates on a pay-as-you-go basis. Customers do not need to reserve storage resources in advance for their peak demands, thereby effectively reducing the actual cost. It also supports elastic TPS for computing specifications, eliminating the need for customers to reserve additional computing resources for unexpected peaks, achieving higher cost efficiency.

Lightweight SDK

TDMQ for RocketMQ 5.x series is compatible with the open-source community SDK, thereby enjoying the benefits of iterative development. The client for the 5.x series is more lightweight, with a new minimalist API design that is easier to integrate and use. Moreover, the 5.x series offers SDKs in a greater variety of programming languages, providing developers with a wider range of technical stack options.

Enhanced Basic Features

The 5.x series of TDMQ for RocketMQ introduces several functional enhancements, such as more flexible control over the duration for which messages are retained, enabling the retention period to be set at the granularities of the cluster or the individual topic. Consumer groups also have more customizable settings. For example, the number of retry attempts for messages can be specified on the server side, and dead letter queues can be freely associated.

##Observability

The 5.x series of TDMQ for RocketMQ introduces a richer set of metrics, including those related to message backlog scenarios, key interface time consumption, error distribution, and storage read-write traffic. These metrics are integrated seamlessly with Tencent Cloud's monitoring and alarm services. Moreover, it provides a comprehensive cloud API that supports integration with self-service Ops systems.

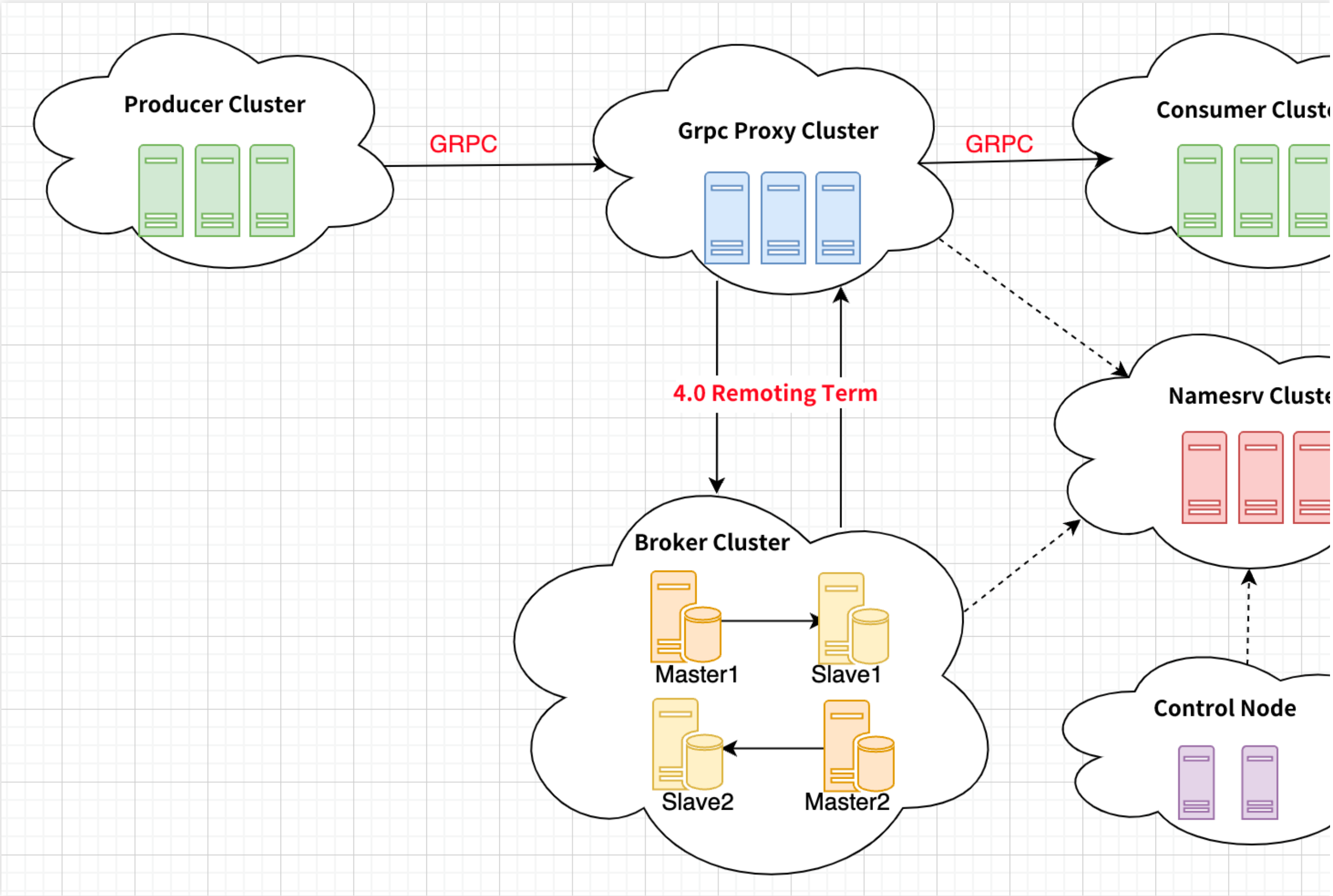
Architecture

Last updated：2024-01-17 16:40:54

This document describes the deployment architecture of TDMQ for RocketMQ 5.x for you to better understand the architectural principles of TDMQ for RocketMQ.

Deployment Architecture

The system deployment architecture of TDMQ for RocketMQ is as follows:



The TDMQ for RocketMQ 5.x introduces the new gRPC protocol and Proxy components, implementing an architecture featuring separation of computation and storage separation. This significantly changes both the Ops and usage of RocketMQ.

The concepts involved are as follows:

Producer cluster: A client-side application responsible for producing and sending messages.

Consumer cluster: A client-side application responsible for message subscription and consumption.

NameServer cluster: A server-side application responsible for routing address location and Broker heartbeat registration.

Heartbeat registration: The NameServer acts as a registration center. Machines of each role must report its status to the NameServer regularly. If a machine does not report within a certain time window, the NameServer will presume it to be faulty and remove it from the availability list.

Route addressing: Every NameServer stores both the complete routing information of the Broker cluster and the queue information for client queries. Producers and consumers use the NameServer to acquire route information of the entire Broker cluster, which then allows for message delivery and consumption.

Proxy cluster: The new elastic, stateless proxy service splits the Broker responsibilities for the 4.x version, abstracting elements such as client protocol adaptation, permissions management, and consumption management.

Broker cluster: Compared with the 4.x series, the Broker in the 5.x series is more focused on the continuous enhancements of storage capabilities.

Purchase Guide

Billing Overview

Last updated：2024-05-14 16:53:51

This section describes the billing composition and billing method of TDMQ for the RocketMQ 5.0 cluster.

Billing Method

TDMQ for RocketMQ offers two types of billing modes: **monthly subscription (prepaid)** and **pay-as-you-go (postpaid)** .

Monthly Subscription is a billing method that requires prepayment for resource use and is primarily suitable for steady business operation scenarios. You need to analyze resource use requirements according to the actual business volumes, paying upfront for one or multiple months or even years.

Pay-As-You-Go is a billing method that charges based on the actual usage of the resource specifications you purchased. It is mainly suitable for scenarios such as testing environments or those with uncertain peak traffic. You can first use the resources and then pay for them, with the costs being settled on the hour.

Billable Items

TDMQ for RocketMQ is sold in the form of clusters composed of the following billable items:

Billable Items	Billing Method	Billing Rules
Storage Fees	Pay-As-You-Go	The storage fees for the TDMQ for RocketMQ clusters are billed based on the size of the storage space occupied by the message storage and its retention duration.
Computing Specifications	Monthly subscription Pay-As-You-Go	TDMQ for RocketMQ clusters uses messaging TPS as their computational capabilities and provides different TPS specifications for each cluster version. Computational fees are billed based on the size of the TPS specification and its duration of use. The calculation rules for TPS are divided into two dimensions: message type and message size. Message Type: TDMQ for RocketMQ supports four types of messages: General messages, Scheduled and Delayed messages, Transactional messages, and Sequential messages. Both Scheduled and Delayed

		messages, Transactional messages, and Sequential messages are classified as advanced feature messages. General Message: Sending or consuming a single generic message. Whether the message is successfully sent or consumed, the message TPS is calculated as one time per second. Advanced Feature Message: Sending or consuming a single advanced feature message; the message TPS will be calculated five times per second. For example, if one Topic sends two transactional messages and consumes one transactional message, the resulting messaging TPS is calculated as 2x5+1x5=15 messages per second. Message Size: The message size is gauged in units of 4 KB. Messages that are less than 4KB will be considered as 4 KB. For instance, for a message request of 18 KB, the TPS for message sending would be ⌈18/4⌉ = 5 times per second. ⌈ ⌉ signifies rounding up to the nearest integer.
Elastic TPS Costs	Pay-As-You-Go	Each specification of TDMQ for the RocketMQ cluster is bound by TPS limitations. For the Pro and Platinum versions of the cluster, once a certain level of TSP specification has been purchased, the part exceeding the computational specification will be charged according to TPS count. **Elastic TPS is suitable for a scenario where occasional bursts of small amounts of traffic occur on the business side. If your business volume frequently exceeds computational specification limitations, it is recommended to upgrade your cluster specifications.**
Public Network Bandwidth Costs	Monthly subscription Pay-As-You-Go	After the public network bandwidth is activated, charges will be based on the duration of use. This pay-as-you-go mode is settled on an hourly basis. In cases of prepaid mode, settlement is carried out monthly. This is ideal for scenarios where peak business traffic is relatively stable at various time slots and only used for short-term purposes. No cost is incurred if the public network access is not activated.

Pricing Explanation

Storage Fees

TDMQ for RocketMQ cluster storage fees are calculated based on the size of the storage space occupied by message storage and the duration of storage. The billing method is pay-as-you-go (postpaid), with the billing unit as "XX dollars/GB/hour." An invoice is generated hourly. An invoice is generated hourly, and less than an hour is considered as one hour.

Storage fees = Storage space × Usage duration × Storage unit price

Storage Charge Type	Price (USD/GB/Hour)
Hourly Storage Billing (Postpaid)	0.0003

Computing Specification Costs

Billing for TDMQ for RocketMQ clusters is calculated based on the size of the specifications and the duration of use. The billing modes include both monthly subscription and pay-as-you-go options.

Monthly Subscription: The billing unit is "XX USD/month."

Pay-as-you-go (postpaid): The billing unit is "XX USD/hour." An invoice is generated hourly. An invoice is generated hourly, and less than an hour is considered as one hour. Settlement is done daily, and the bill will be pushed for deductions on the following day.

Computing Specification Costs = Unit price of the computing specification x Usage duration.

Version	TPS Specification	Topic Upper Limit (Quantity)	Group Upper Limit (Quantity)	Region: Beijing, Guangzhou, Shanghai, Nanjing, Chengdu, Chongqing, and Qingyuan.		Region: Hong Kor Taipei (China), Tc Singapore, Bangk Silicon Valley.	
				Pay-As-You-Go Price (USD/Hour)	Monthly Subscription Price (USD/Month)	Pay-As-You-Go Price (USD/Hour)	M S P (U
Trial Version	500	50	500	0.11	54.86	0.15	7
Basic	1,000	100	1,000	0.24	114.29	0.31	1
	2,000	100	1,000	0.36	171.43	0.46	2
	3,000	100	1,000	0.48	228.57	0.62	2
	4,000	100	1,000	0.60	285.71	0.77	3
	5,000	100	1,000	0.71	342.86	0.93	4
	6,000	200	2,000	0.83	400.00	1.08	5
	7,000	200	2,000	0.95	457.14	1.24	5
	8,000	200	2,000	1.07	514.29	1.39	6

	9,000	200	2,000	1.19	571.43	1.55	7
	10,000	200	2,000	1.31	628.57	1.70	8
Professional Version	4,000	300	3,000	1.14	548.57	1.49	7
	6,000	300	3,000	1.63	780.00	2.11	1
	8,000	300	3,000	2.11	1,011.43	2.74	1
	10,000	300	3,000	2.59	1,242.86	3.37	1
	15,000	325	3,250	3.30	1,585.71	4.30	2
	20,000	350	3,500	3.90	1,871.43	5.07	2
	25,000	375	3,750	4.49	2,157.14	5.84	2
	30,000	400	4,000	5.09	2,442.86	6.62	3
	35,000	425	4,250	5.68	2,728.57	7.39	3
	40,000	450	4,500	6.28	3,014.29	8.16	3
	45,000	475	4,750	6.88	3,300.00	8.94	4
	50,000	500	5,000	7.47	3,585.71	9.71	4
	55,000	550	5,500	7.66	3,675.00	9.95	4
	60,000	600	6,000	8.21	3,942.86	10.68	5
	65,000	650	6,500	8.77	4,210.71	11.40	5
	70,000	700	7,000	9.33	4,478.57	12.13	5
	75,000	750	7,500	9.89	4,746.43	12.86	6
	80,000	800	8,000	10.45	5,014.29	13.58	6
	85,000	850	8,500	11.00	5,282.14	14.31	6
	90,000	900	9,000	11.56	5,550.00	15.03	7
	95,000	950	9,500	12.12	5,817.86	15.76	7
	100,000	1,000	10,000	12.68	6,085.71	16.48	7
Platinum Version	10,000	1,000	10,000	4.61	2,214.29	6.00	2

(Exclusive Resources)	20,000	1,000	10,000	5.95	2,857.14	7.74	3
	30,000	1,000	10,000	7.29	3,500.00	9.48	4
	40,000	1,000	10,000	8.63	4,142.86	11.22	5
	50,000	1,000	10,000	9.97	4,785.71	12.96	6
	60,000	1,100	11,000	10.42	5,000.00	13.54	6
	70,000	1,200	12,000	11.61	5,571.43	15.09	7
	80,000	1,300	13,000	12.80	6,142.86	16.64	7
	90,000	1,400	14,000	13.99	6,714.29	18.19	8
	100,000	1,500	15,000	15.18	7,285.71	19.73	9
	120,000	1,600	16,000	17.56	8,428.57	22.83	1
	140,000	1,700	17,000	19.94	9,571.43	25.92	1
	160,000	1,800	15,000	21.73	10,428.57	28.24	1
	180,000	1,900	19,000	23.81	11,428.57	30.95	1
	200,000	2,000	20,000	25.89	12,428.57	33.66	1
	250,000	2,500	25,000	31.10	14,928.57	40.43	1
	300,000	3,000	30,000	36.31	17,428.57	47.20	2
	350,000	3,500	35,000	41.52	19,928.57	53.97	2
	400,000	4,000	40,000	46.73	22,428.57	60.74	2
	450,000	4,500	45,000	51.93	24,928.57	67.51	3
	500,000	5,000	50,000	57.14	27,428.57	74.29	3
	600,000	6,000	60,000	63.99	30,714.29	83.19	3
	700,000	7,000	70,000	73.66	35,357.14	95.76	4
	800,000	8,000	80,000	83.33	40,000.00	108.33	5
	900,000	9,000	90,000	93.01	44,642.86	120.91	5
	1,000,000	10,000	100,000	102.68	49,285.71	133.48	6

Elastic TPS Costs

Only the Professional version and Platinum version clusters support the "Elastic TPS" feature and are subject to these costs.

The TROCKET cluster's Elastic TPS are calculated by the pay-as-you-go (postpaid) billing method. The billing unit is "XX dollars/TPS/hour". An invoice is generated hourly, and less than an hour is considered as one hour.

The cost of elastic TPS for each hour = the highest increment TPS value in 1 hour x Unit price of the elastic TPS.

Version	TPS Specification (Transactions/Second)	Elastic TPS Upper Limit (Transactions/Second)	Price (USD/TPS/Hour, Region: Beijing, Guangzhou, Shanghai, Nanjing, Chengdu, Chongqing, and Qingyuan)	Price (USD/TPS/Hour, Region: Hong Kong (China), Taipei (China), Tokyo, Singapore, Bangkok, and Silicon Valley)
Trial Version	Does Not Support Elastic TPS Capability	Not Involved		
Basic	Does Not Support Elastic TPS Capability			
Professional Version	4,000	2,500	0.0008	0.00104
	6,000	4,000		
	8,000	5,000		
	10,000	6,000		
	15,000	9,000		
	20,000	12,000		
	25,000	13,000		
	30,000	15,000		
	35,000	18,000		
	40,000	20,000		
	45,000	22,000		
	50,000	25,000		

	55,000				
	60,000				
	65,000				
	70,000				
	75,000	30,000			
	80,000				
	85,000				
	90,000				
	95,000	35,000			
	100,000	40,000			
Platinum Version	10,000	6,000	0.0017	0.00221	(
	20,000	12,000			
	30,000	18,000			
	40,000	25,000			
	50,000	30,000			
	60,000				
	70,000				
	80,000	40,000			
	90,000				
	100,000				
	120,000				
	140,000	50,000			
	160,000	60,000			
	180,000				
	200,000	80,000			

	250,000			
	300,000	100,000		
	350,000			
	400,000	150,000		
	450,000			
	500,000	200,000		
	600,000	250,000		
	700,000			
	800,000	300,000		
	900,000			
	1,000,000	350,000		

Billing Example:

Suppose your cluster is located in Guangzhou, the cluster type is professional version, and the purchased basic messaging TPS specification is 10,000 times/second, the elastic TPS upper limit is 6,000 times/second, and the unit price of elastic TPS is $0.0008/TPS/second. Suppose the message sending and receiving situation of this cluster is as follows between 14:00–15:00:

At 14:00, the peak TPS usage of the cluster is 9,000 times per second with no use of elastic TPS.

At 14:01, the peak TPS usage of the cluster is 9,500 times per second with no use of elastic TPS.

At 14:02, the peak TPS usage of the cluster is 10,500 times per second, thus 500 times per second of the elastic TPS is used.

……, the peak TPS usage of the cluster consistently remained between 8,000–9,000 times per second with no use of elastic TPS.

At 14:58, the peak TPS usage of the cluster is 12,000 times per second, thus 2,000 times per second of the elastic TPS is used.

At 14:59, the peak TPS usage of the cluster is 11,000 times per second, thus 1,000 times per second of the elastic TPS is used.

Consequently, within this hour, we take the maximum increments of TPS in one hour, which is 2,000 times per second. Therefore, the cost for the elastic TPS generated within this hour is 2,000 x 0.0008 = 1.6 USD.

Public Network Bandwidth Costs

Public Network Bandwidth Costs support two billing modes: monthly subscription and hourly billing.

Monthly Subscription

Bandwidth Range	Price (USD/Month)				
	Beijing, Guangzhou, Shenzhen Finance, Shanghai, Nanjing, Chengdu, Chongqing, and Qingyuan	Hong Kong (China), Taipei (China), and Silicon Valley	Singapore	Bangkok and Tokyo	Shanghai Autonomous Driving Zone
1 Mbps	3.29	4.29	3.29	3.57	4.93
2 Mbps	6.57	8.57	6.57	7.14	9.86
3 Mbps	10.14	12.86	9.86	10.71	15.21
4 Mbps	13.71	17.14	13.14	14.29	20.57
5 Mbps	17.86	21.43	16.43	17.86	26.79
6 Mbps and above (n is the configured upper limit of bandwidth)	17.86 + 11.43 x (n-5)	21.43 + 14.29 x (n-5)	16.43 + 11.43 x (n-5)	17.86 + 11.43 x (n-5)	43.93 + 17.14 x (n-5)

Pay-as-You-Go Billing Mode

Billing is based on the total amount of data transmitted over the public network (measured in GB), with settlements completed on an hourly basis. Charges are settled according to actual traffic usage.

Bandwidth Range	Price (USD/Hour)		
	Beijing, Guangzhou, Shanghai, Nanjing, Chengdu, Chongqing, Qingyuan, Shenzhen Finance, Hong Kong (China), Taipei	Singapore and Silicon Valley	Region: Shanghai Autonomous Driving Zone

	(China), Tokyo, and Bangkok		
1–5 Mbps	0.0057	0.0050	0.0086
6 Mbps and above (n is the configured upper limit of bandwidth)	0.0200	0.0171	0.0300

Charges for Topics Beyond Specification Limits

Considering the stability of the cluster and real-world use cases, the maximum number of Topics allowed varies with different TPS specifications. Customers can increase the Topic quantity limit on their own via the page. Charges apply according to a tiered pricing structure for any amount that exceeds the free quota.

Monthly Subscriptions

Excess Topic Quantity Pricing Tiers	Price (region: Beijing, Guangzhou, Shanghai, Nanjing, Chengdu, and Chongqing)	Price (Region: Hong Kong (China), Taipei (China), Tokyo, Singapore, Bangkok, and Silicon Valley)	Price (region: Shenzhen Finance and Shanghai Autonomous Driving Zone)
0–100	1.6598 USD/Unit/Month	2.1577 USD/Unit/Month	2.6556 USD/Unit/Month
101–200	1.3831 USD/Unit/Month	1.7981 USD/Unit/Month	2.213 USD/Unit/Month
201–500	1.1065 USD/Unit/Month	1.4385 USD/Unit/Month	1.770 USD/Unit/Month
501–1,500	0.8299 USD/Unit/Month	1.0788 USD/Unit/Month	1.3278 USD/Unit/Month
1,501–2,000	0.5533 USD/Unit/Month	0.7192 USD/Unit/Month	0.8852 USD/Unit/Month
Above 2,000	0.274 USD/Unit/Month	0.3596 USD/Unit/Month	0.4426 USD/Unit/Month

Purchasing pay-as-you-go instances

Excess Topic Quantity Pricing Tiers	Price (region: Beijing, Guangzhou, Shanghai,	Price (Region: Hong Kong (China),	Price (region: Shenzhen Finance and Shanghai

	Nanjing, Chengdu and Chongqing)	Singapore, Bangkok, and Silicon Valley)	Autonomous Driving Zone)
0–100	0.0035 USD/Unit/Hour	0.0045 USD/Unit/Hour	0.0055 USD/Unit/Hour
101–200	0.0028 USD/Hour	0.0036 USD/Hour	0.0044 USD/Hour
201–500	0.0022 USD/Hour	0.0028 USD/Hour	0.0035 USD/Unit/Hour
501–1,500	0.0017 USD/Hour	0.0022 USD/Hour	0.0028 USD/Hour
1,501–2,000	0.0011 USD/Hour	0.0014 USD/Hour	0.0018 USD/Hour
Above 2,000	0.0006 USD/Hour	0.0007 USD/Hour	0.0009 USD/Hour

Product Series

Last updated：2024-05-14 16:56:53

TDMQ for RocketMQ offers various edition specifications to cater to the diverse needs of different business scenarios and scales. The differences in functions between these editions are as follows.

Item	Trial Edition	Basic Edition	Pro Edition	Platinum Edition
TPS specifications (times/second)	500	1000-10000	4000-100000	10000-1000000
Message trace	Supported			
Read/write traffic ratio	Adjustable			
Message retention duration configuration	1 to 3 days		1 to 7 days	
Message retention granularity configuration	Cluster granularity		Topic granularity	
SQL filtering	Supported		Supported	
Message retry policy	Default		Customizable	
Elastic TPS	Not supported		Supported	
Duration of scheduled messages	7 days	40 days	40 days	Customizable, up to a maximum of 1 year
Message size	4MB		Customizable, 20 MB	
Service availability	SLA not guaranteed	99.95%	99.95%	99.99%
Deployment Architecture	Resource sharing	Resource sharing	Resource sharing	Exclusive resource allocation

Deployment architecture	24/7 work order service	Professional service support, training and communication, key event protection services, and architecture consultation and suggestions

Purchase Methods

Last updated：2024-01-17 16:41:12

TDMQ for RocketMQ clusters currently support the **monthly subscription** and **pay-as-you-go** billing modes. You can follow the operations below to purchase services:

1. Log in to the Tencent Cloud RocketMQ Console.
2. Select **Cluster** from the left sidebar and click **Create Cluster** to open the purchase page.
3. On the purchase page, select the region, availability zone, cluster type, and cluster specification.
4. Specify the information, click **Buy Now**, and make the payment as prompted by the system to complete the purchase.

Refund

Last updated：2024-01-17 16:41:26

Pay-as-you-go

Clusters under pay-as-you-go mode can be terminated at any moment. Once terminated, the billing will cease immediately.

Monthly Subscription

Refund Description

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

Products purchased in a promotional event must adhere to the campaign policy. In case of a conflict between the campaign policy and refund rules, the campaign policy prevails. If the campaign policy expressly prohibits refunds, refund application cannot be initiated.

Currently, orders made via promotional channels does not allow self-initiation of refunds. Please submit a ticket to apply for a refund.

Tencent Cloud reserves the right to reject the return request if we detect suspected anomalies or malicious returns.

Refund Quantity and Channels

Refund amount = Paid order amount - Consumed resource amount

The calculation is based on the **usage duration**:

Consumed amount = (Usage duration/Total duration) × Original order cost × Current discount

Note

If the usage duration is less than one day, it will be calculated as one day, with the system's discount according to the usage duration applied.

Arrears

Last updated：2024-01-17 16:41:32

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.

Notice

When the cluster is no longer in use, terminate it to avoid ongoing charges.

Upon the termination or repossession of a cluster, the associated data will be cleared and cannot be retrieved.

Pay-as-you-go

Under the pay-as-you-go mode, the billing cycle is "daily". That is, the billing system measures service usage and generates a bill based on the previous day's usage on the next day, and deducts the service fees from your account according to the bill amount.

If the current account balance is insufficient but the current usage falls within the free tier, the service can still be used.

If your account balance is insufficient and you do not have the privilege of uninterrupted service during overdue payment, TDMQ for RocketMQ can continue to be used and be charged for 24 hours. Then, TDMQ for RocketMQ services will be suspended, disrupting the normal sending and receiving of messages and the routine operation of the console and APIs, and resource occupancy fees will be incurred.

Upon service suspension, the system will process TDMQ for RocketMQ in the following ways:

Time after Service Suspension	Description
≤ 7 days	If your account is recharged to a positive balance within this period, the billing continues and you can reactivate TDMQ for RocketMQ.
	If your account does not have a positive balance, you cannot reactivate TDMQ for RocketMQ.
> 7 days	If your account does not have a positive balance, pay-as-you-go TDMQ for RocketMQ resources will be terminated, and all data will be erased and cannot be retrieved. When your resources are terminated, the Tencent Cloud account creator and all collaborators will be notified by email and SMS.

Monthly Subscription

TDMQ for RocketMQ exclusively adopts the monthly subscription billing mode.

Alert for Overdue Payment

Seven days before the expiration of your monthly-subscribed instance, the system will automatically send you an expiration warning message every other day. This message will be sent via **email and SMS** to the creator and all collaborators of the Tencent Cloud account.

Overdue Payment Notification

Starting from the day of the expiration of your monthly-subscribed instance, an overdue payment isolation alert will be sent to you every other day. These alert messages will be sent via **email and SMS** to the creator and all collaborators of the Tencent Cloud account.

Overdue Payment Policy

If the account balance is sufficient and auto-renewal has been configured, the device will undertake automatic renewal on its expiration date.

For seven days after expiration, your cluster instance can function normally. If a renewal is made within this period, **the start date for the renewed cluster instance's period will be the termination date of the previous cycle**.

In the event that your cluster instance is not renewed within seven days after expiration, the cluster service will be suspended. Then, all related cluster resources will be terminated, and all data will be cleared and cannot be retrieved.

Upon cluster termination, the creator and all collaborators of the Tencent Cloud account will be notified via email and SMS.

Note

Upon receipt of an overdue payment notice, promptly proceed to the Billing Center in the console to top up your account, thereby avoiding any potential disruption to your operations.

If you have any questions related to your billing details, you may cross-verify these details by visiting the Resource Bills page in the console.

If you have any questions about specific charges, see Pricing Details for in-depth description of each billing item and the corresponding billing rules.

You can self-configure alerts for overdue payments through the balance alert feature available in the Billing Center. For more information, see Balance Alerts Guide.

Getting Started

Resource Creation and Preparation

Last updated：2024-01-17 17:44:26

Operation Scenarios

Before using the SDK to send and receive messages, you need to create resources such as clusters and Topic in the console of TDMQ for RocketMQ . Relevant resource information needs to be configured when running the client.

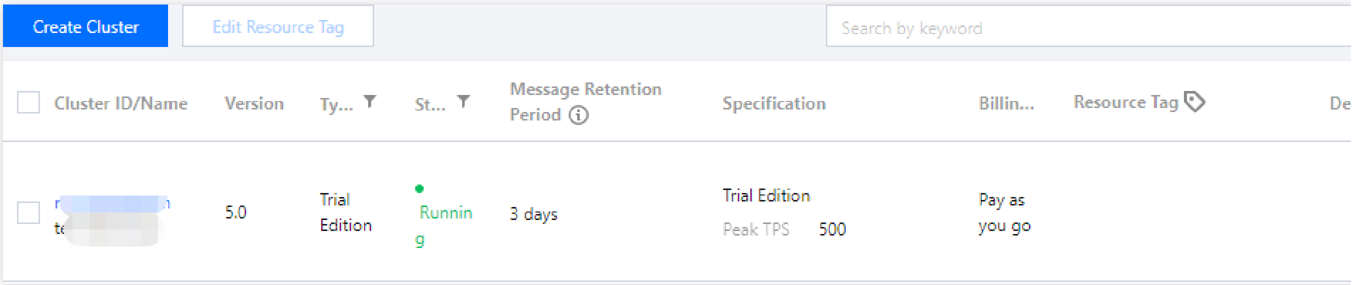
Prerequisites

You have completed Sign up for a Tencent Cloud account.

Steps

Step 1. Create a new cluster

1. Log in to the RocketMQ console, enter the **Cluster Management** page, and select the target region.
2. Click **Create Cluster**, select the appropriate cluster specifications, and then click **Buy Now** to establish a new cluster.



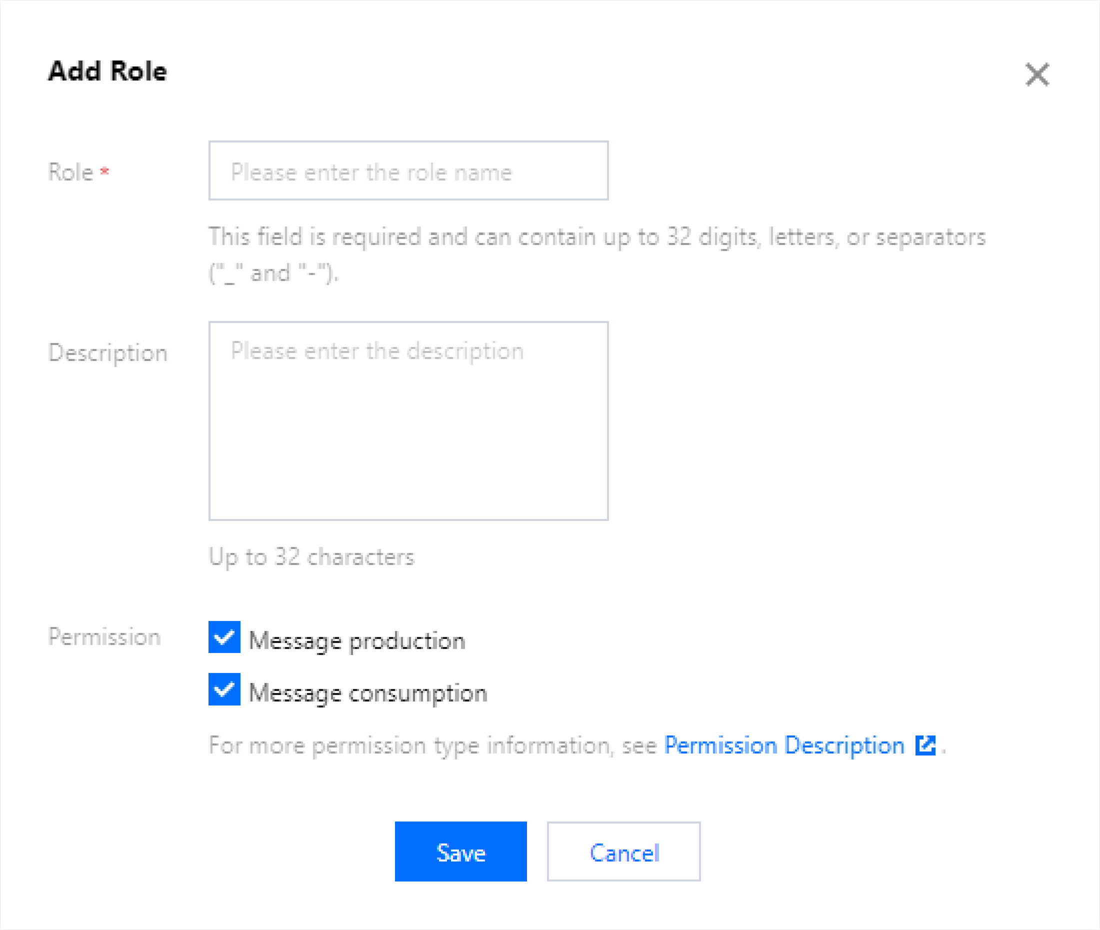
□	Cluster ID/Name	Version	Ty...	St...	Message Retention Period ⓘ	Specification	Billin...	Resource Tag	De
□	[redacted]	5.0	Trial Edition	● Running	3 days	Trial Edition Peak TPS 500	Pay as you go		

3. From the cluster list page, click the created cluster ID. In **Access Information** section of the cluster information page, you can view the access point details of the cluster.

Step 2. Configure cluster permissions

1. Click the "ID" of the created cluster in Step 1 to access the basic information page of the cluster.

2. Select the Cluster Permissions tab at the top of the page, click **Add Role** to create a role, and configure production and consumption permissions for it.



Add Role

Role * Please enter the role name

This field is required and can contain up to 32 digits, letters, or separators ("_" and "-").

Description Please enter the description

Up to 32 characters

Permission ☑ Message production

☑ Message consumption

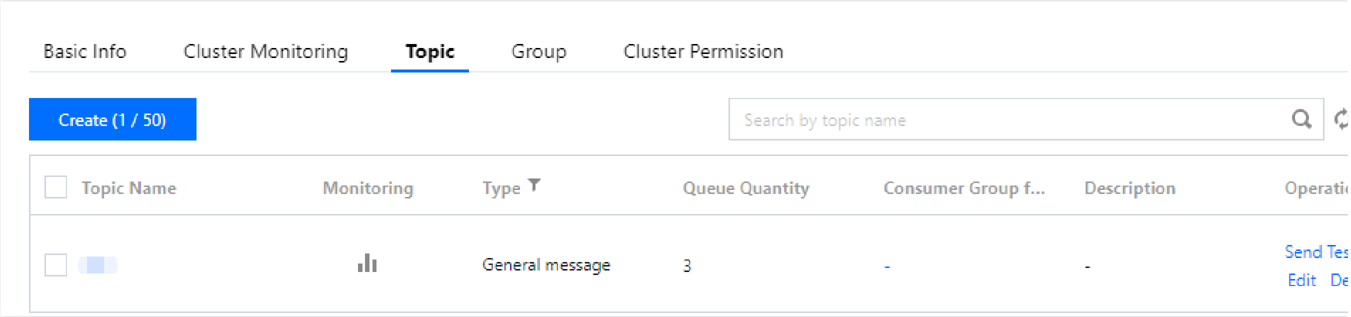
For more permission type information, see Permission Description.

Step 3. Create a Topic

1. From the cluster permissions list page, select the Topic tab to access the Topic list page.
2. Click **Create**, enter a Topic name, select **General message** as the type, then click **Submit** to create a Topic.

Note:

This document describes the process of using general messages as an example, therefore, the created Topic following these steps for general messages cannot be used to send or receive messages of other types.



Basic Info | Cluster Monitoring | **Topic** | Group | Cluster Permission

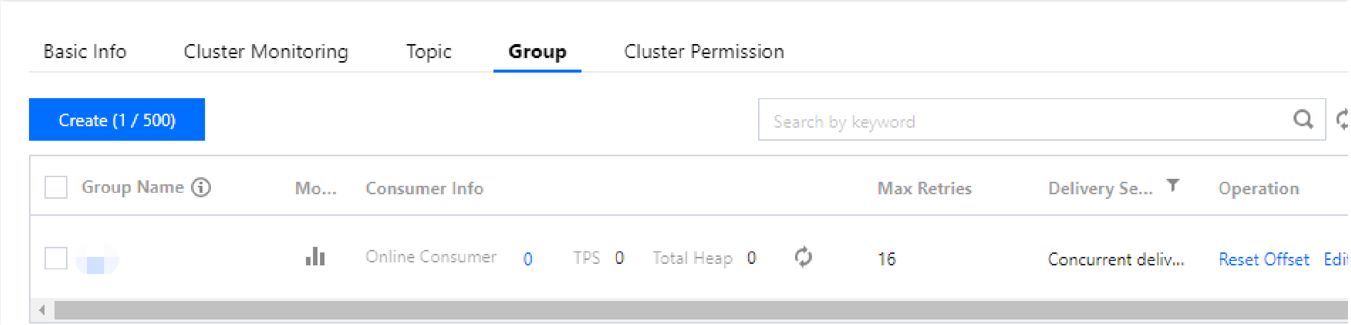
Create (1 / 50)

Search by topic name

□	Topic Name	Monitoring	Type	Queue Quantity	Consumer Group f...	Description	Operati
□	[redacted]		General message	3	-	-	Send Tes Edit De

Step 4. Create a Group

1. From the Topic list page, select the **Group** tab at the top to access the Group list page.
2. Click **Create**, enter a Group name, leave the other settings at their defaults, then click **Submit** to create a Group.



Basic Info | Cluster Monitoring | Topic | **Group** | Cluster Permission

Create (1 / 500)

Search by keyword

□	Group Name	Mo...	Consumer Info	Max Retries	Delivery Se...	Operation
□	[redacted]		Online Consumer 0 TPS 0 Total Heap 0	16	Concurrent deliv...	Reset Offset Edi

Use SDK for TDMQ 5.x to Send/Receive General Messages

Last updated：2024-01-17 17:44:44

Operation Scenarios

In calling Java SDK for example, this article introduces the process of using an open-source SDK to send and receive messages, thus enhancing your comprehension of the complete procedure in messaging.

Note:

Take the Java client as an exemple. For clients employing other languages, please refer to the SDK Documentation.

Prerequisites

Resources Creation and Preparationalready completed

Install JDK 1.8 or the later

Install Maven 2.5 or later

Download the demo

Steps

Step 1: Install the Java dependency

Incorporate the relevant dependencies in the Java project. Take a Maven project as an example, add the following dependency to pom.xml:

```
<dependencies>
        <dependency>
            <groupId>org.apache.rocketmq</groupId>
            <artifactId>rocketmq-client-java</artifactId>
            <version>5.0.5</version>
        </dependency>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
            <version>1.7.7</version>
        </dependency>
```

```
</dependencies>
```

Step 2: Generate a Message

```
public class NormalMessageSyncProducer {
    private static final Logger log = LoggerFactory.getLogger(NormalMessageSyncProd

    private NormalMessageSyncProducer() {
    }
    public static void main(String[] args) throws ClientException, IOException {
        final ClientServiceProvider provider = ClientServiceProvider.loadService();

        // Add the configuration's ak and sk.
        String accessKey = "yourAccessKey"; //ak
        String secretKey = "yourSecretKey"; //sk
        SessionCredentialsProvider sessionCredentialsProvider =
            new StaticSessionCredentialsProvider(accessKey, secretKey);

        // Enter the service access address provided by Tencent Cloud.
        String endpoints = "rmq-xxx.rocketmq.xxxtencenttdmq.com:8081";
        ClientConfiguration clientConfiguration = ClientConfiguration.newBuilder()
            .setEndpoints(endpoints)
            .enableSsl(false)
            .setCredentialProvider(sessionCredentialsProvider)
            .build();
        String topic = "yourNormalTopic";
        // In most case, you don't need to create too many producers, singleton pat
        final Producer producer = provider.newProducerBuilder()
            .setClientConfiguration(clientConfiguration)
            // Set the topic name(s), which is optional but recommended. It makes p
            // route before message publishing.
            .setTopics(topic)
            // May throw {@link ClientException} if the producer is not initialized
            .build();
        // Define your message body.
        byte[] body = "This is a normal message for Apache RocketMQ".getBytes(Stand
        String tag = "yourMessageTagA";
        final Message message = provider.newMessageBuilder()
            // Set topic for the current message.
            .setTopic(topic)
            // Message secondary classifier of message besides topic.
            .setTag(tag)
            // Key(s) of the message, another way to mark message besides message i
            .setKeys("yourMessageKey-1c151062f96e")
            .setBody(body)
            .build();
        try {
            final SendReceipt sendReceipt = producer.send(message);
            log.info("Send message successfully, messageId={}", sendReceipt.getMess
        } catch (Throwable t) {
            log.error("Failed to send message", t);
        }
        // Close the producer when you don't need it anymore.
        producer.close();
    }
}
```

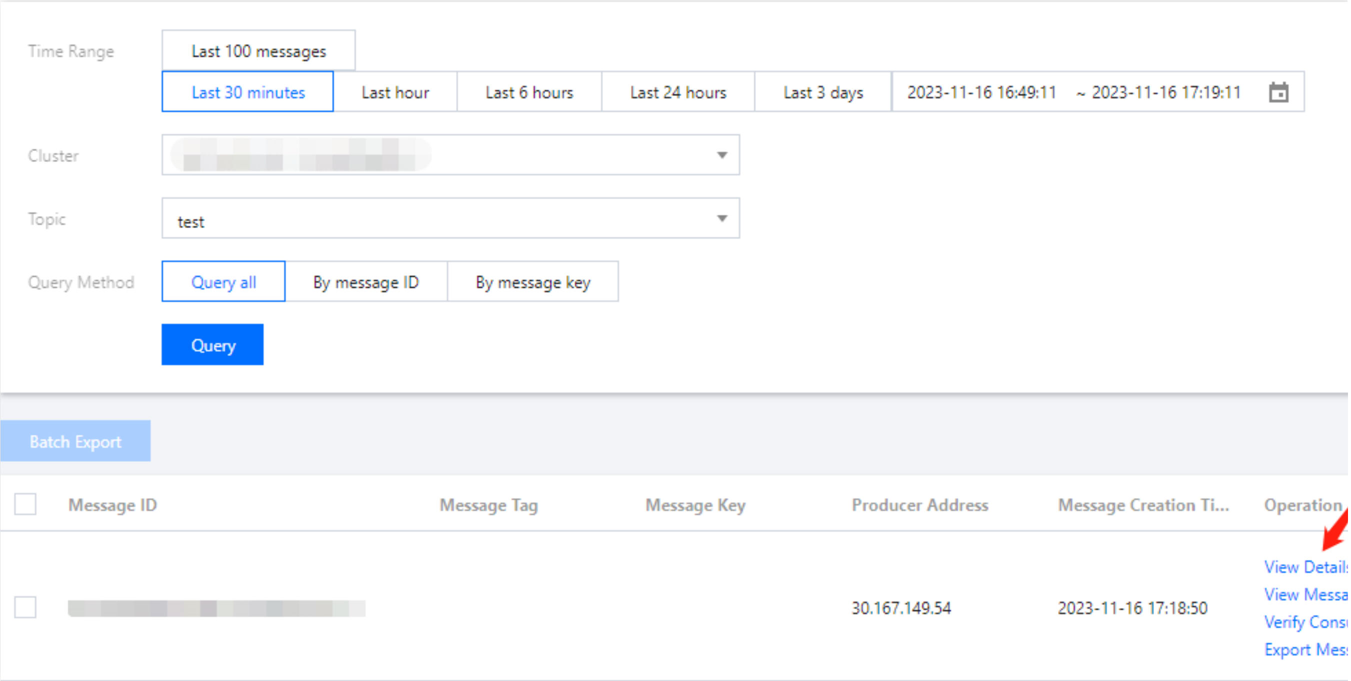
Step 3: Consume the message

Tencent Cloud's TDMQ for RocketMQ Version 5.x series supports two consumption modes: Push Consumer and Simple Consumer. The following code example illustrates the use of Push Consumer:

```
public class NormalPushConsumer {
    private static final Logger log = LoggerFactory.getLogger(NormalPushConsumer.cl

    private NormalPushConsumer() {
    }
    public static void main(String[] args) throws ClientException, IOException, Int
        final ClientServiceProvider provider = ClientServiceProvider.loadService();

        // Add the configuration's ak and sk.
        String accessKey = "yourAccessKey"; //ak
        String secretKey = "yourSecretKey"; //sk
        SessionCredentialsProvider sessionCredentialsProvider =
            new StaticSessionCredentialsProvider(accessKey, secretKey);

        // Enter the service access address provided by Tencent Cloud.
        String endpoints = "rmq-xxx.rocketmq.xxxtencenttdmq.com:8081";
        ClientConfiguration clientConfiguration = ClientConfiguration.newBuilder()
            .setEndpoints(endpoints)
            .enableSsl(false)
            .setCredentialProvider(sessionCredentialsProvider)
            .build();
        String tag = "*";
        FilterExpression filterExpression = new FilterExpression(tag, FilterExpress
        String consumerGroup = "yourConsumerGroup";
        String topic = "yourTopic";
        // In most case, you don't need to create too many consumers, singleton pat
        PushConsumer pushConsumer = provider.newPushConsumerBuilder()
            .setClientConfiguration(clientConfiguration)
            // Set the consumer group name.
            .setConsumerGroup(consumerGroup)
            // Set the subscription for the consumer.
            .setSubscriptionExpressions(Collections.singletonMap(topic, filterExpre
            .setMessageListener(messageView -> {
                // Handle the received message and return consume result.
                log.info("Consume message={}", messageView);
                return ConsumeResult.SUCCESS;
            })
            .build();
        // Block the main thread, no need for production environment.
        Thread.sleep(Long.MAX_VALUE);
        // Close the push consumer when you don't need it anymore.
        pushConsumer.close();
    }
}
```

Step 4: View message details

After sending the message, a message ID (messageID) is generated. Developers can then verify this recently sent message on the "Message Query" page as shown below. Additionally, it allows viewing specific details and track information of the message. For more information, please refer to the Querying Message section.



Time Range: Last 100 messages | Last 30 minutes | Last hour | Last 6 hours | Last 24 hours | Last 3 days | 2023-11-16 16:49:11 ~ 2023-11-16 17:19:11

Cluster: [redacted]

Topic: test

Query Method: Query all | By message ID | By message key

Query

Batch Export

□	Message ID	Message Tag	Message Key	Producer Address	Message Creation Ti...	Operation
□	[redacted]			30.167.149.54	2023-11-16 17:18:50	View Detail View Messa Verify Cons Export Mes

Use SDK for TDMQ 4.x to Send/Receive General Messages

Last updated：2024-01-17 17:45:31

Operation Scenarios

In calling 4.0 Java SDK for example, this article introduces the process of using an open-source SDK to send and receive messages, thus enhancing your comprehension of the complete procedure in messaging.

Note:

Take the Java client as an exemple. For clients employing other languages, please refer to the SDK Documentation.

Prerequisites

Resources Creation and Preparation already completed

Install JDK 1.8 or the later

Install Maven 2.5 or the later

Download the demo

Steps

Step 1: Install the Java dependency

Incorporate the relevant dependencies in the Java project. Take a Maven project as an example, add the following dependency to pom.xml:

```
<!-- in your <dependencies> block -->
  <dependency>
      <groupId>org.apache.rocketmq</groupId>
      <artifactId>rocketmq-client</artifactId>
      <version>4.9.7</version>
  </dependency>

  <dependency>
      <groupId>org.apache.rocketmq</groupId>
      <artifactId>rocketmq-acl</artifactId>
      <version>4.9.7</version>
```

```
    </dependency>
```

Step 2: Generate a Message

```
// Instantiate the message producer.
   DefaultMQProducer producer = new DefaultMQProducer(
       groupName,
       new AclClientRPCHook(new SessionCredentials(accessKey, secretKey)) // ACL Pe
   );
   // Set the NameServer address. The address should be in the form of xxx.tencentt
```

```
    producer.setNamesrvAddr(nameserver);
    // Start the Producer instance.
    producer.start();

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

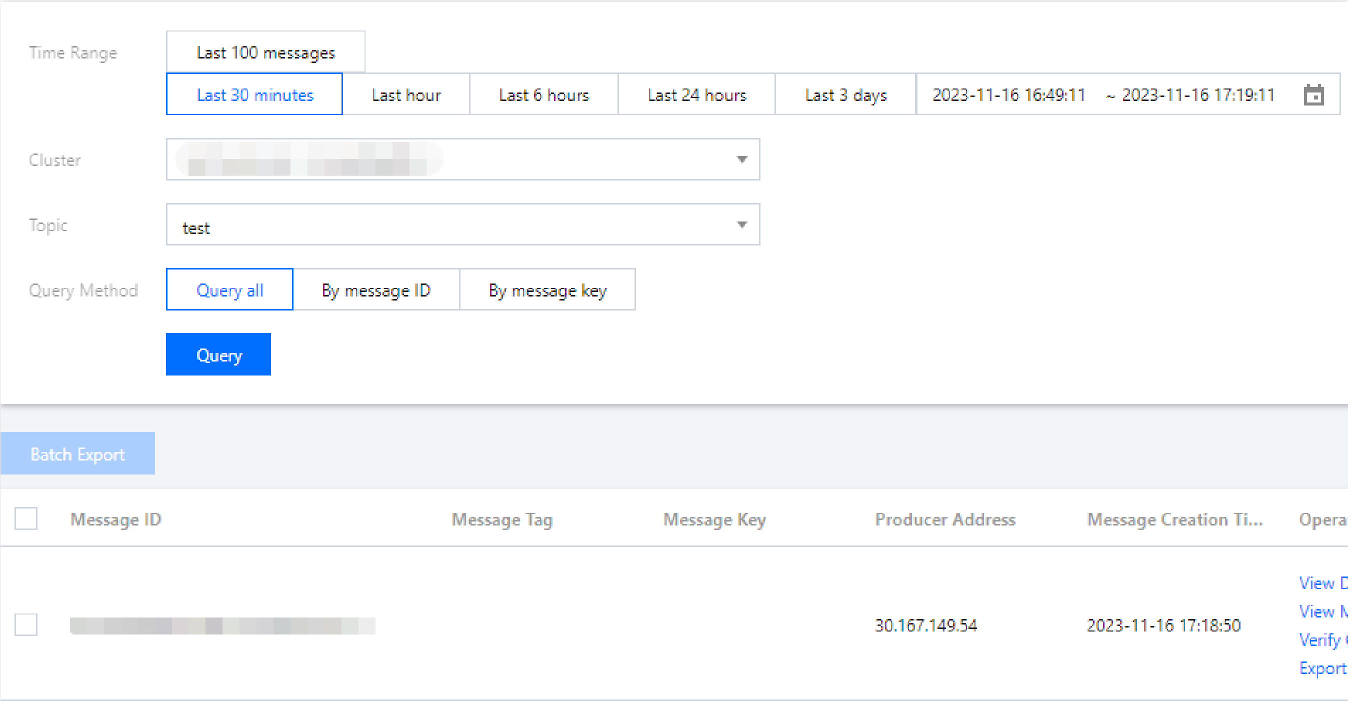
Step 3: Consume the message

The following code example illustrates the use of Push Consumer. For further details, please refer to the more detailed 4.x documentation.

```
// Instantiate the consumer
DefaultMQPushConsumer pushConsumer = new DefaultMQPushConsumer(
            groupName,
            new AclClientRPCHook(new SessionCredentials(accessKey, secretKey))); //
        // Set the NameServer address.
pushConsumer.setNamesrvAddr(nameserver);
        // Subscribe to a topic.
pushConsumer.subscribe(topic_name, "*");
      // Register a callback implementation class to process messages pulled from t
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 to the pro
          return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
      });
      // Start the consumer instance.
pushConsumer.start();
```

Step 4: View message details

After sending the message, a message ID (messageID) is generated. Developers can then verify this recently sent message on the "Message Query" page as shown below. Additionally, it allows viewing specific details and track information of the message. For more information, please refer to Message Query.



Time Range: Last 30 minutes, 2023-11-16 16:49:11 ~ 2023-11-16 17:19:11

Cluster: [redacted]

Topic: test

Query Method: Query all

□	Message ID	Message Tag	Message Key	Producer Address	Message Creation Ti...	Opera
□	[redacted]			30.167.149.54	2023-11-16 17:18:50	View D View N Verify Export

Console Guide
Cluster Management
Creating Cluster

Last updated：2024-01-17 18:06:24

Operation Scenarios

Cluster is a form of resource dimension of TDMQ for RocketMQ, where resources in disparate clusters are fully isolated, such as Topic, Group, and etc. Each cluster adheres to specific resource constraints, such as a cap on the number of Topic and the duration of message retention. Typically, specialized clusters are designated for individual environments such as development, testing, and production.

Steps

Create a cluster

1. Log in to the RocketMQ console and go to the **Cluster** page.
2. On the **Cluster** page, select the region and click **Create Cluster** to enter to the purchase page, where you can configure the relevant information.

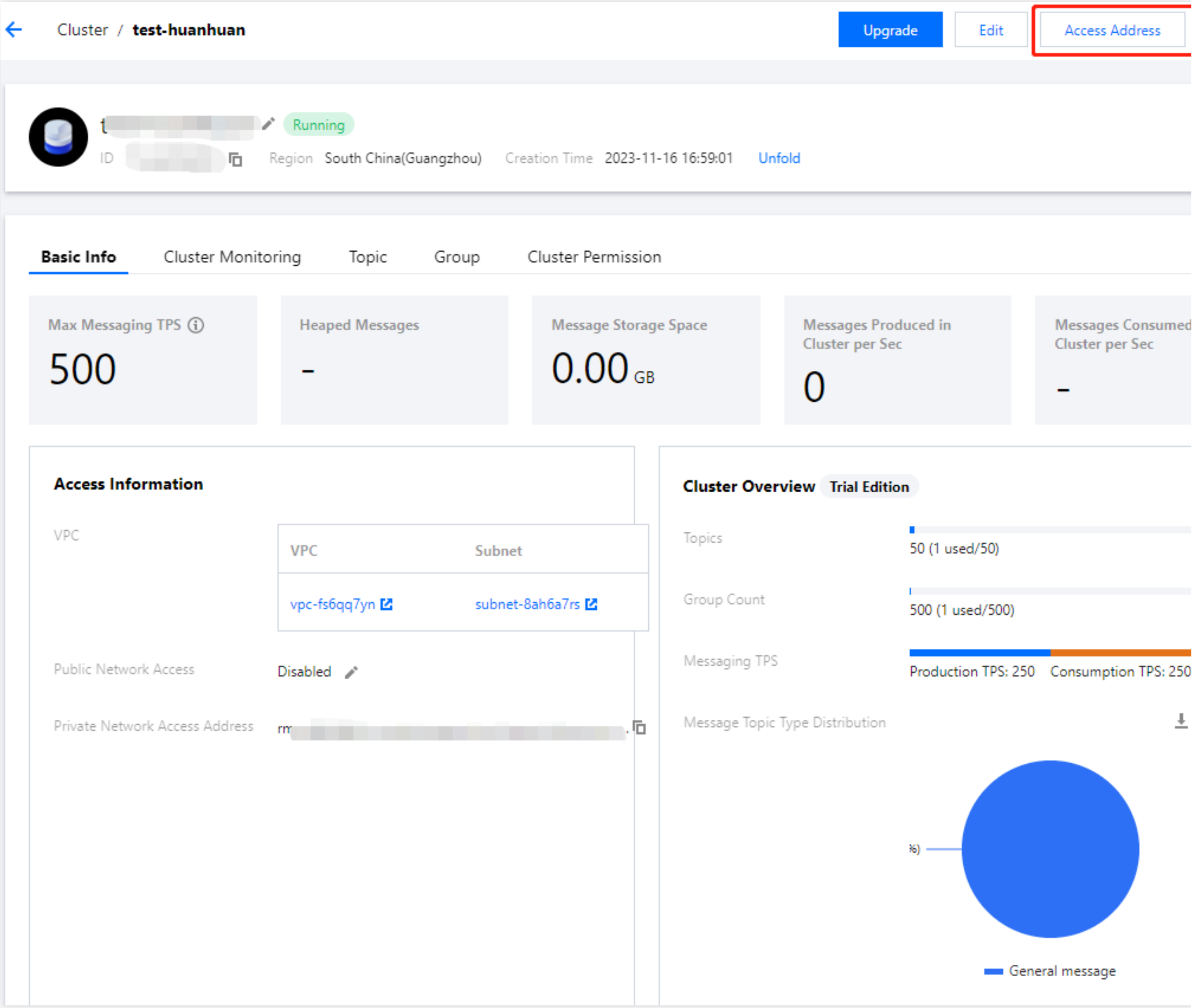
Parameter	Required or Not	Description
Cluster Version	Yes	Select 5.x
Billing Mode	Yes	The 5.x architecture cluster currently supports two billing modes: pay-as-you-go and monthly subscription.
Region	Yes	Select the region that is closest to your operations. Tencent Cloud products in various regions are unable to intercommunicate via the private network, and the region selection cannot be altered after purchasing. For instance, CVMs in the Guangzhou region cannot access clusters to the Shanghai region via the private network. In circumstances of required inter-regional private network communication, please refer to Peering Connections.
Cluster Specification	Yes	The following editions are currently supported: Trial, Basic, Pro, and Platinum. These differing cluster types exhibit variants in both performance, specifications, and functions. For more details, please refer to Product Series.

		Trial Edition: It is designed for corporate clients and individual developers to experience product functions in the initial development phase. For testing and experiencing purposes only, it is not recommended for use in a production environment. Basic Edition: It is designed for customers with moderate scale, providing basic capabilities such as message sending and receiving that are compatible with the open source RocketMQ community. Pro Edition: It is designed for enterprise-level clients with larger business scale, but have no specific requirements for the physical resources isolation. It supports the sharing underlying resources, with the highest available TPS specification reaching up to 150,000 TPS. Platinum Edition: It is designed for large corporates and large-scale business scenarios that need physical resource separation. This is the only specification type possessing the underlying resources. It offers the broadest TPS specification span, up to a maximum of a million TPS.
TPS Specification	Yes	TPS Specification contains the entire sum of message production and message consumption. Messages are measured in units of 4KB each, with special message types converted at specific proportions. For detailed rules, please refer to Billing Overview. Trial Edition: Supports up to 500. Basic Edition: Supports 1000, 2000, 4000, and 6000. Pro Edition: Supports a range between 4,000 to 150,000 TPS. Platinum Edition: Supports a range between 6,000 to 1,000,000 TPS.
Message Retention Time	Yes	The message retention time ranges from 24 to 72 hours. Regardless of whether the message has been consumed, it will be deleted after exceeding the retention time.
Virtual Private Cloud (VPC)	Yes	Authorize domain binding of the newly purchased cluster's access point to the Virtual Private Cloud (VPC).
Public Network Access	No	Activating the public network bandwidth will incur additional fees. Once activated, it can be disabled via the cluster management page. For more details, please refer to Public Network Fee Description.
Cluster Name	Yes	Enter the cluster name with 3-64 characters long, and can only contain numbers, letters, "-" and "_".
Tag	No	Tags are used for the classified management of resources from various dimensions. For detailed usage, please refer to Managing Resources Using Tags.
Terms and Conditions	Yes	Tick the box to confirm that you have read and accepted the <TDMQFOR ROCKETMQ SERVICE LEVEL AGREEMENT>.

3. Click **Buy Now**, wait for 3-5 minutes, and the cluster creation will be accomplished.

Follow-up steps:

1. Acquire the access address to obtain the connection information of the server.



Cluster / **test-huanhuan**

t[redacted] Running

ID [redacted] Region South China(Guangzhou) Creation Time 2023-11-16 16:59:01 Unfold

Max Messaging TPS ⓘ 500

Heaped Messages -

Message Storage Space 0.00 GB

Messages Produced in Cluster per Sec 0

Messages Consumed Cluster per Sec -

Access Information

VPC

VPC	Subnet
vpc-fs6qq7yn	subnet-8ah6a7rs

Public Network Access Disabled

Private Network Access Address rm[redacted]

Cluster Overview Trial Edition

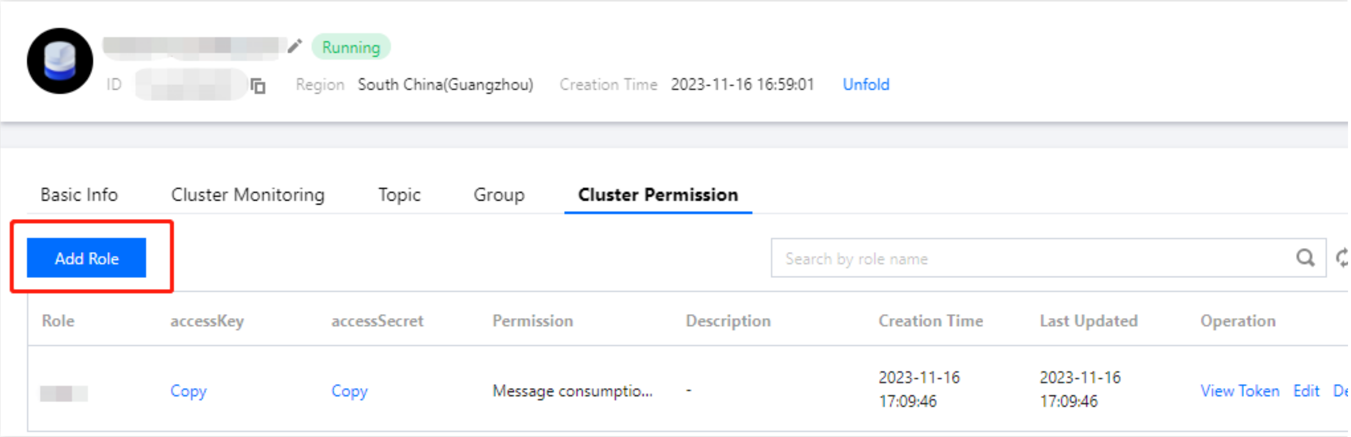
Topics 50 (1 used/50)

Group Count 500 (1 used/500)

Messaging TPS Production TPS: 250 Consumption TPS: 250

Message Topic Type Distribution

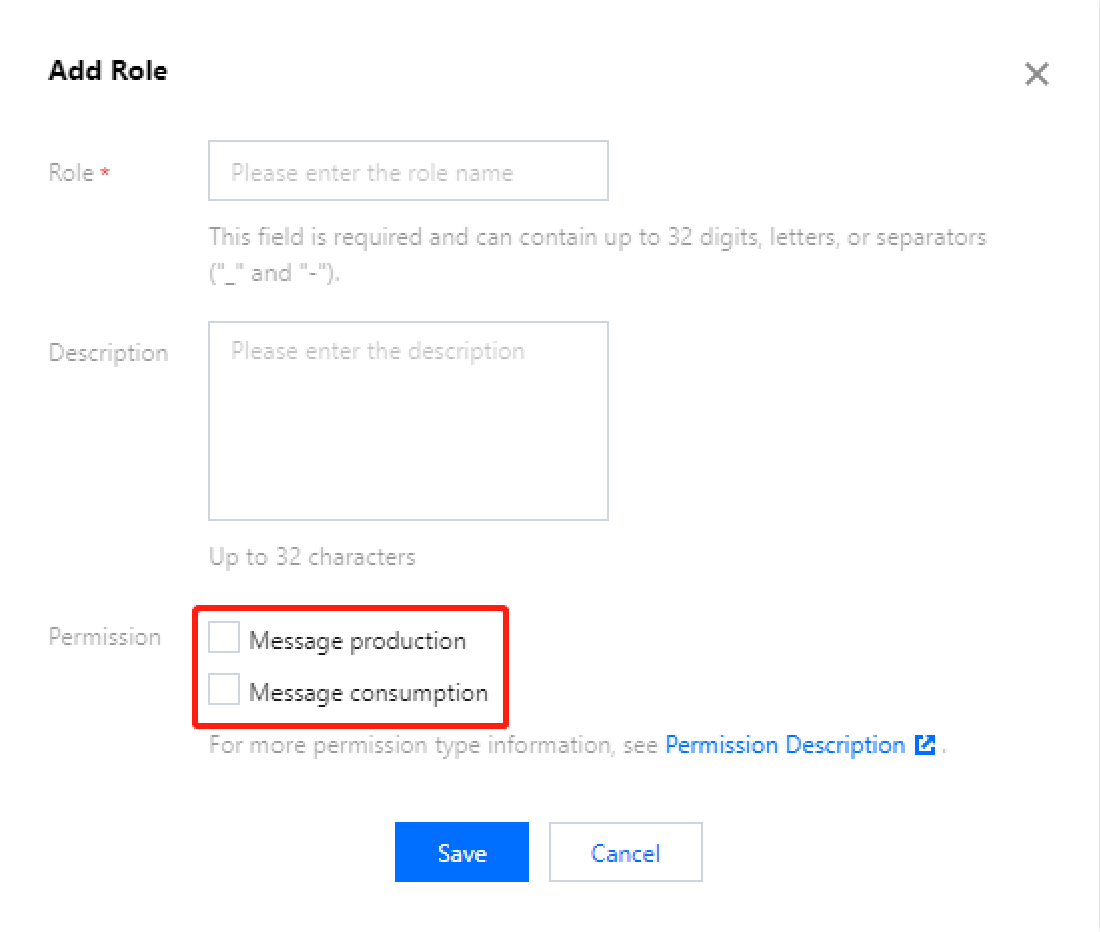
2. In the cluster, add a role and provide it with message production and consumption permissions for this cluster.



[redacted] Running

ID [redacted] Region South China(Guangzhou) Creation Time 2023-11-16 16:59:01 Unfold

Role	accessKey	accessSecret	Permission	Description	Creation Time	Last Updated	Operation
[redacted]	Copy	Copy	Message consumptio...	-	2023-11-16 17:09:46	2023-11-16 17:09:46	View Token Edit De



Add Role

Role * Please enter the role name

This field is required and can contain up to 32 digits, letters, or separators ("_" and "-").

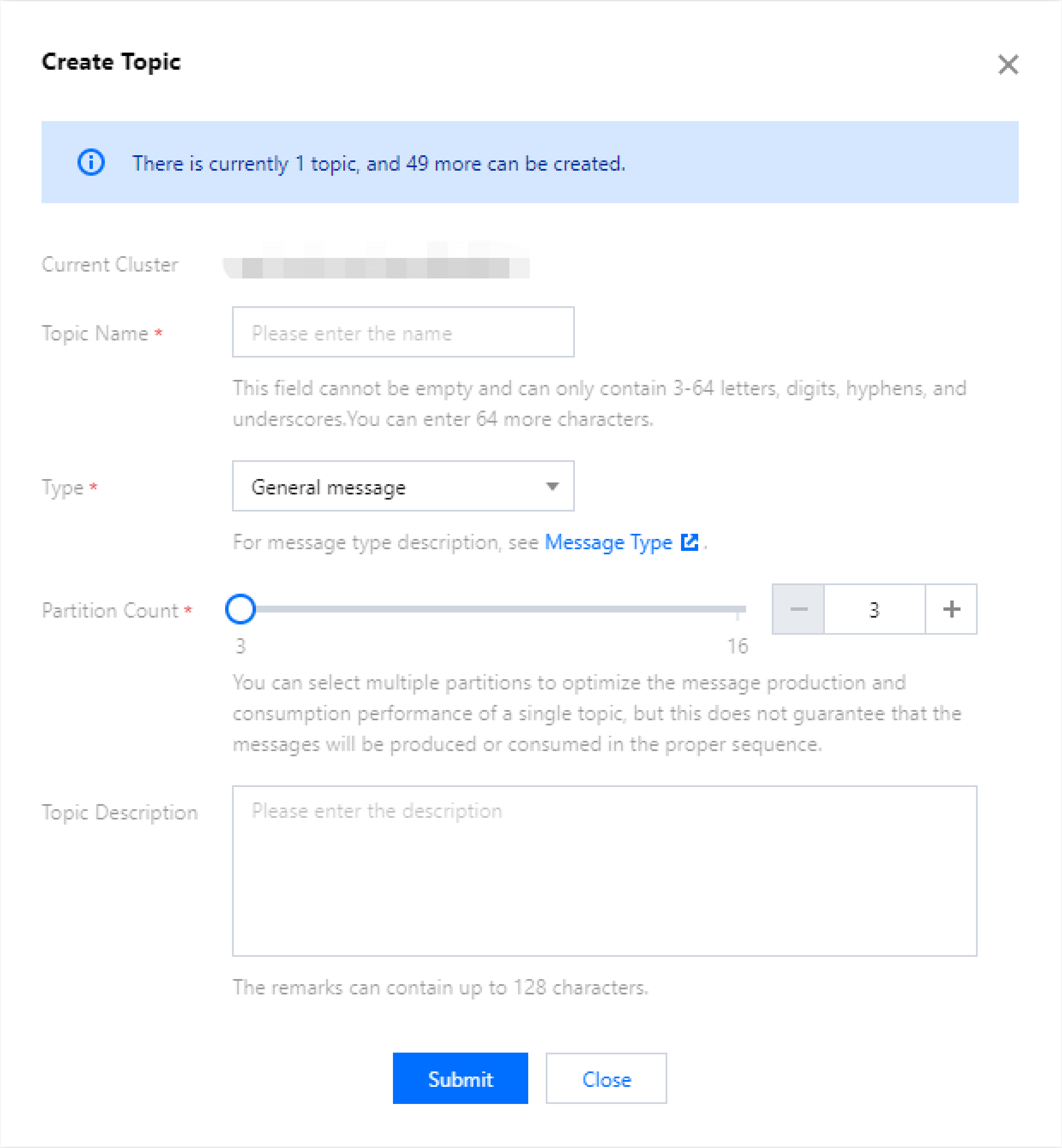
Description Please enter the description

Up to 32 characters

Permission □ Message production
□ Message consumption

For more permission type information, see Permission Description.

3. In the cluster, complete **Create Topic**, and **Create Group**.



Create Topic

ⓘ There is currently 1 topic, and 49 more can be created.

Current Cluster [redacted]

Topic Name * Please enter the name

This field cannot be empty and can only contain 3-64 letters, digits, hyphens, and underscores.You can enter 64 more characters.

Type * General message

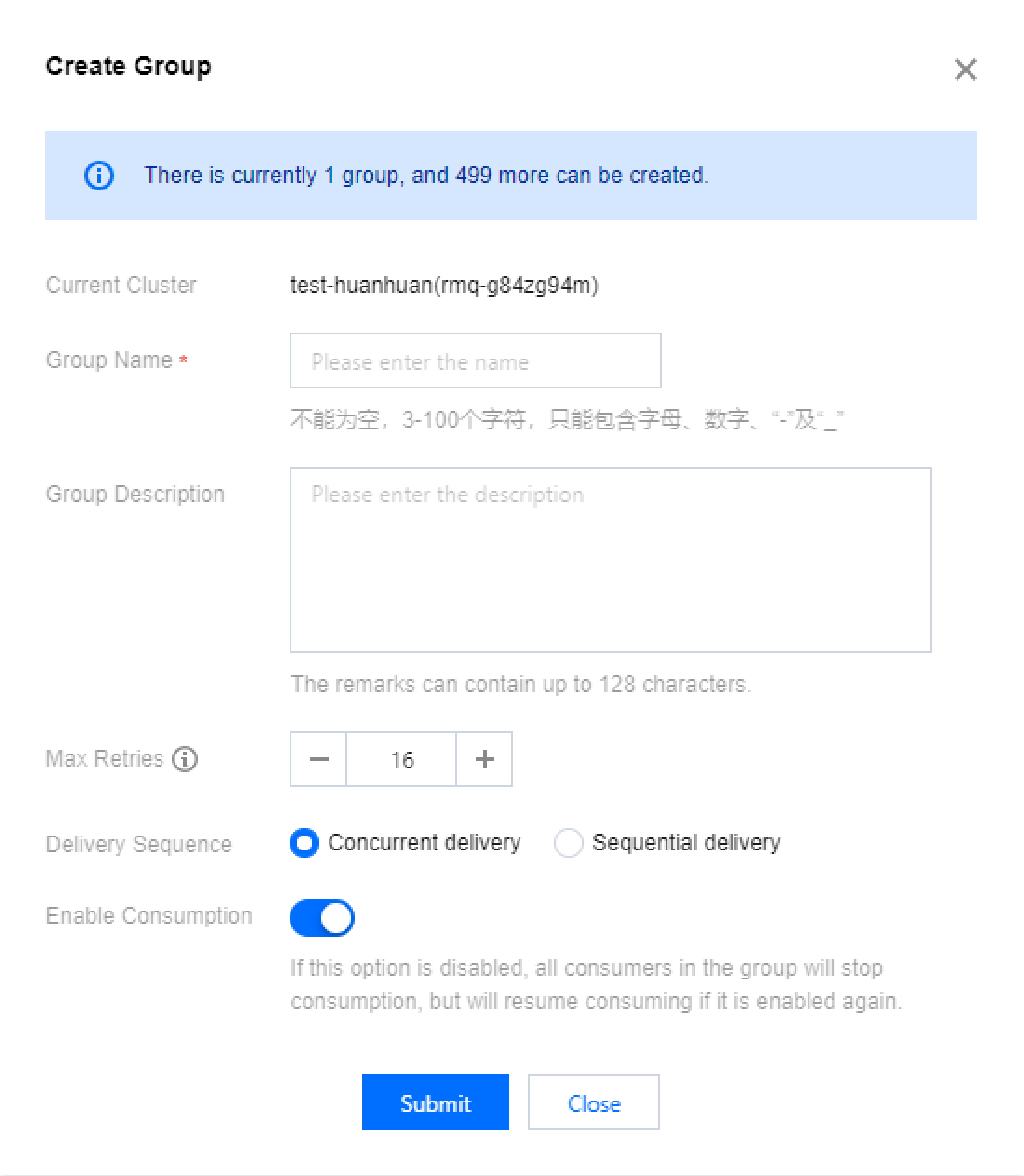
For message type description, see Message Type.

Partition Count * 3 16 − 3 +

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 Please enter the description

The remarks can contain up to 128 characters.



Create Group

There is currently 1 group, and 499 more can be created.

Current Cluster test-huanhuan(rmq-g84zg94m)

Group Name * Please enter the name

不能为空，3-100个字符，只能包含字母、数字、"-"及"_"

Group Description Please enter the description

The remarks can contain up to 128 characters.

Max Retries 16

Delivery Sequence ● Concurrent delivery ○ Sequential delivery

Enable Consumption

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

4. Compose a Demo as directed in the **SDK documentation**, configure the connection details, and proceed with message production and consumption.

View cluster details

On the **Cluster** page, click the cluster ID to enter to its details page. In that page, you can inquire the following information:

Basic information of the cluster (cluster name/ID, region, creation time, description, and resource tag).

Cluster data statistics: Displays the current cluster's message consumption rate, message production rate, heaped messages amount, the cluster's production traffic per second, and the cluster's consumption traffic per second in chosen time frame.

Access information: Displays the access point information of both private and public networks.

Cluster overview: Displays the quantities of various resources in the current cluster, the usage of resource quota, and the distribution of message types, and etc.

Top cluster resource consumption: Displays the ranking of Group and Topic in the current cluster that occupy the primary resource, including the rankings for the amount of heaped Group and badmails, the rankings for Topic production and consumption speed, and the rankings of Topics that occupy storage space.

Upgrade Clusters

Last updated：2024-01-17 17:47:26

Operation Scenarios

If the present cluster specifications fail to meet your business requirements, you can enhance both your cluster and TPS specifications in the console.

Note:

At present, only upgrading of cluster and TPS specifications is permissible. Downgrading is not yet supported.

During the upgrading process of RocketMQ clusters, whether for cluster or TPS specifications, Tencent Cloud provides a seamless and imperceptible upgrading experience, obviating the need for any downtime handling on the client's application side.

Steps

There are two entries for upgrading your cluster specifications:

Entry one: Log in to the RocketMQ Console***,*** and on the **Cluster List** page, click **Upgrade** in the operation column for the cluster where adjustments need to be made.

Entry two: Log in to the RocketMQ Console**,** and on the **Cluster Information** page, click **Upgrade** in the upper right corner.

Target Cluster Specifications: Only the upgrading of cluster specifications is supported; downgrading is not supported.

Target TPS Specifications: Only the upgrading of TPS specifications is supported; downgrading is not supported.

Delete Cluster

Last updated：2024-01-17 17:47:38

Operation Scenarios

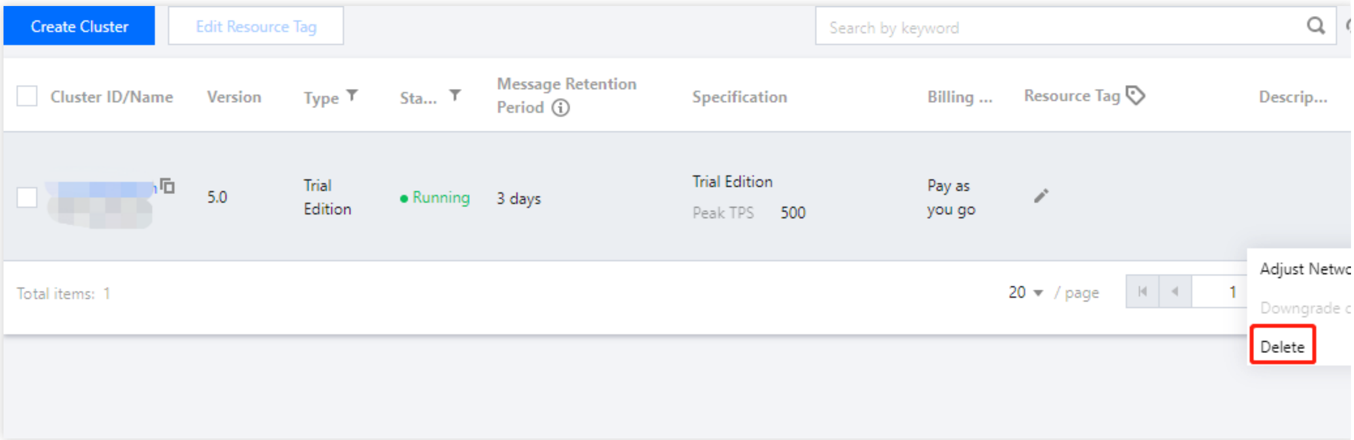
When you no longer require a RocketMQ cluster, you can manually terminate it via the console to prevent extra charges.

Note:

After deletion, all the configurations under this cluster will be erased and are irretrievable. Please proceed with caution.

Steps

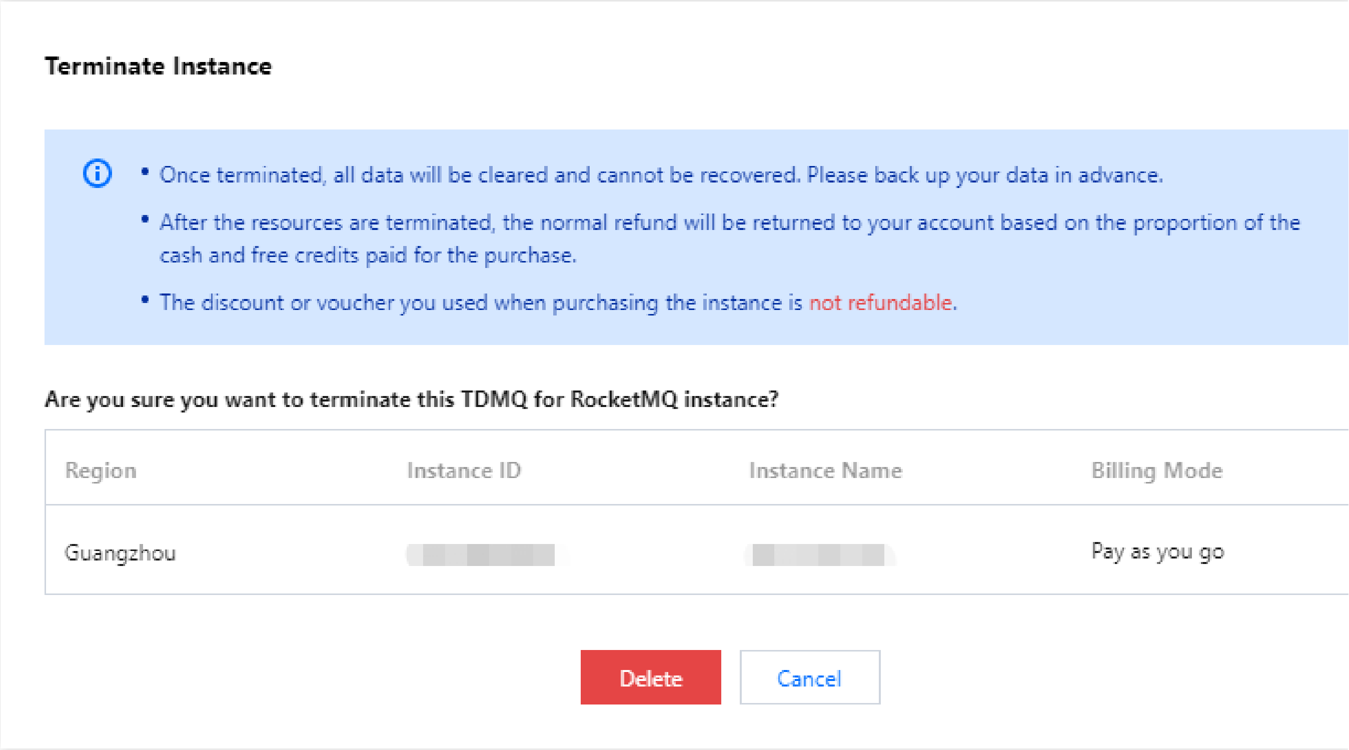
1. Log in to the Console for RocketMQ.
2. On the **Cluster** list page, click **More** in the operation column of the instance you wish to delete, then click **Delete**.



□ Cluster ID/Name	Version	Type	Sta...	Message Retention Period	Specification	Billing ...	Resource Tag	Descrip...
□ [redacted]	5.0	Trial Edition	● Running	3 days	Trial Edition Peak TPS 500	Pay as you go		

Total items: 1 20 / page 1

3. Within the deletion confirmation dialog box, click **Delete** to successfully delete the cluster.



Terminate Instance

- Once terminated, all data will be cleared and cannot be recovered. Please back up your data in advance.
- After the resources are terminated, the normal refund will be returned to your account based on the proportion of the cash and free credits paid for the purchase.
- The discount or voucher you used when purchasing the instance is not refundable.

Are you sure you want to terminate this TDMQ for RocketMQ instance?

Region	Instance ID	Instance Name	Billing Mode
Guangzhou	[redacted]	[redacted]	Pay as you go

Note:

To prevent misoperations from causing the deletion of internal data (such as topics, groups, and roles) within the cluster, the console will verify your cluster resources when deleting the cluster. If any resources, such as topics or groups, have not been deleted, the cluster cannot be removed.

Adjust Public Network Bandwidth

Last updated：2024-01-17 17:47:50

Operation Scenarios

After cluster creation, you can enable or disable public network access by adjusting the public network bandwidth, modify the size of the public network bandwidth, and set public network security policies to restrict user access.

Steps

There are two entries to adjust the public network bandwidth:

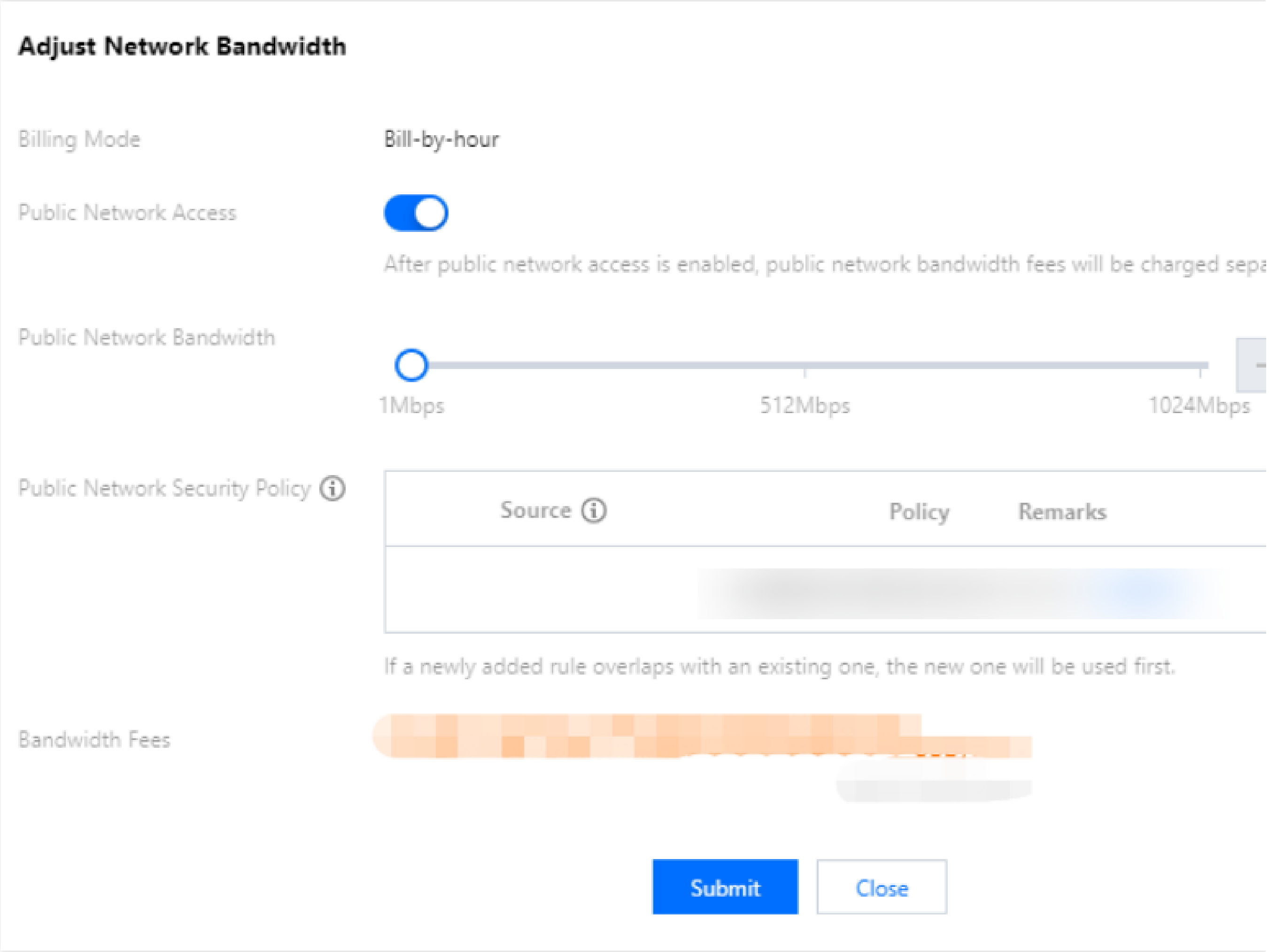
Entry One: On the Cluster List page, click **More** > **Adjust Network Bandwidth** in the operation column.

Entry Two: On the **Basic Information** page of the cluster, click **Edit** under the **Public Network Access** in the Access Information module.

Public Network Access: After enabling the public network bandwidth, additional costs will be incurred. For more billing details, please refer to Public Network Billing Overview.

Public Network Bandwidth: Select the desired size of the public network bandwidth to be adjusted.

Public Network Security Policy: Specify the IPs allowed to access. If no security policy is set, all IPs will be denied access by default. If a newly added rule overlaps with an existing rule, the new one will be used first.



Adjust Network Bandwidth

Billing Mode: Bill-by-hour

Public Network Access: ●

After public network access is enabled, public network bandwidth fees will be charged sepa

Public Network Bandwidth:

Public Network Security Policy ⓘ

Source ⓘ	Policy	Remarks
[redacted]		

If a newly added rule overlaps with an existing one, the new one will be used first.

Bandwidth Fees: [redacted]

Role and Permission

Last updated：2024-01-17 17:48:06

Definition

The "role" in RocketMQ is a proprietary concept in RocketMQ and distinct from Tencent Cloud's "role". It is the smallest unit of authority partitioning performed by the user in RocketMQ. Users can assign different production and consumption permissions under multiple clusters to numerous roles. Each role has its unique corresponding access key. Users can access RocketMQ for message production and consumption by adding these keys into the client.

The application scenarios are as follows:

Users require to securely employ RocketMQ for message production and consumption.

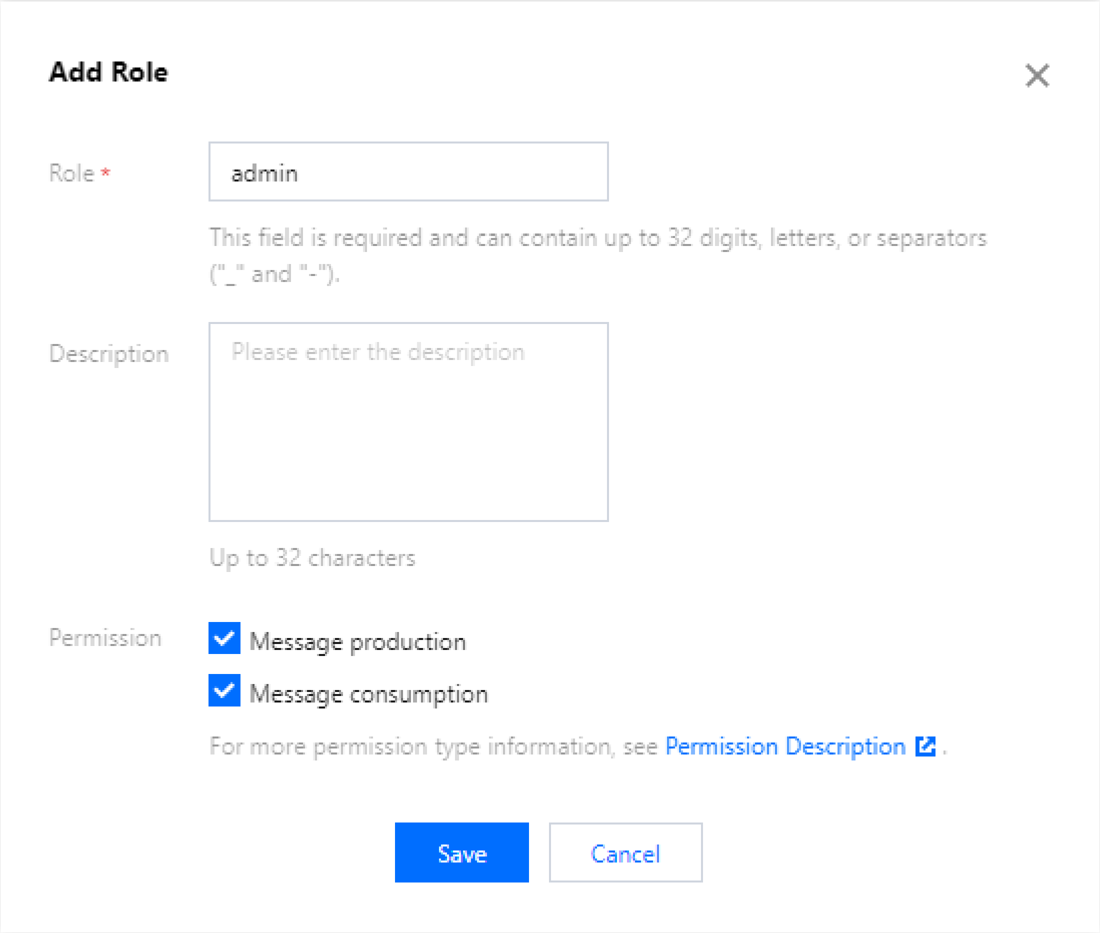
Users require to define production consumption permissions of varying roles for diverse clusters.

For instance, a company comprises Department A and Department B. The system operated by Department A generates transaction data. The system operated by Department B performs data analysis and presentation based on these transaction data. To comply with the principle of least permission, two roles could be configured, providing Department A with exclusive permission to produce messages in the transaction system cluster, while Department B would be granted permission solely to consume messages. Such a setup significantly avoids issues related to data confusion or business dirty data that could arise from ambiguities in permission.

Steps

Add roles and assign permissions

1. Log in to the Console for RocketMQ.
2. From the left sidebar, select **Cluster List**. After selecting the region, click the "ID" of the desired cluster to configure roles for and proceed to the basic information page.
3. At the top of the page, select the **Cluster Permissions** tab. Click **Add Role**, enter a role name, and configure production and consumption permissions.
4. Click **Save** to complete role creation.



Add Role

Role * admin

This field is required and can contain up to 32 digits, letters, or separators ("_" and "-").

Description: Please enter the description

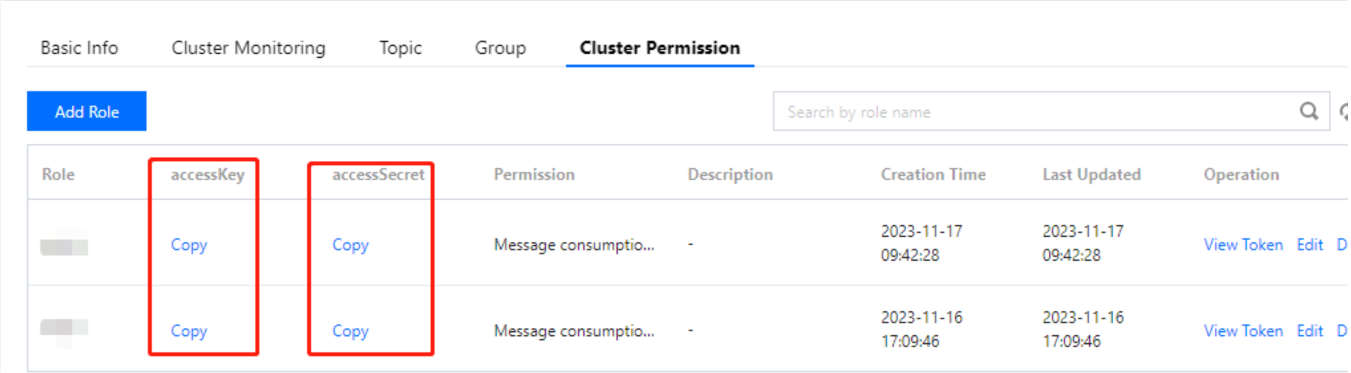
Up to 32 characters

Permission ☑ Message production
☑ Message consumption

For more permission type information, see Permission Description.

Verify the effectiveness of permissions

1. Copy the role key from the cluster permission list.



Role	accessKey	accessSecret	Permission	Description	Creation Time	Last Updated	Operation
[redacted]	Copy	Copy	Message consumptio...	-	2023-11-17 09:42:28	2023-11-17 09:42:28	View Token Edit D
[redacted]	Copy	Copy	Message consumptio...	-	2023-11-16 17:09:46	2023-11-16 17:09:46	View Token Edit D

Note:

Key leakage may lead to data leakage. Please keep your key well.

2. Add the copied role key into the parameters of the client. For guidance on how to add the key parameter in the client code, please refer to Sample Code from RocketMQ.

The following provides a suggested approach.

2.1 Declare two fields, `ACL_SECRET_KEY` and `ACL_SECRET_ACCESS`. When using various frameworks, it is recommended to read these from the configuration files.

```
private static final String ACL_ACCESS_KEY = "eyJr****";
private static final String ACL_SECRET_KEY = "xxx"; /
```

2.2 Declare a static function to load a RocketMQ Client `RPCHook` object.

```
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.

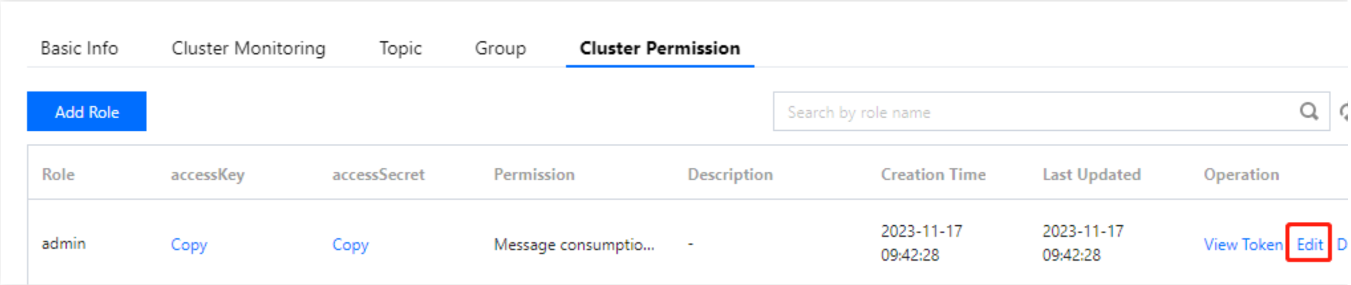
Here is a code example for creating a `producer` :

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

3. Run the configured client to draw upon topic resources in the relevant cluster, adhering to recently established permissions for production or consumption. If no 'permission denied' error message is generated, this denote a successfully set configuration.

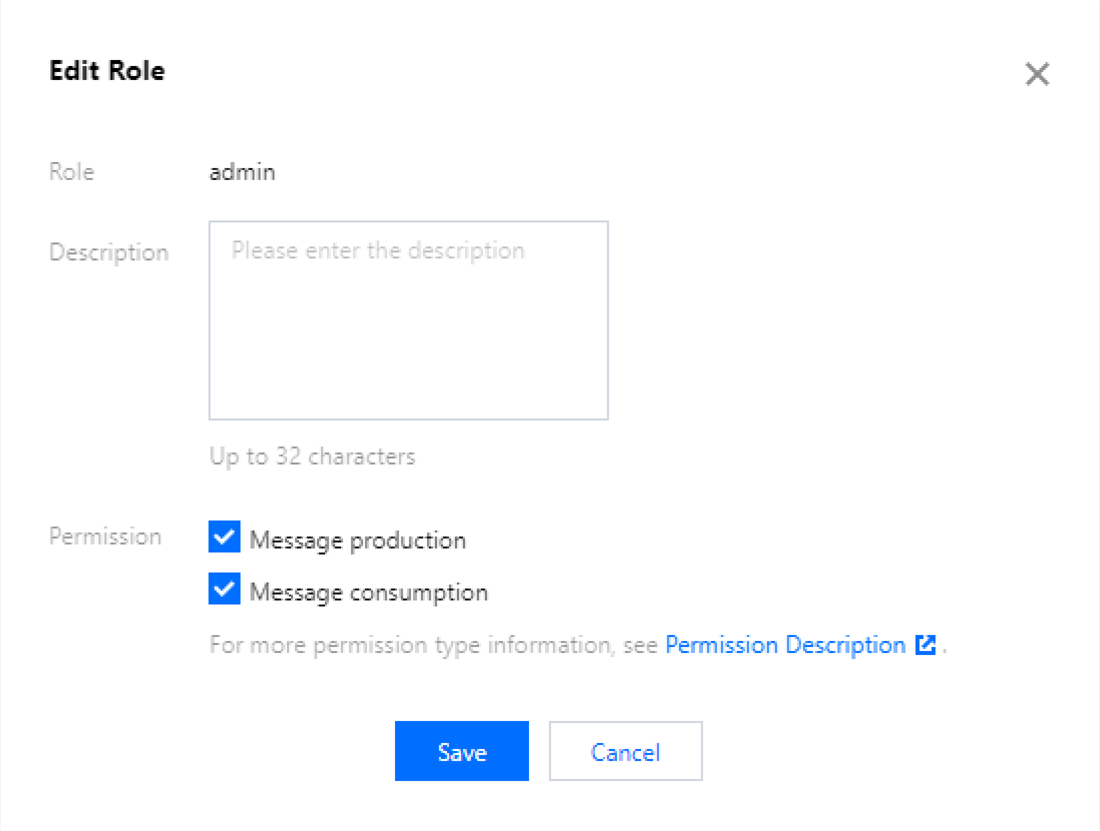
Edit permission

1. In the cluster permissions list, locate the desired role to edit the permission, then click **Edit** in the action column.



Role	accessKey	accessSecret	Permission	Description	Creation Time	Last Updated	Operation
admin	Copy	Copy	Message consumptio...	-	2023-11-17 09:42:28	2023-11-17 09:42:28	View Token Edit D

2. In the edit pop-up window, after modifying the permission details, click **Save**.



Edit Role

Role: admin

Description: Please enter the description

Up to 32 characters

Permission: ☑ Message production ☑ Message consumption

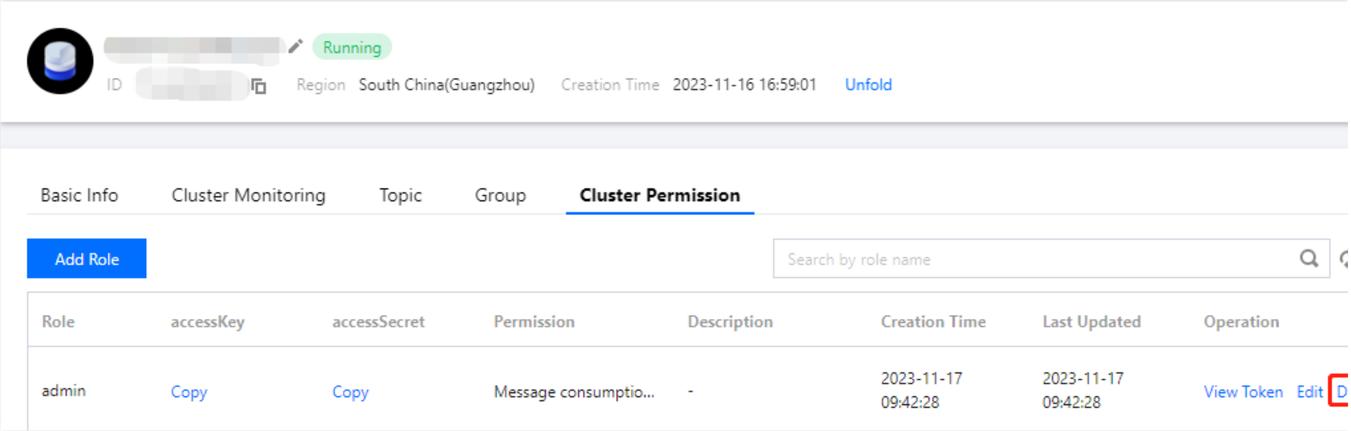
For more permission type information, see Permission Description .

Delete role

Note:

After role deletion, any keys (accessKey and accessSecret) used to produce or consume messages under the role will become defunct instantly. Please ensure that the current line of business no longer relies on this role for message production or consumption, failing which, client exceptions could emerge due to an inability to produce or consume messages.

1. In the cluster permission list, locate the role that requires permission deletion. Click **Delete** in the action column.

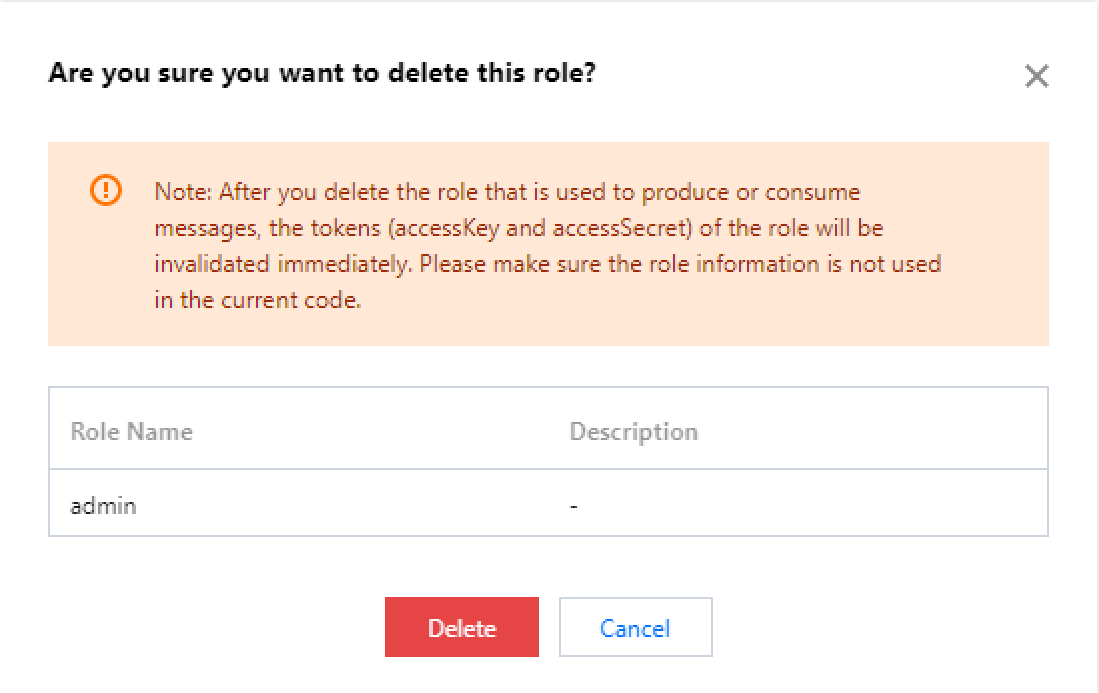


[redacted] Running

ID [redacted] Region South China(Guangzhou) Creation Time 2023-11-16 16:59:01 Unfold

Role	accessKey	accessSecret	Permission	Description	Creation Time	Last Updated	Operation
admin	Copy	Copy	Message consumptio...	-	2023-11-17 09:42:28	2023-11-17 09:42:28	View Token Edit D

2. In the deletion pop-up window, click **Delete** to remove the role.



Are you sure you want to delete this role?

Note: After you delete the role that is used to produce or consume messages, the tokens (accessKey and accessSecret) of the role will be invalidated immediately. Please make sure the role information is not used in the current code.

Role Name	Description
admin	-

Switching Cluster Billing Mode

Last updated：2024-06-13 16:07:35

Operation Scenarios

To make it easier for you to use RocketMQ, Tencent Cloud has opened the feature allowing the conversion between pay-as-you-go and monthly subscription billing modes for RocketMQ. This document introduces the operations for switching billing modes in the RocketMQ console.

Conversion Rules

After the successful conversion of the billing method and payment, the cluster will be billed in the new billing mode immediately, with the new cluster's start time being the time of successful conversion.

After the conversion of the cluster billing mode, the public network bandwidth will also automatically switch its billing mode.

RocketMQ does not support the five-day unconditional refund after switching from pay-as-you-go to monthly subscription billing.

Use Limits

Only clusters in the running status support switching billing modes. Clusters in isolating/delivery/abnormal statuses do not support changing the billing mode.

Directions

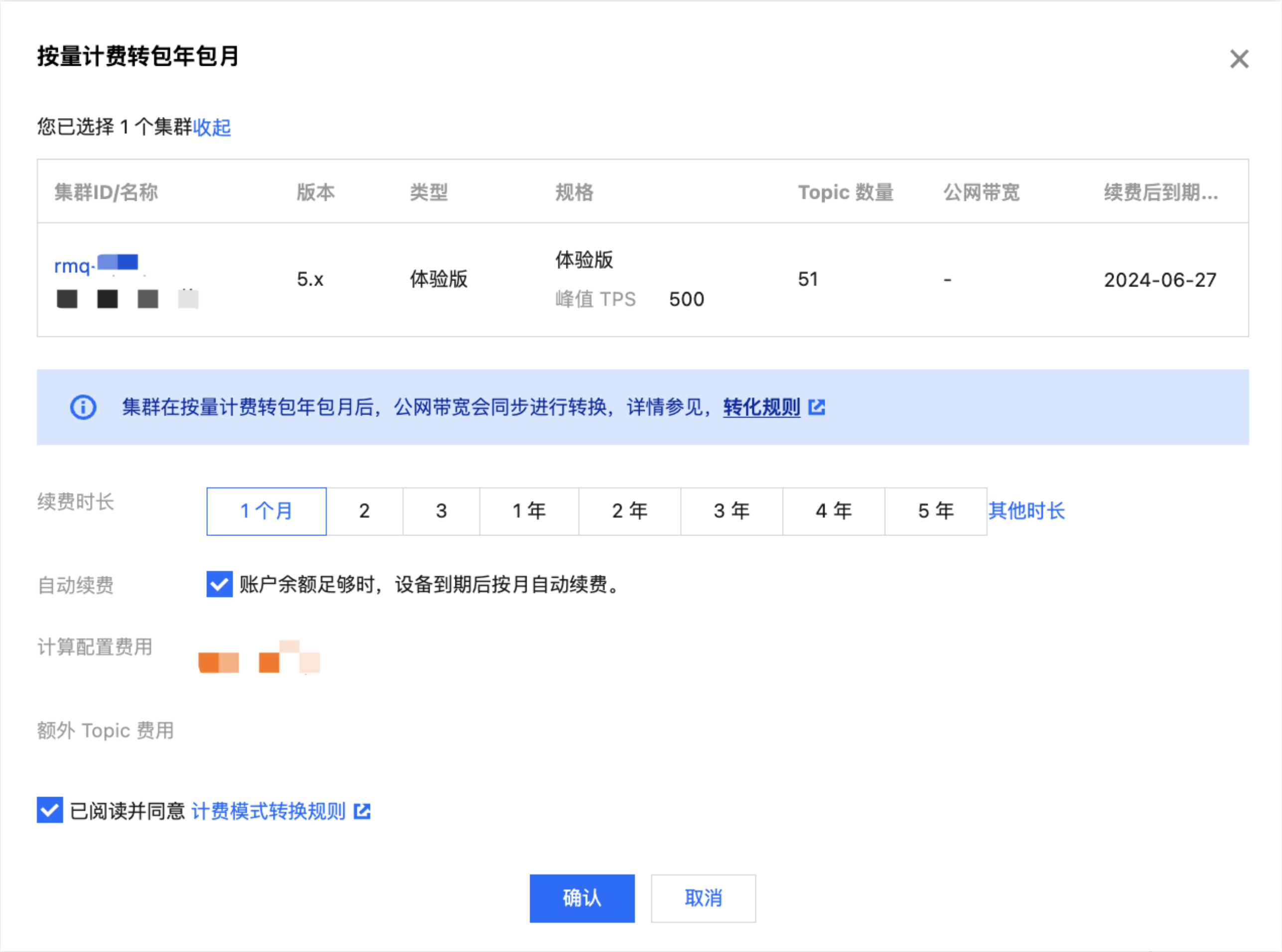
Switch from Pay-as-You-Go to Monthly Subscription

Switch from Monthly Subscription to Pay-as-You-Go

1. log in to the RocketMQ console.

2. In the left sidebar, click **Cluster Management**, then click **More** in the operation column of the target cluster > **Switch to Monthly Billing**.

3. In the pop-up pay-as-you-go to monthly billing window, set the renewal duration and whether to enable auto-renewal according to your actual needs.



按量计费转包年包月 ×

您已选择 1 个集群收起

集群ID/名称	版本	类型	规格	Topic 数量	公网带宽	续费后到期...
rmq-[redacted]	5.x	体验版	体验版 峰值 TPS 500	51	-	2024-06-27

ⓘ 集群在按量计费转包年包月后，公网带宽会同步进行转换，详情参见，转化规则

续费时长: 1 个月 | 2 | 3 | 1 年 | 2 年 | 3 年 | 4 年 | 5 年 | 其他时长

自动续费: ☑ 账户余额足够时，设备到期后按月自动续费。

计算配置费用: [redacted]

额外 Topic 费用

☑ 已阅读并同意 计费模式转换规则

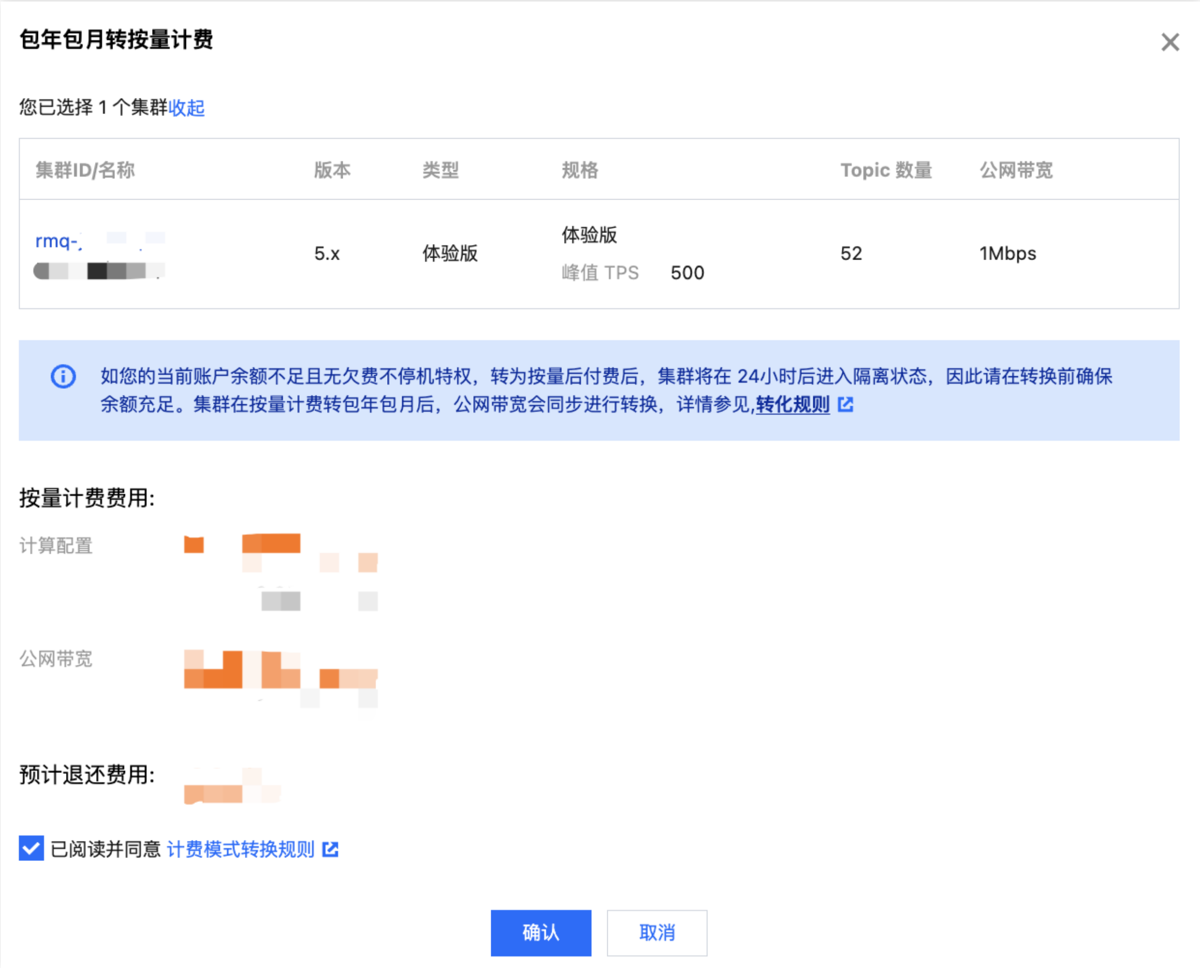
4. Check the box for **I have read and agree to the billing mode conversion rules**, and then click **Confirm**.

5. Follow the prompts on the page to complete the payment and finish the conversion operation.

1. log in to RocketMQ Console.

2. In the left sidebar, click **Cluster Management**, then click **More** in the operation column of the target cluster > **Switch to Pay-as-You-Go**.

3. In the pop-up window, set the renewal duration and whether to enable auto-renewal according to your actual needs.



包年包月转按量计费

您已选择 1 个集群收起

集群ID/名称	版本	类型	规格	Topic 数量	公网带宽
rmq-[redacted]	5.x	体验版	体验版 峰值 TPS 500	52	1Mbps

如您的当前账户余额不足且无欠费不停机特权，转为按量后付费后，集群将在 24小时后进入隔离状态，因此请在转换前确保余额充足。集群在按量计费转包年包月后，公网带宽会同步进行转换，详情参见,**转化规则**

按量计费费用:

计算配置 [redacted]

公网带宽 [redacted]

预计退还费用: [redacted]

☑ 已阅读并同意 计费模式转换规则

4. Check the box for **I have read and agree to the billing mode conversion rules**, then click **Confirm** to complete the conversion.

Topic Management

Last updated：2024-04-25 15:42:53

Operation Scenarios

Topic is a central concept in the TDMQ for RocketMQ. It is typically used for the organization and centralized management of various messages generated by the system. For instance, messages associated with transactions can be congregated within a Topic titled "trade", available for subscription by other consumers.

In practical application scenarios, a Topic often corresponds to a business aggregation. Developers determine to design different Topics based on the design of their own system and data architecture.

This guide provides instructions on how to use Topics for the categorization and management of messages when using TDMQ for RocketMQ.

Steps

Establish a topic

1. Log in to the RocketMQ Console*.*

2. Select the **Topic Management** tab on the left sidebar, choose the region and cluster, then click **Create** to go to the Topic creation page.

3. In the Create Topic dialog box, please complete the following details.

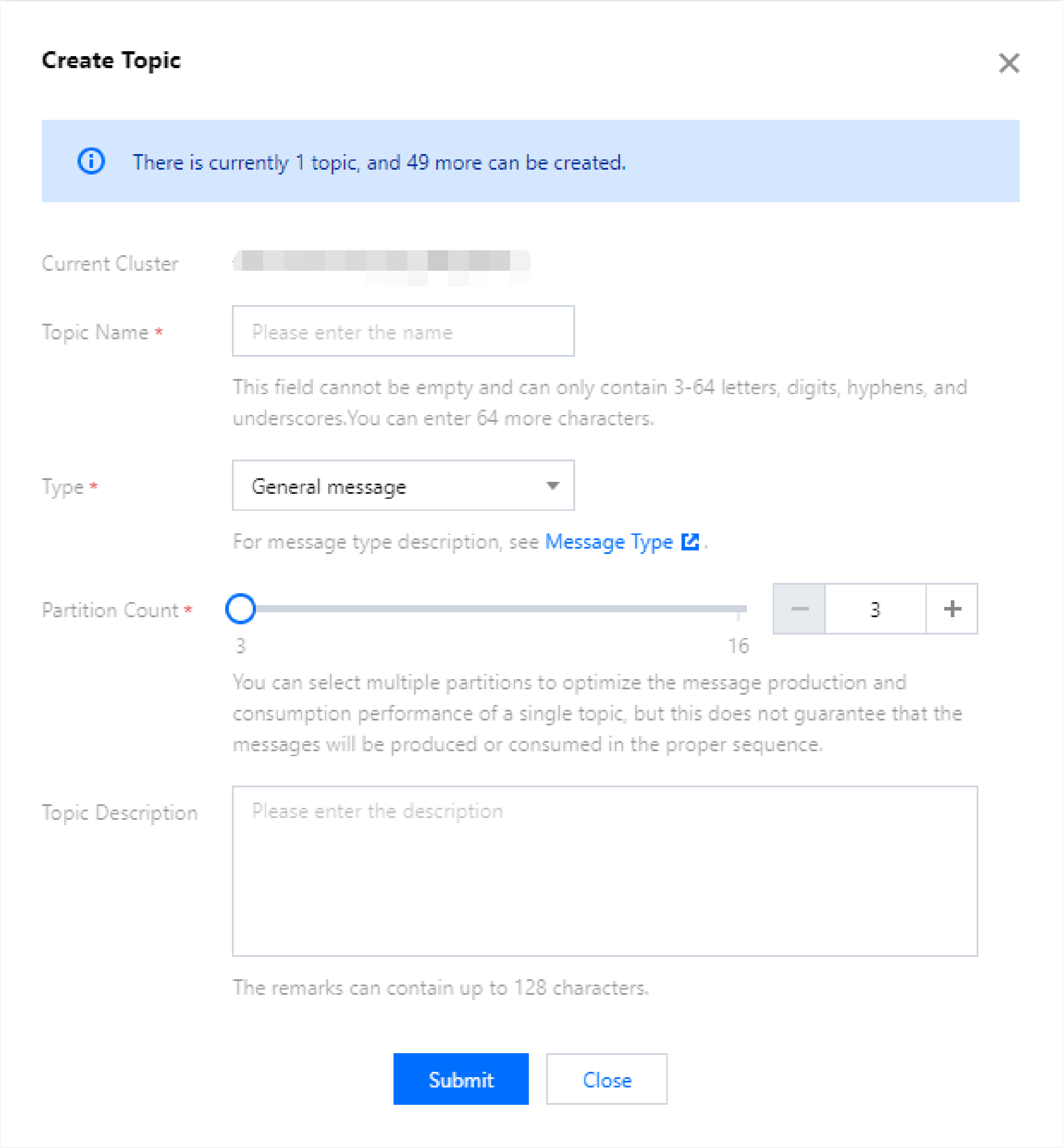
Topic Name: Input the name of the topic (cannot be modified after creation), containing 3-64 characters including only letters, numbers, "-" and "_".

Type: Select the message type, including general, sequential, delayed, and transaction messages. (For more information, see Message Type).

Partition Count: Select the number of partitions, supporting up to 16 partitions. Using numerous partitions can enhance the production and consumption performance of a single topic, while the sequentiality cannot be assured.

Topic Description: Provide a description for the Topic, limited to a maximum of 128 characters.

4. Click **Submit** and the newly created topic can be viewed in the topic list.



Create Topic

There is currently 1 topic, and 49 more can be created.

Current Cluster [redacted]

Topic Name * Please enter the name

This field cannot be empty and can only contain 3-64 letters, digits, hyphens, and underscores.You can enter 64 more characters.

Type * General message

For message type description, see Message Type .

Partition Count * 3 16 − 3 +

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.

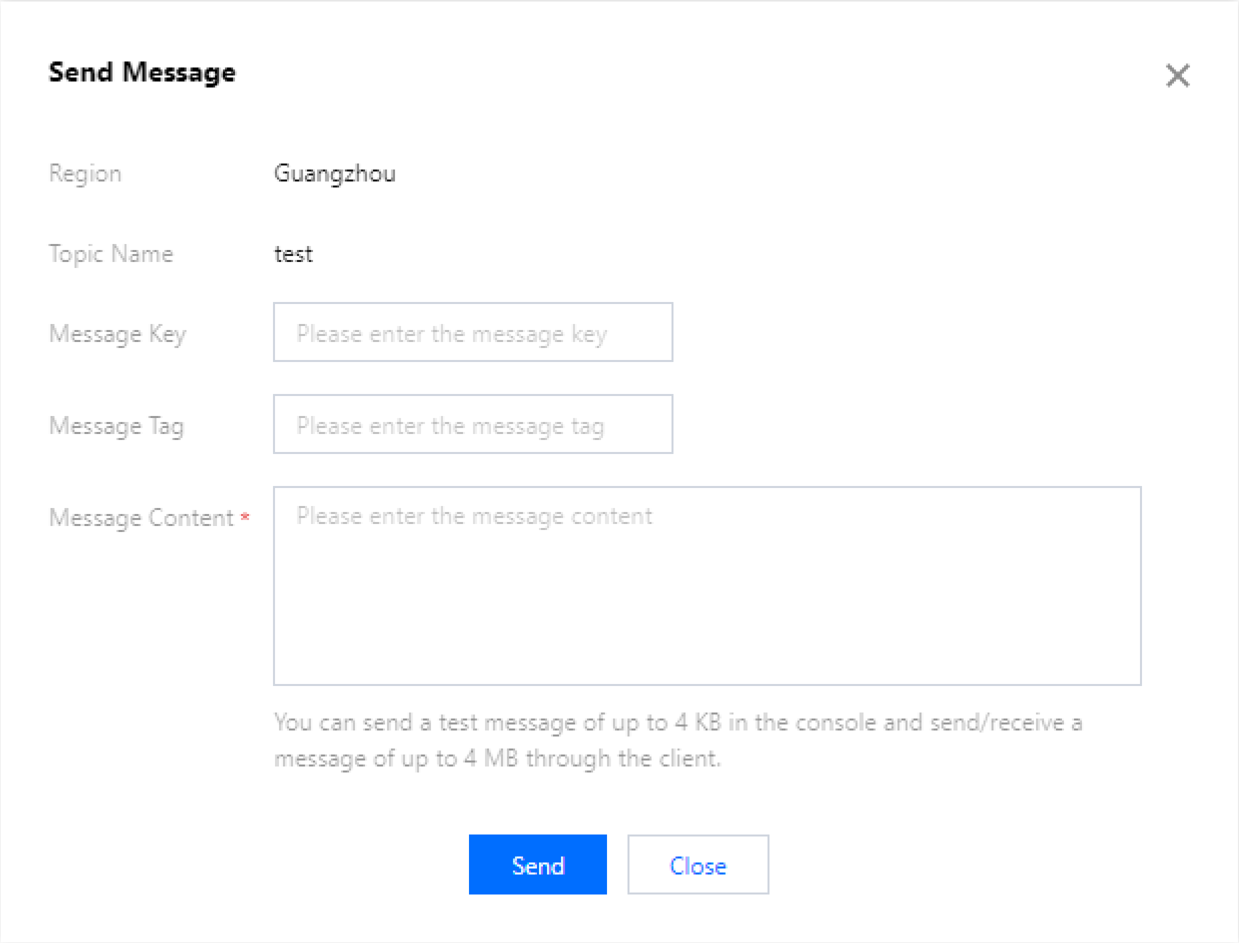
Topic Description Please enter the description

The remarks can contain up to 128 characters.

Send test messages

Messages can be manually transmitted to specific topics through actions performed in the RocketMQ console.

1. In **Topic Management** list, click **Send Message** in the action bar of the intended topic.
2. In the pop-up window, enter the message key, message tag, and message content, then click **Send**.



Send Message

Region Guangzhou

Topic Name test

Message Key: Please enter the message key

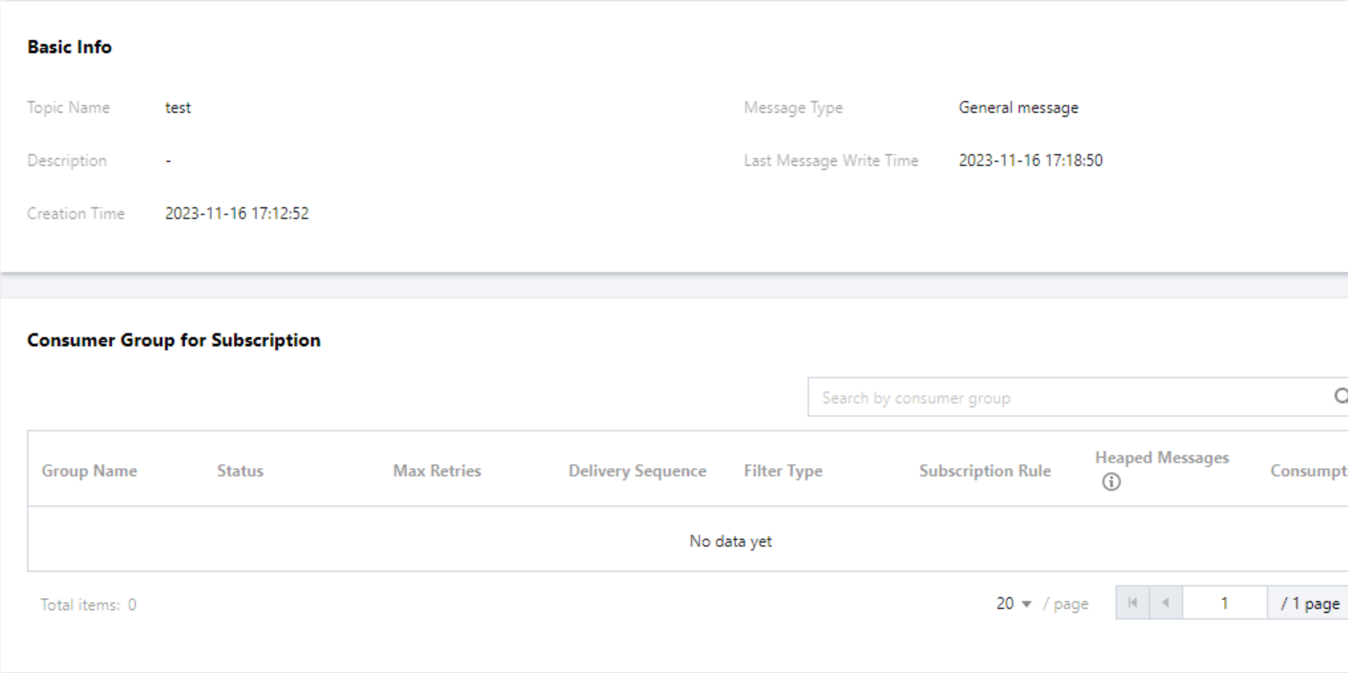
Message Tag: Please enter the message tag

Message Content *: Please enter the message content

You can send a test message of up to 4 KB in the console and send/receive a message of up to 4 MB through the client.

View subscribed groups

1. In **Topic Management** list, click the "ID" of the desired topic.
2. You will be redirected to the Group list page, which displays the information of Groups subscribed to that specific Topic.



Basic Info

Topic Name	test	Message Type	General message
Description	-	Last Message Write Time	2023-11-16 17:18:50
Creation Time	2023-11-16 17:12:52		

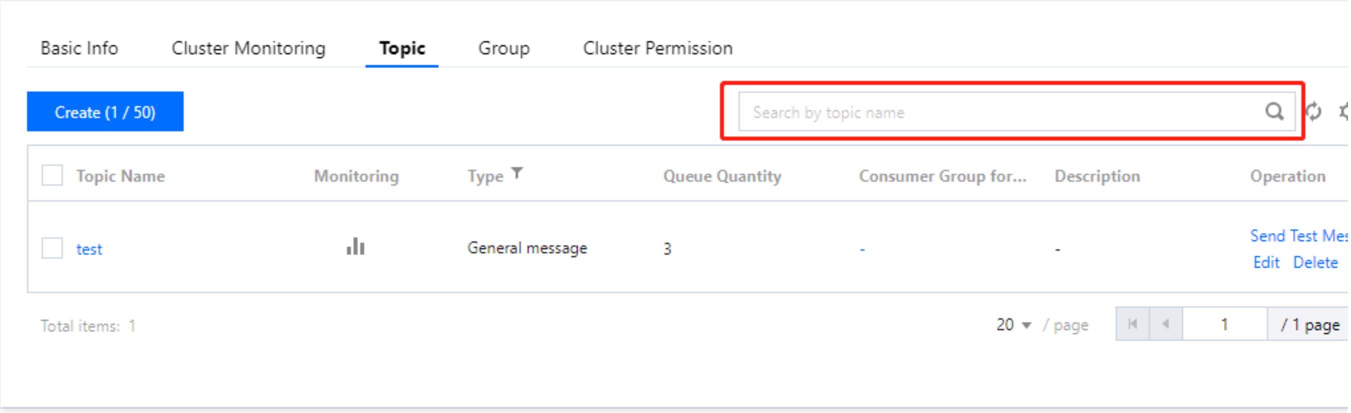
Consumer Group for Subscription

Group Name	Status	Max Retries	Delivery Sequence	Filter Type	Subscription Rule	Heaped Messages ⓘ	Consumpt
No data yet							

Total items: 0

Query a topic

You can search by a Topic name in the search box located at the top-right corner of the **Topic Management** list page. TDMQ for RocketMQ will perform a fuzzy match and present the corresponding search results.

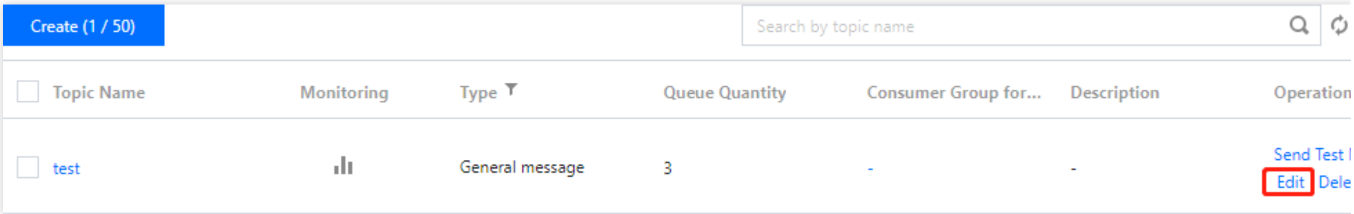


□ Topic Name	Monitoring	Type	Queue Quantity	Consumer Group for...	Description	Operation
□ test		General message	3	-	-	Send Test Mes Edit Delete

Total items: 1

Edit a topic

1. In **Topic Management** list, locate the Topic that requires editing, and click **Edit** in the action bar.



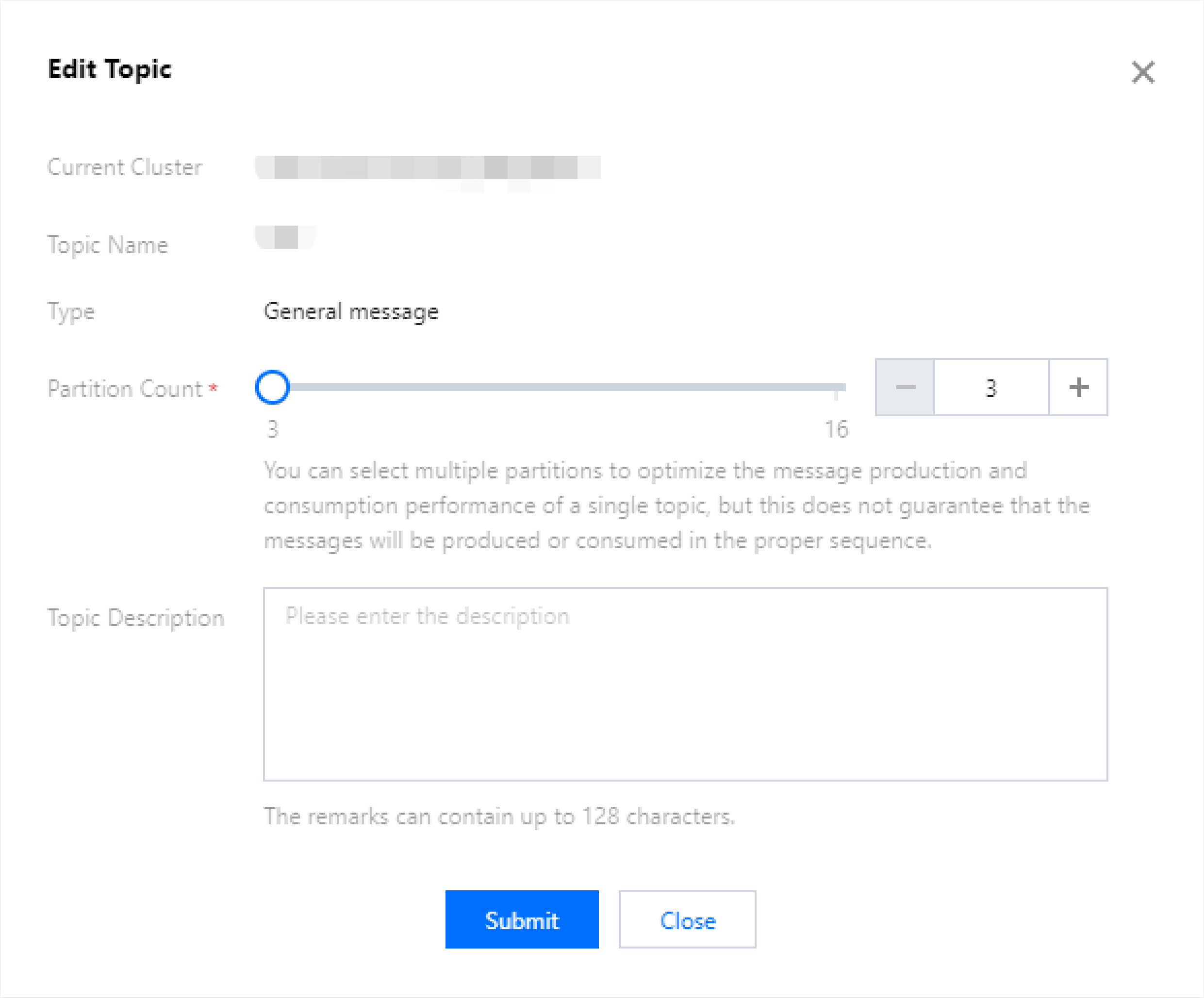
Create (1 / 50)

Search by topic name

□ Topic Name	Monitoring	Type	Queue Quantity	Consumer Group for...	Description	Operation
□ test		General message	3	-	-	Send Test Edit Dele

2. In the pop-up dialog box, you can modify the Topic Description.

3. Click **Submit** to complete the editing of the Topic.



Edit Topic

Current Cluster [redacted]

Topic Name [redacted]

Type General message

Partition Count * 3

3 16

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.

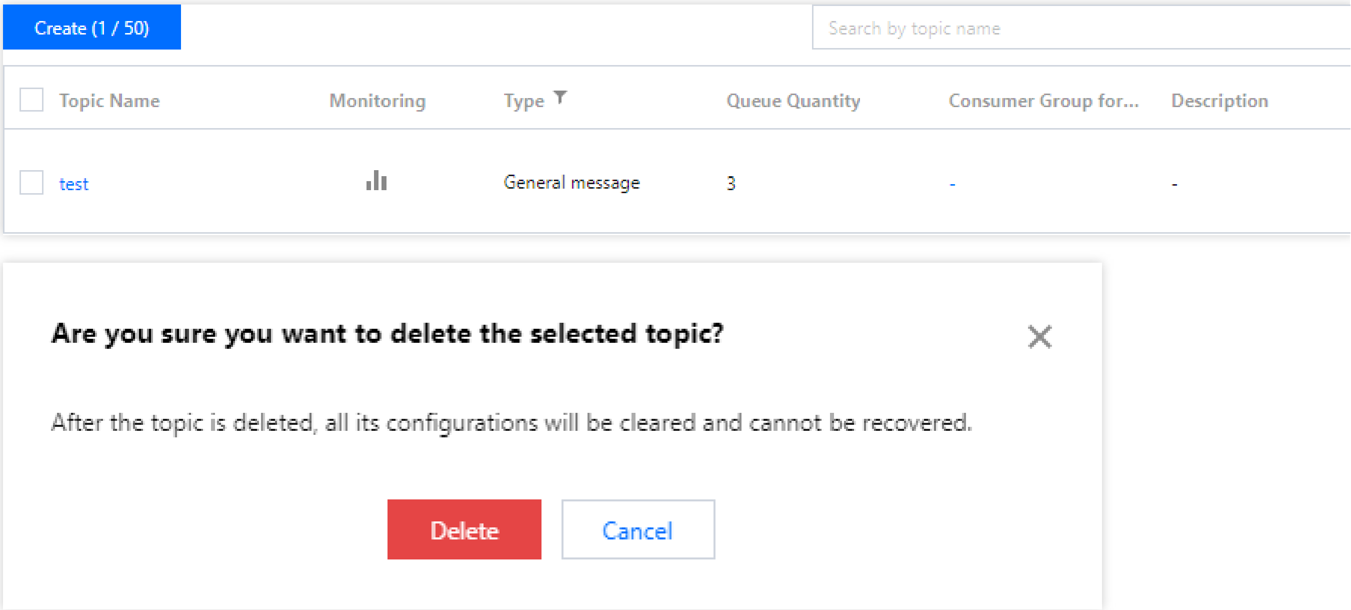
Topic Description: Please enter the description

The remarks can contain up to 128 characters.

Delete a topic

Batch Deletion: In **Topic Management** list, choose all the topics that need to be deleted. Click **Batch Delete** on the top left, and in the prompt box, press **Delete** to conclude the deletion.

Individual Deletion: In **Topic Management** list, locate the Topic that requires removal. Click **Delete** in the action bar, and in the prompt box, press **Delete** to conclude the deletion.



□ Topic Name	Monitoring	Type	Queue Quantity	Consumer Group for...	Description
□ test	📊	General message	3	-	-

Are you sure you want to delete the selected topic?

After the topic is deleted, all its configurations will be cleared and cannot be recovered.

Note:

After the topic is deleted, all its configurations will be cleared and cannot be recovered. Please proceed with caution.

Metadata Import/Export

Metadata Export

You can directly export metadata through the ⤓ button located at the upper-right corner of the topic list page. The exported metadata will be formatted as an .xlsx spreadsheet file.

Metadata Import

If you need to load topic information from one cluster into another, after metadata is exported, click the ⤒ button located in the upper-right corner of the Topic list page to import topic data into the specified namespace.

Group Management

Last updated：2024-01-17 17:48:36

Operation Scenarios

A group is used to identify a category of Consumers. These consumers typically consume the same kind of messages and subscribe to them in a consistent manner.

This document provides instructions on how to create, delete, and query a Group in the TDMQ for RocketMQ console.

Prerequisites

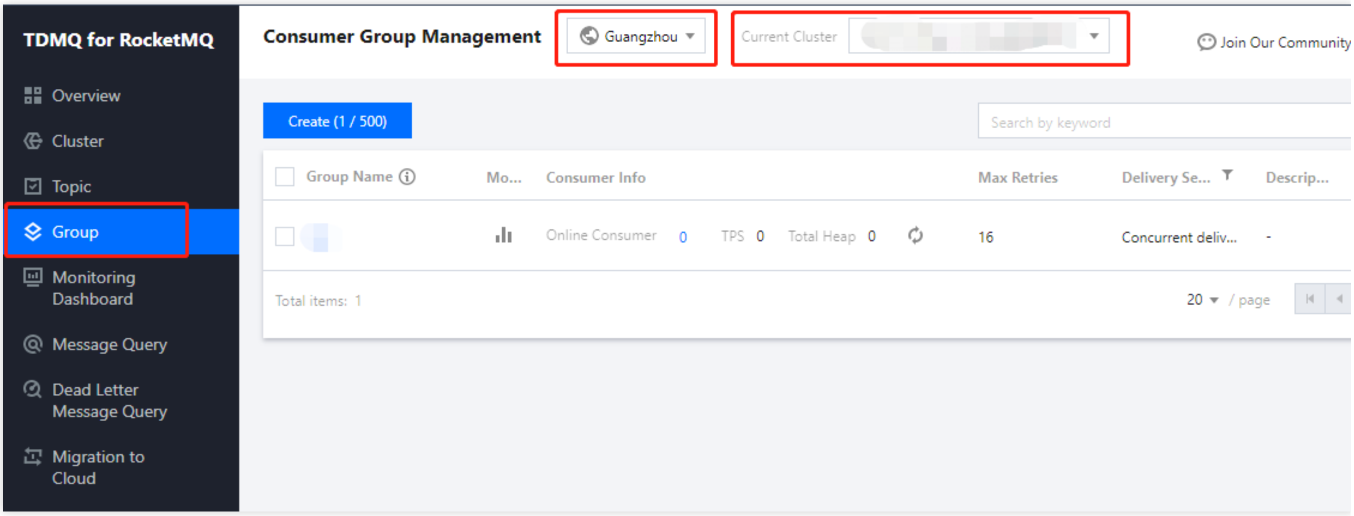
A corresponding namespace must be previously established.

You have created a message producer and consumer using the SDK provided by TDMQ, and they are functioning normally.

Steps

Create a Group

1. Log in to the Console for RocketMQ.
2. Navigate to **Group Management** from the left navigation bar, select the desired region and intended cluster.



Group Name	Mo...	Consumer Info	Max Retries	Delivery Se...	Descrip...
		Online Consumer 0 TPS 0 Total Heap 0	16	Concurrent deliv...	-

Total items: 1

20 / page

3. Click **Create** to access the Group creation page.

4. Provide the relevant Group details.

Group Name: Enter the group name (cannot be edited after creation). It can contain 3 to 64 characters, including letters, numbers, "-" and "_".

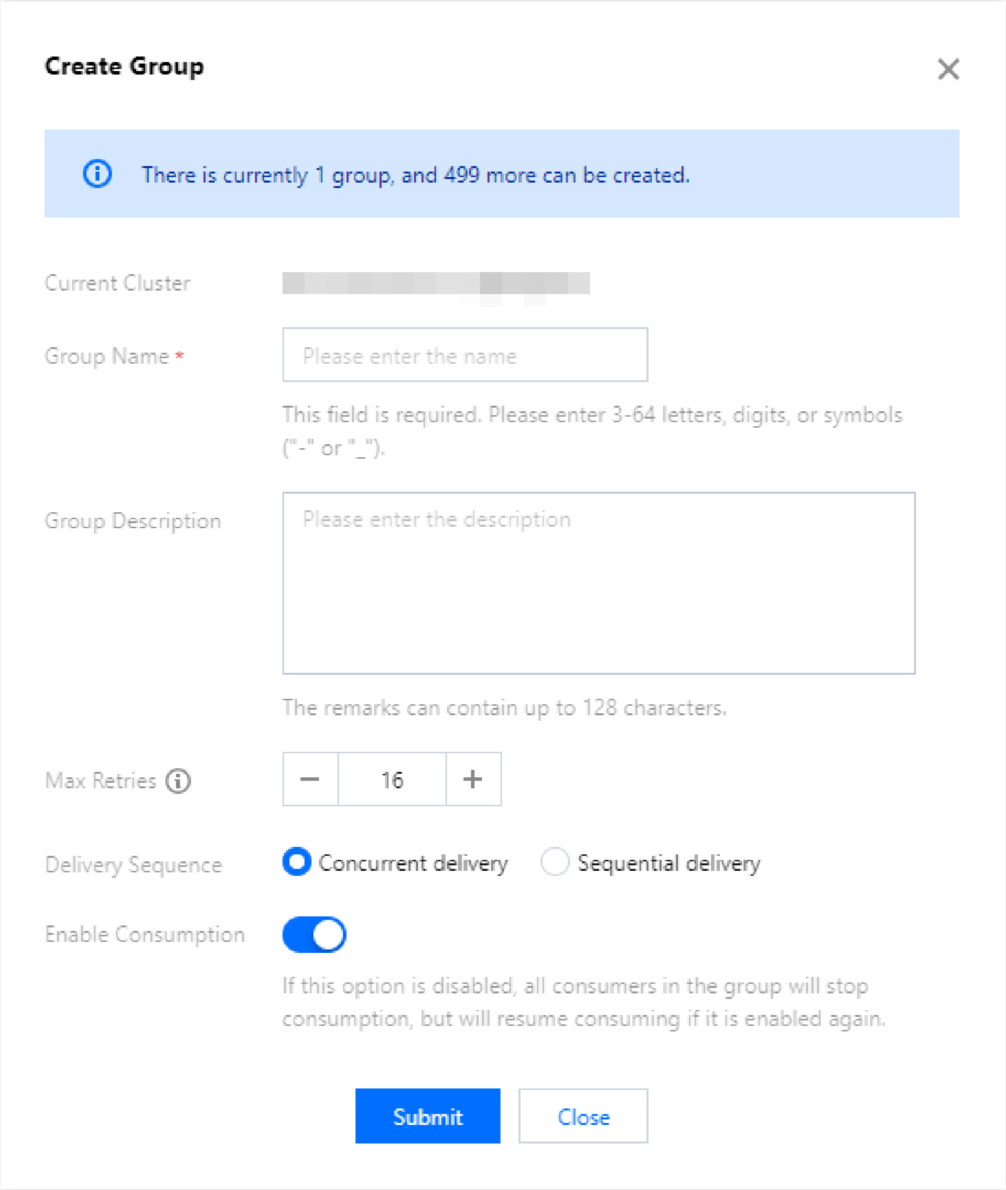
Group Description: Enter a description for the group.

Maximum Retries: Indicates the maximum number of times a message can be redelivered. If a message is still not successfully consumed after exceeding the maximum number of retries, the message will be delivered to the dead letter queue or discarded. If you are using the RocketMQ 4.x client, the number of message retries is determined by the retry count you set in the client. If you are using the RocketMQ 5.x client, then the number of message retries is based on what you have set on the current page.

Delivery Order: The order in which the server delivers messages to consumers. It supports sequential and concurrent delivery. By default, messages are delivered concurrently.

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

5. Click **Submit** to complete Group creation.



Create Group

ⓘ There is currently 1 group, and 499 more can be created.

Current Cluster [redacted]

Group Name * Please enter the name

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

Group Description Please enter the description

The remarks can contain up to 128 characters.

Max Retries ⓘ − 16 +

Delivery Sequence ● Concurrent delivery ○ Sequential delivery

Enable Consumption ●

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

View Consumer Details

1. In **Group** list, click the Group name to enter the client connection list, where you can view the basic information about the Group as well as the client connection list.

Group Name

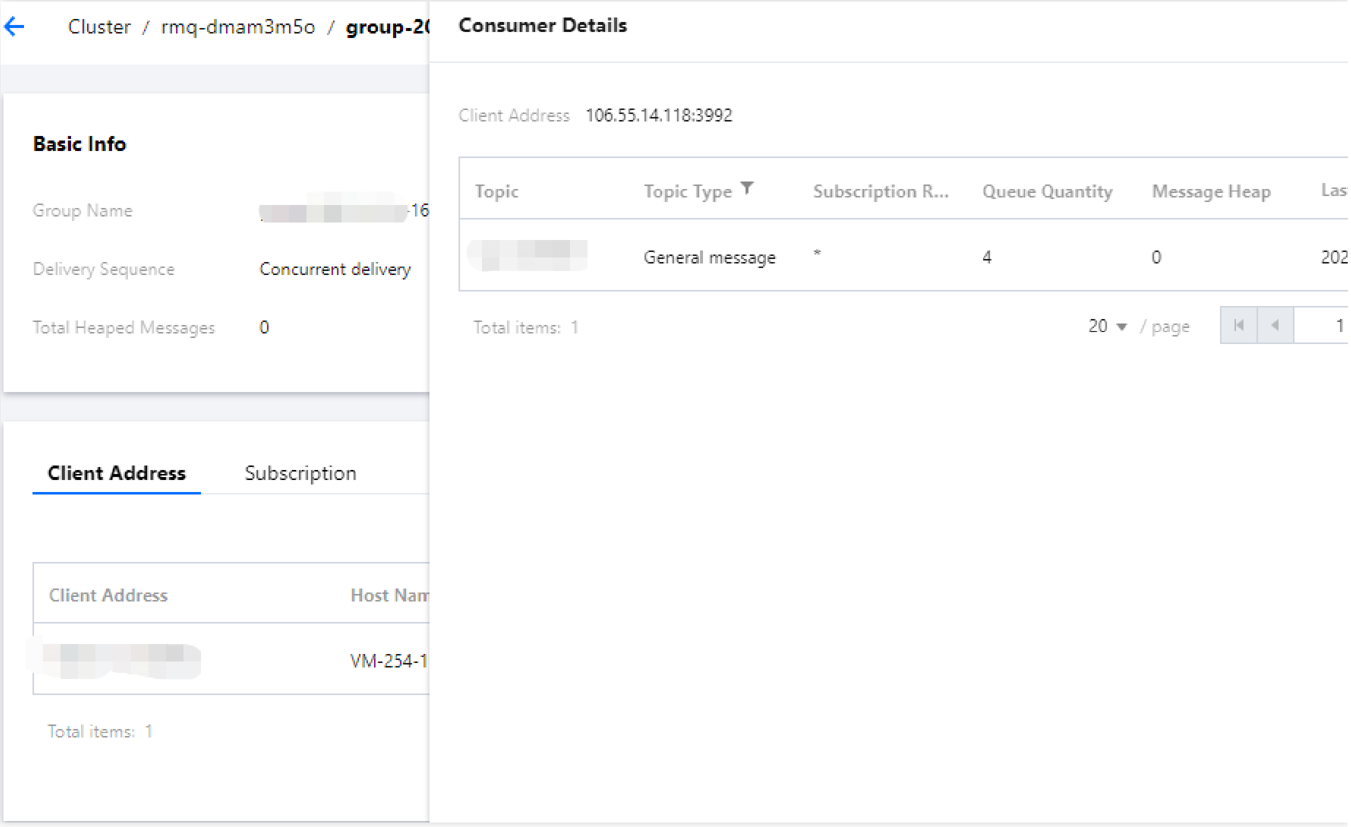
Creation Time

Delivery Orderliness: Sequential delivery or Concurrent delivery

Consumer Type: PUSH or PULL

Total Backlog: The total number of heaped messages.

2. Click **View Details** in the client operation bar to review consumer details.



Basic Info

Group Name	[redacted]-16
Delivery Sequence	Concurrent delivery
Total Heaped Messages	0

Client Address	Host Nam
[redacted]	VM-254-1

Total items: 1

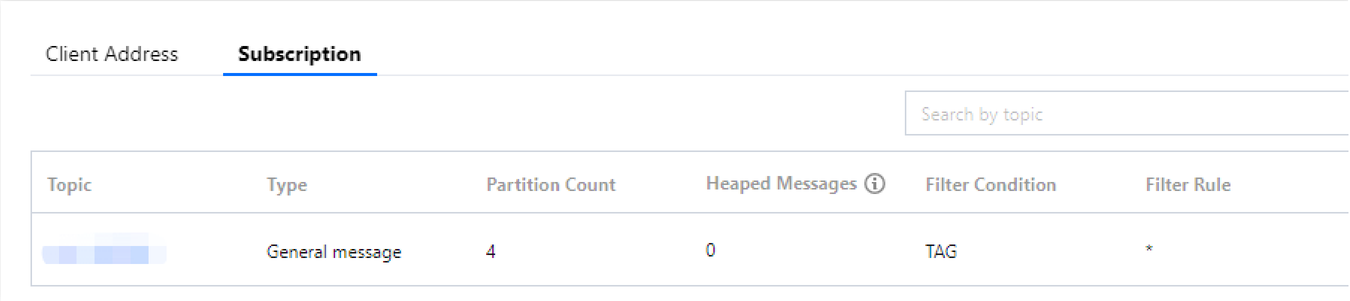
Consumer Details

Client Address 106.55.14.118:3992

Topic	Topic Type	Subscription R...	Queue Quantity	Message Heap	Las
[redacted]	General message	*	4	0	202

Total items: 1

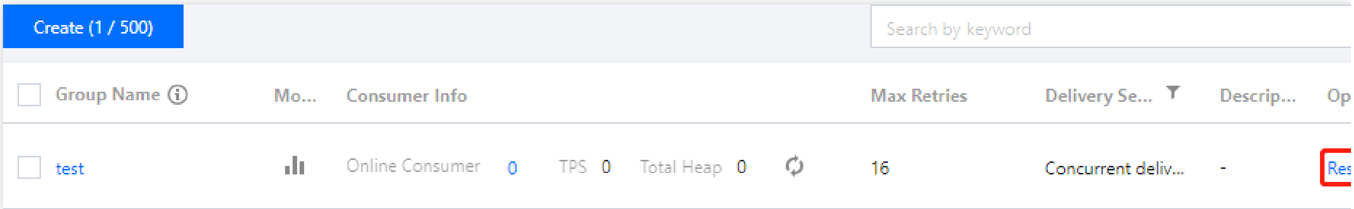
3. Switch to the Subscription tab to view the list of Topics subscribed by the Group and their subscription properties.



Topic	Type	Partition Count	Heaped Messages ⓘ	Filter Condition	Filter Rule
[redacted]	General message	4	0	TAG	*

Set an offset

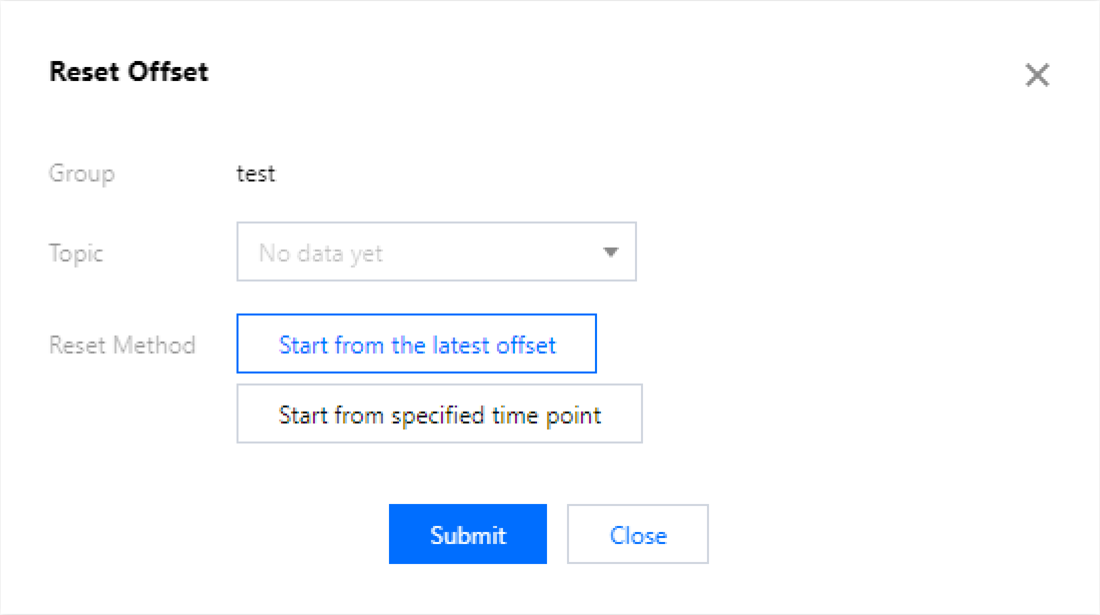
1. In Group list, click **Reset Offset** in the operation column of the intended Group.



	Group Name	Mo...	Consumer Info	Max Retries	Delivery Se...	Descrip...	Op
□	test		Online Consumer 0 TPS 0 Total Heap 0	16	Concurrent deliv...	-	Res

2. In the pop-up window, you can either choose to **Start from the latest offset** or **Start from a specified point** to set the Topic's **Consumer Offset** (Namely, specify where the consumer under this subscription begins to consume messages).

3. Click **Submit** to complete the setting.


Reset Offset

Group test

Topic No data yet

Reset Method Start from the latest offset

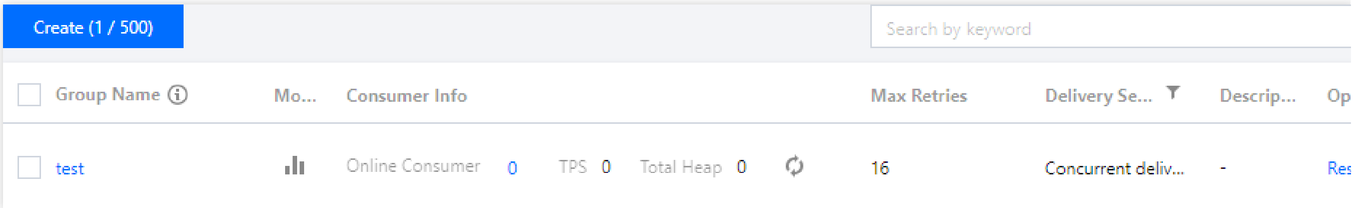
Start from specified time point

Note:

TDMQ-RocketMQ supports resetting the offset (consumption position) for offline Groups, but currently only supports consumer groups under the Push consumption model; otherwise, it might fail to reset.

Edit Group

1. In Group list, click **Edit** in the operations column of the intended Group.

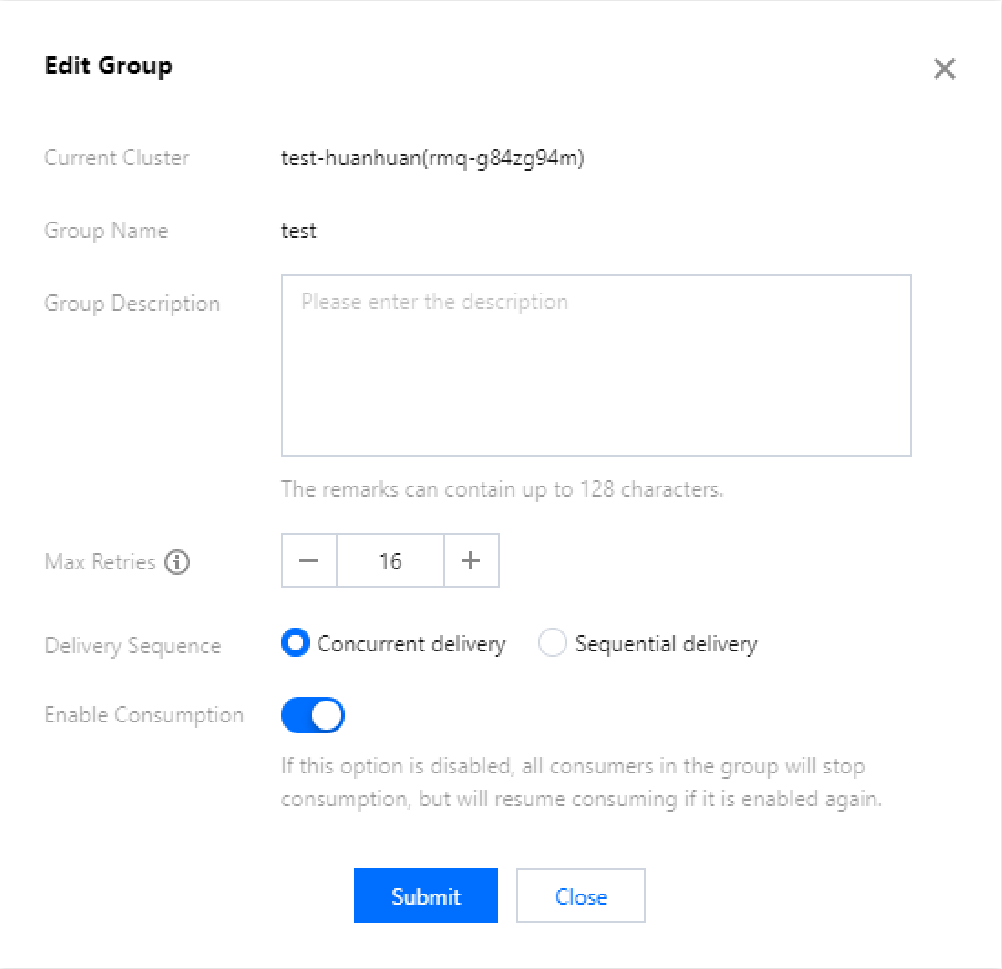


Create (1 / 500)

Search by keyword

□ Group Name	Mo...	Consumer Info	Max Retries	Delivery Se...	Descrip...	Op
□ test		Online Consumer 0 TPS 0 Total Heap 0	16	Concurrent deliv...	-	Res

2. In the pop-up window, edit the Group information.
3. Click **Submit** to complete the modification.



Edit Group

Current Cluster: test-huanhuan(rmq-g84zg94m)

Group Name: test

Group Description: Please enter the description

The remarks can contain up to 128 characters.

Max Retries: 16

Delivery Sequence: ● Concurrent delivery ○ Sequential delivery

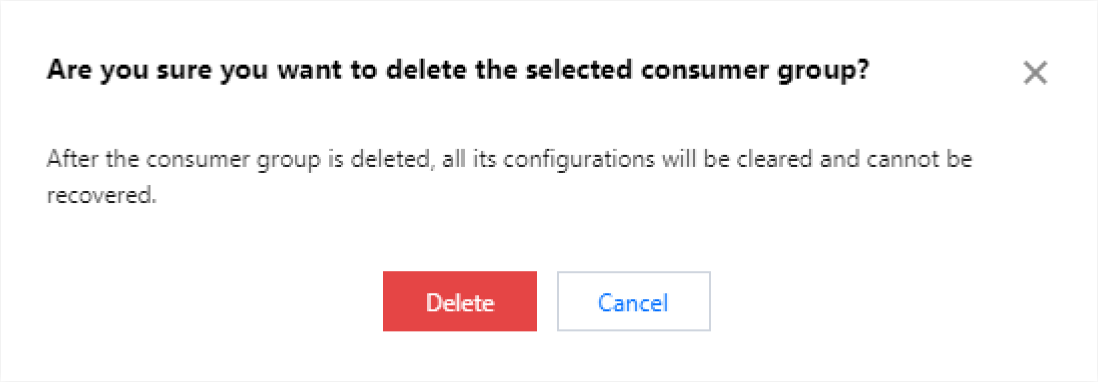
Enable Consumption

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

Delete Group

Batch Deletion: In Group list, select all the Groups that need to be deleted, click **Batch Delete** at the top-left corner. In the prompt box, click **Delete** to conclude the deletion.

Single Deletion: In Group list, locate the Group that requires to delete, click **Delete** in the Operations column. In the prompt box, click **Delete** to conclude the deletion.



Are you sure you want to delete the selected consumer group?

After the consumer group is deleted, all its configurations will be cleared and cannot be recovered.

Note:

After the Group is deleted, consumers identified by this Group will cease to receive messages immediately. All configurations under this Group will be erased and are irretrievable. Please proceed with caution.

Metadata Import/Export

Export Metadata

You can directly export metadata through the

⤓ button located at the upper-right corner of the Group list page. The exported metadata will be formatted as an .xlsx spreadsheet file.

Import Metadata

If you need to load Group information from one cluster into another, after exporting metadata, click the

⤒ button located in the upper-right corner of the Group list page to import Group data into the specified namespace.

Monitoring Alarm

Last updated：2024-01-17 17:48:49

Operation Scenarios

TDMQ for RocketMQ allows you to monitor resources created under your account, including clusters, topics, and groups. Based on these metrics, you can analyze the cluster usage and promptly address any possible risks. Moreover, you can set alarm rules for the monitoring metrics to receive notifications in case of abnormal data, allowing you to manage risks promptly and ensure stable system performance.

Monitoring Metrics

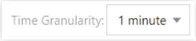
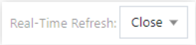
The monitoring metrics supported by TDMQ for RocketMQ are as follows:

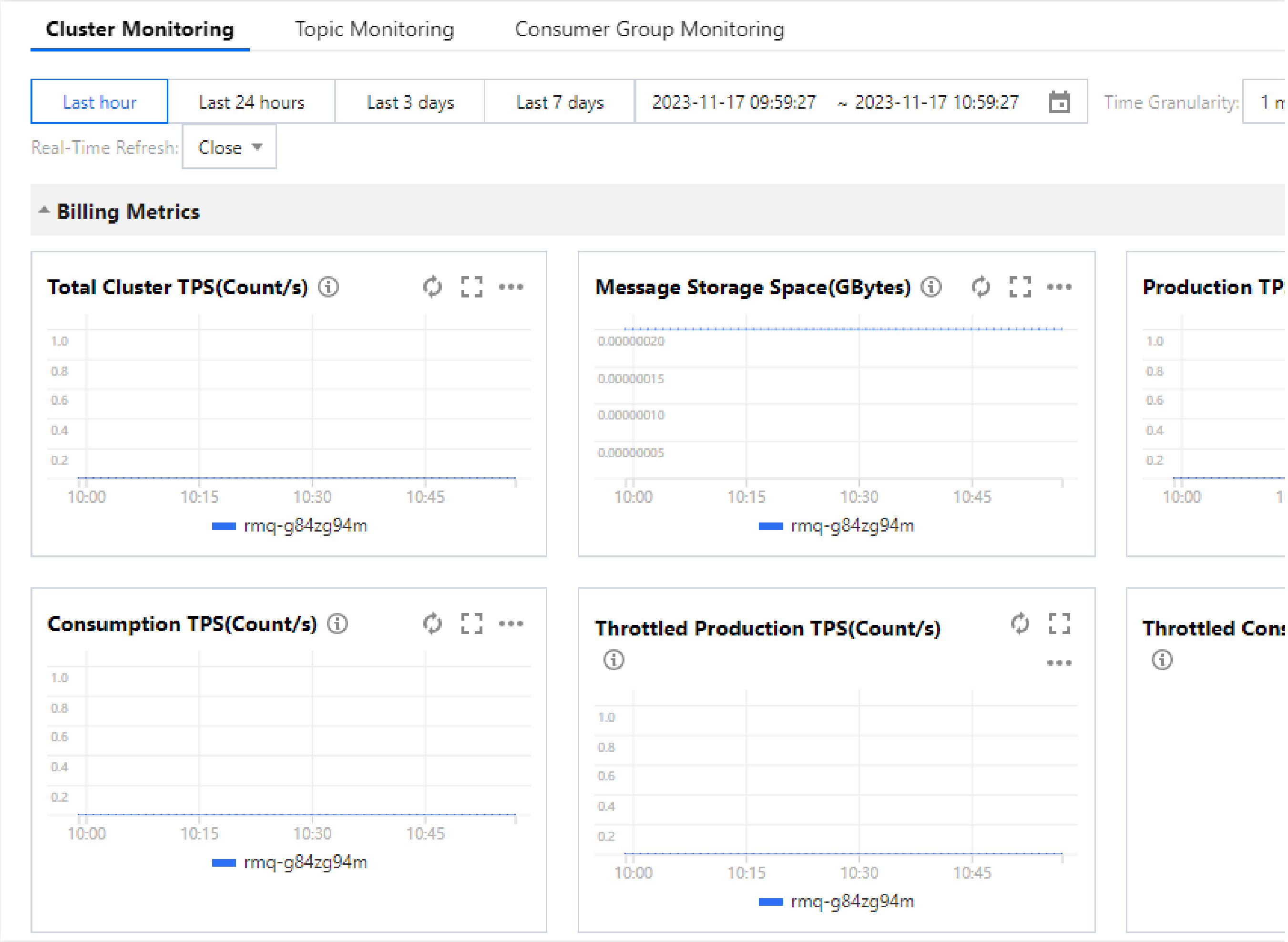
Classification	Unit	Metric
Cluster	Count	Number of Heaped Messages
	Count/s	Billing API Production Consumption Count
	Count/s	Billing API Consumption Throttling Count
	Count/s	Billing API Consumption Count
	Count/s	Billing API Production Throttling Count
	Count/s	Billing API Production Count
Topic	Bytes/s	Message Production Byte TPS
	Count/s	Message Production Count TPS
	Count	Number of Heaped Messages
	Count/s	Billing API Consumption Throttling Count
	Count/s	Billing API Consumption Count
	Count/s	Billing API Production Throttling Count
	Count/s	Billing API Production Count
	Count/s	Number of Messages Consumed

	Bytes/s	Bytes of Messages Consumed
	Bytes	Topic Message Storage Size
Group	Count	Number of Consumers
	Count	Number of Heaped Messages
	Count/s	Billing API Consumption Throttling Count
	Count/s	Billing API Consumption Count
	Count/s	Number of Messages Consumed
	Bytes/s	Bytes of Messages Consumed

View Monitoring Data

1. Log in to the RocketMQ console.
2. On the left sidebar, click **Monitoring Dashboard** and select the region and cluster to be viewed.
3. On the monitoring page, select the desired resource tab and define the time range to access the corresponding monitoring data.

Icon	Description
 Time Granularity: 1 minute	Click to adjust time granularity of the chart. 1 minute, 5 minutes, and 1 hour are supported.
 Real-Time Refresh: Close	Click to fetch the latest monitoring data, it supports setting 30 seconds, 1 minute, and 5 minutes as automatic refresh intervals for the monitoring data.
•••	Click to copy the chart to the dashboard. For more information about the dashboard, please refer to What is Dashboard.

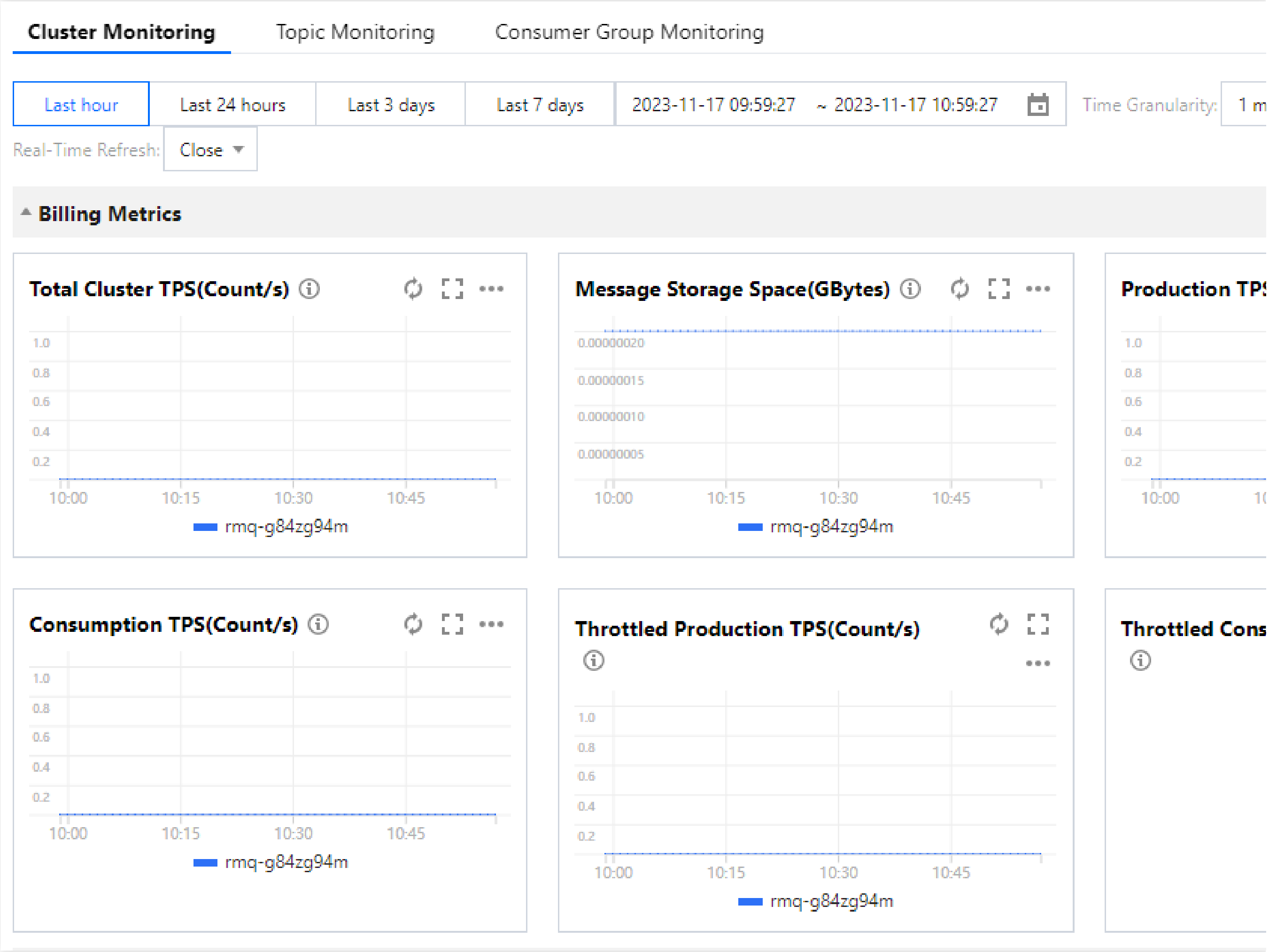


Configure Alarm Rules

Create Alarm Rules

You can configure alarm rules for monitoring metrics. In the event that a monitoring metric hits the pre-set alarm threshold, TCOP can timely notify you of any exceptional circumstances through various mediums such as email, SMS, WeChat, and telephone.

1. On **Monitoring** page of the cluster, click the alarm icon as shown below to redirect to the TCOP Console for configuring an alarm policy.



2. On the Alarm Policy page, choose the policy type and instance for alarm setting, establish the alarm rule, and set up the alarm notification template.

Policy Type: Select **TDMQ/RocketMQ5 Cluster**.

Alarm Object: Select the **RocketMQ** instance 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, please refer to Creating trigger condition template.

Note:

Metric: For instance, if you select 1 minute as the statistical granularity for the "message production TPS" metric, then if the message production TPS exceeds the threshold for N consecutive data points in that minute, an alarm will be triggered.

Alarm Frequency: For instance, "Alarm once every 30 minutes" implies that if a metric surpasses the threshold during multiple consecutive statistical granularity in 30 minutes, then a single alarm will be activated. Within these 30 minutes, no further alarms will be triggered until the next 30-minute interval. If the metric consistently exceeds the threshold during this subsequent interval, another alarm will be dispatched.

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

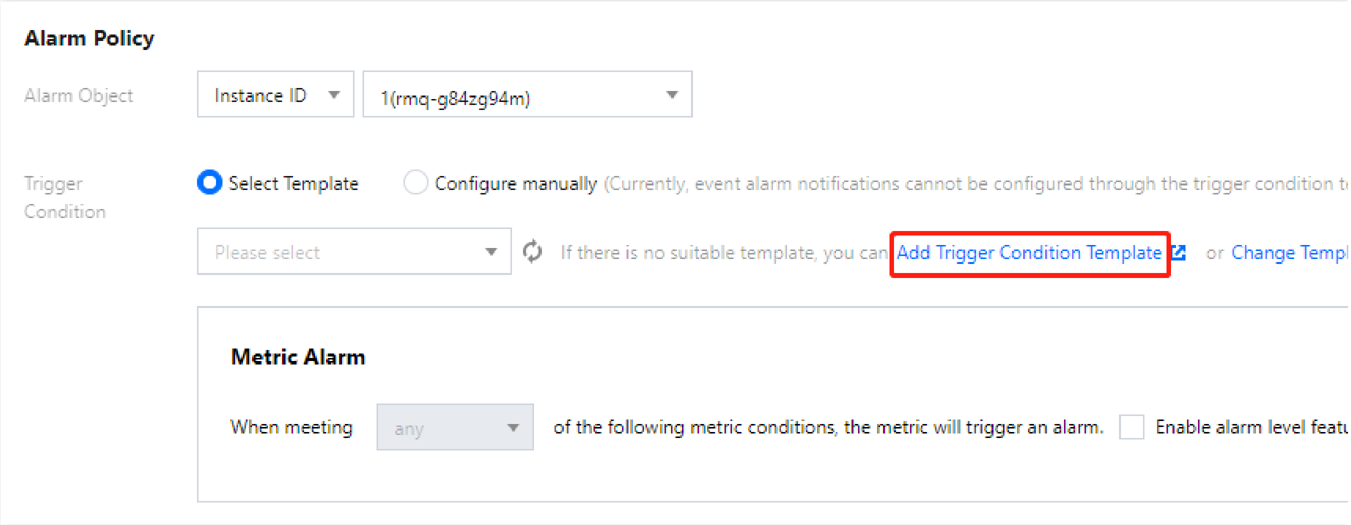
3. Click **Complete** to complete the configuration.

Note:

For more details on alarms, please refer to TCOP Alarm Service.

Create a new trigger condition template

1. Log in to the TCOP console.
2. In **Configure Alarm Rules**, click **Select Template**> **Add Trigger Condition Template** to access the trigger condition list page.



Alarm Policy

Alarm Object: Instance ID, 1(rmq-g84zg94m)

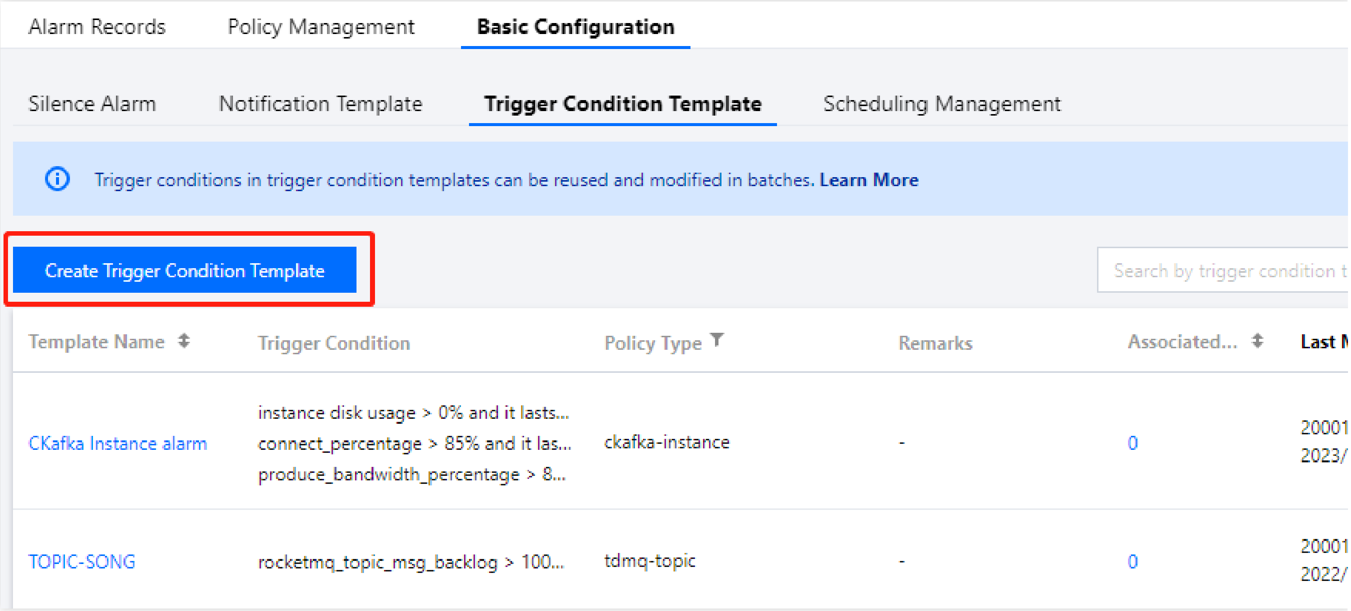
Trigger Condition: ● Select Template ○ Configure manually (Currently, event alarm notifications cannot be configured through the trigger condition t

Please select — If there is no suitable template, you can Add Trigger Condition Template or Change Templ

Metric Alarm

When meeting any of the following metric conditions, the metric will trigger an alarm. □ Enable alarm level featu

3. Click **Create Trigger Condition Template** on the trigger condition template page.



Trigger conditions in trigger condition templates can be reused and modified in batches. Learn More

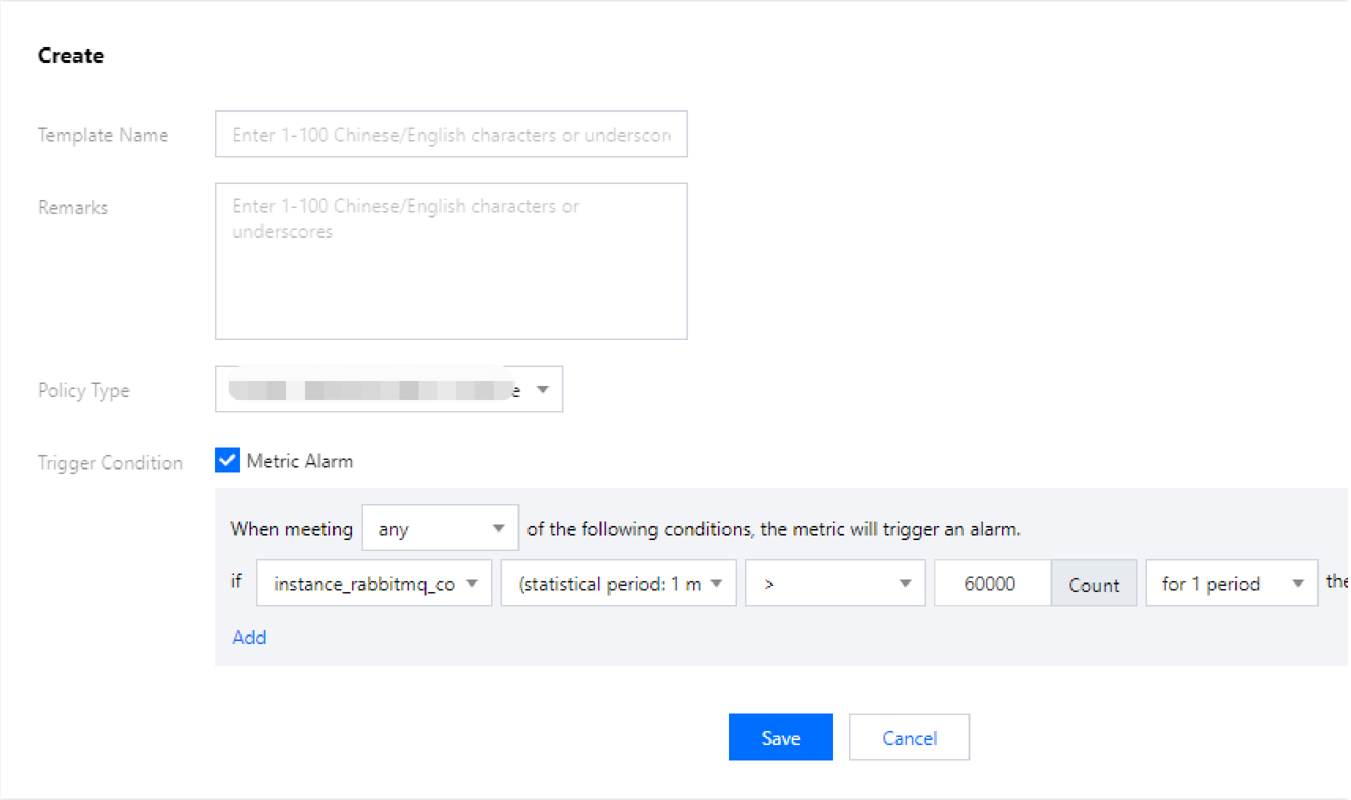
Create Trigger Condition Template
Search by trigger condition t

Template Name	Trigger Condition	Policy Type	Remarks	Associated...	Last M
CKafka Instance alarm	instance disk usage > 0% and it lasts... connect_percentage > 85% and it las... produce_bandwidth_percentage > 8...	ckafka-instance	-	0	20001 2023/
TOPIC-SONG	rocketmq_topic_msg_backlog > 100...	tdmq-topic	-	0	20001 2022/

4. On the template creation page, configure the strategy type.

Policy Type: Select **TDMQ/RocketMQ5**.

Triggers Condition: By selecting this option, the suggested alarm policies from system will occur.



Template Name: Enter 1-100 Chinese/English characters or underscor

Remarks: Enter 1-100 Chinese/English characters or underscores

Policy Type:

Trigger Condition: ☑ Metric Alarm

When meeting any of the following conditions, the metric will trigger an alarm.

if instance_rabbitmq_co (statistical period: 1 m > 60000 Count for 1 period the

5. After ensuring all details are correct, click **Save**.

6. Navigate back to the alarm policy creation page and click **Refresh**. The newly configured alarm policy template will occur.

Message Query

Query General Message

Last updated：2024-05-14 17:32:36

When sending a message from its origin at the producer to the TDMQ for RocketMQ server, and finally to the consumers, TDMQ for RocketMQ records the flow progress of the message and displays as a message trace in the console.

The message trace records the full course of the message, from its origin at the producer to the TDMQ for RocketMQ server, and finally to the consumers. This includes details about each phase, such as time duration (accurate to the microsecond), execution results, producer IP, and consumer IP.

If you use a client version 5.0 or later one for producing and consuming messages, there is no need to individually activate the trace switch on the client.

If you use a client v4.x, you need to enable the message trace feature via the client. An example of specific setting is as follows:

Producer settings

Push consumer settings

Pull consumer settings

Spring Boot Starter Integration (version 2.2.2 and above)

```
DefaultMQProducer producer = new DefaultMQProducer(namespace, groupName,
    // ACL permissions
    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);
```

```
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);
        }
    }

}
```

Operation Scenarios

When you need to investigate the following issues, you can utilize the message query features in the TDMQ for RocketMQ console. You can search for specific messages by time dimension, or by message ID or message key found in the logs, in order to view the message content, parameters, and trace.

Observe the specific content and parameters of a given message.

Examine the producer IP from which a message was dispatched, verify its successful transmission, and determine the precise timestamp of its arrival at the server.

Inspect whether the message has been persistently stored.

Investigate which consumers have ingested the message, ascertain the success of the consumption, and determine the exact moment at which the consumption was acknowledged.

Conduct a performance analysis of the distributed system by scrutinizing the message processing latency of the message queue.

Steps

1. Log in to the RocketMQ console and click on **Message Query** in the left sidebar.
2. On the message query page, after selecting the region, follow the instructions to input the query conditions.

Time Range: Select the desired query time range, supporting the options of the most recent 100 messages (default choice displays the latest 100 messages in chronological order), the last 30 minutes, last 1 hour, last 6 hours, last 24 hours, last 3 days, or a customizable time range.

Cluster: Select the cluster where the Topic you intend to query is located.

Topic: Select the desired Topic.

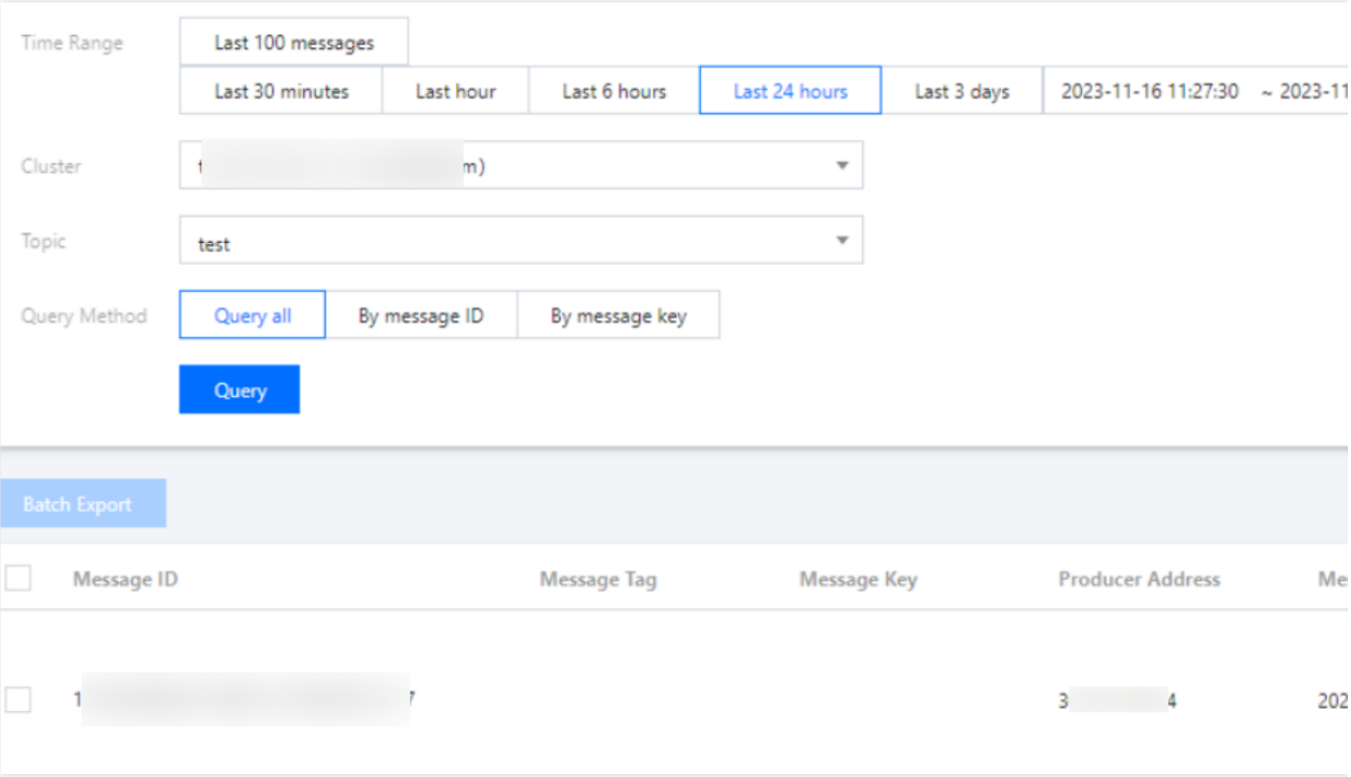
Query Method: The message query feature supports the following query methods.

Query all: This method is suitable for querying all messages.

By message ID: This is a precise and high-speed query method with an exact match.

By message key: This method is a fuzzy query. It is suitable for a scenario where you have set a message key with no recorded message ID.

3. Click **Query**. The list below will show all the results of the search, displayed by pages.



Time Range: Last 100 messages | Last 30 minutes | Last hour | Last 6 hours | Last 24 hours | Last 3 days | 2023-11-16 11:27:30 ~ 2023-11

Cluster: t[redacted]m)

Topic: test

Query Method: Query all | By message ID | By message key

Query

Batch Export

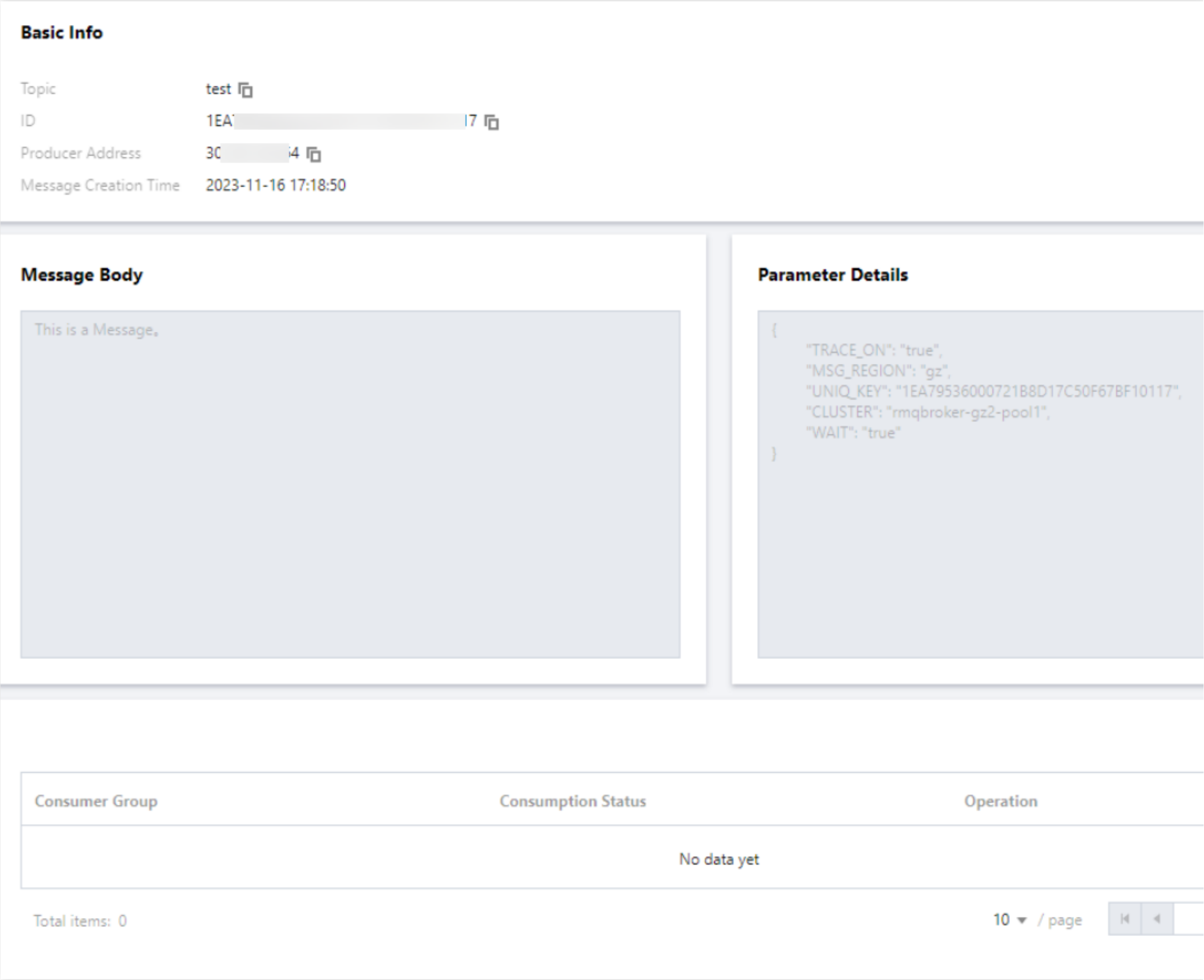
□	Message ID	Message Tag	Message Key	Producer Address	Me
□	1[redacted]7			3[redacted]4	202

4. Locate contents or parameters of the message you require to view. Click **View Details** in the operation column to access the basic information, content (message body), detailed parameters, and consumption status of the message. In the consumption status section, you can view the Group that consumed the message and the consumption status.

In the operation column, you can also perform the following actions:

Resend: This action sends the message to a specific client again. If the message has already been successfully consumed, resending it may lead to duplicated consumption.

Exception Diagnosis: If consumption is abnormal, you can access the exception diagnosis information.



Basic Info

Topic	test
ID	1EA[redacted]7
Producer Address	30[redacted]4
Message Creation Time	2023-11-16 17:18:50

Message Body

This is a Message.

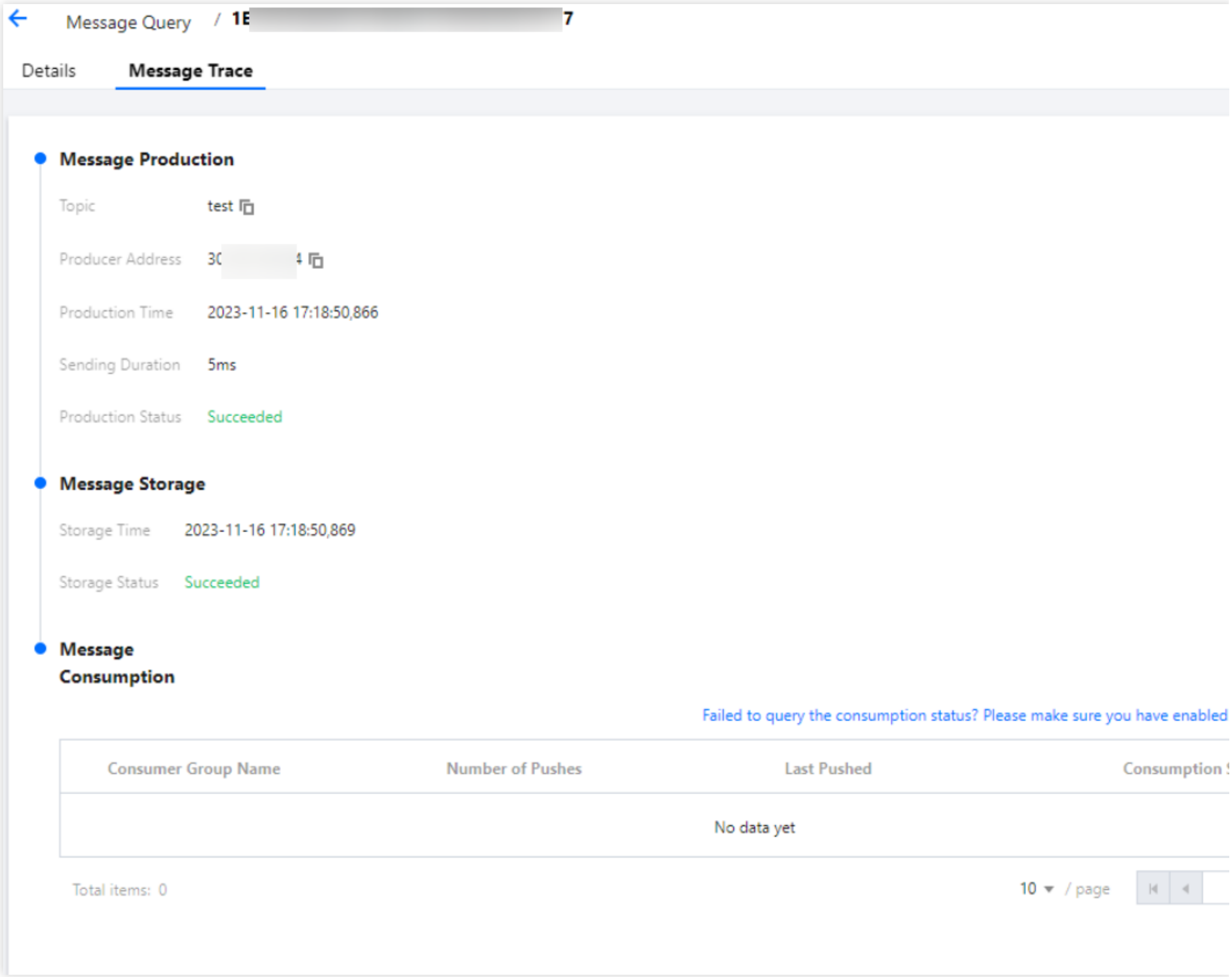
Parameter Details

```
{
    "TRACE_ON": "true",
    "MSG_REGION": "gz",
    "UNIQ_KEY": "1EA79536000721B8D17C50F67BF10117",
    "CLUSTER": "rmqbroker-gz2-pool1",
    "WAIT": "true"
}
```

Consumer Group	Consumption Status	Operation
	No data yet	

Total items: 0

5. Click **View Message Trace** in the operation column, or click **Message Trace** in the details page. You can view the message trace of the message (For detailed information, please refer to Overview of Message Trace Query Results).



Message Query / [redacted]

Details Message Trace

Message Production

Topic test

Producer Address [redacted]

Production Time 2023-11-16 17:18:50,866

Sending Duration 5ms

Production Status Succeeded

Message Storage

Storage Time 2023-11-16 17:18:50,869

Storage Status Succeeded

Message Consumption

Failed to query the consumption status? Please make sure you have enabled

Consumer Group Name	Number of Pushes	Last Pushed	Consumption
No data yet			

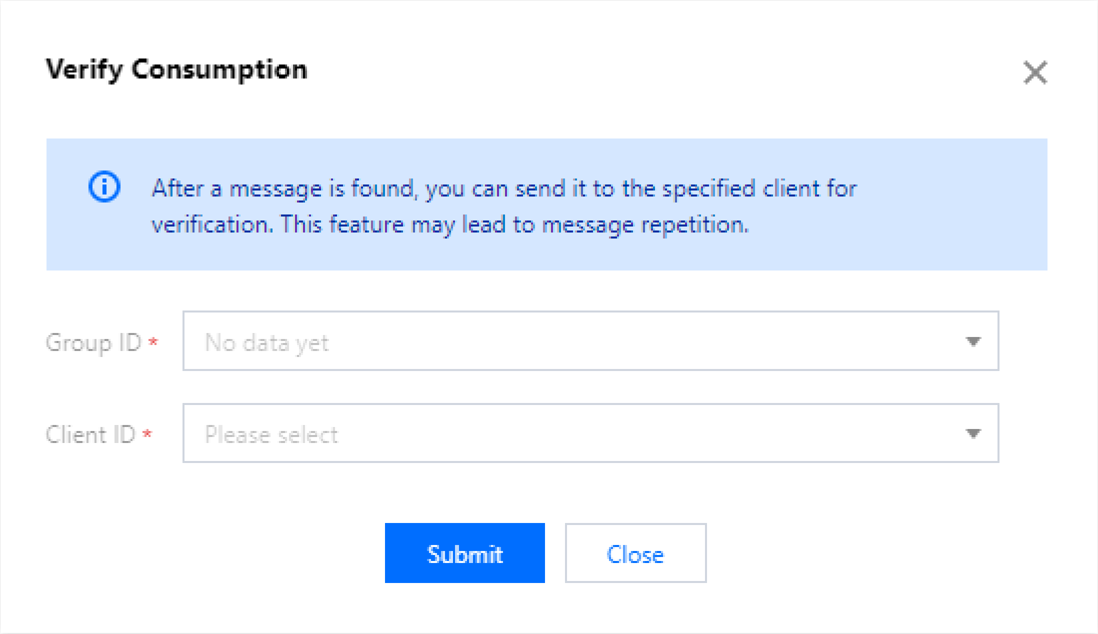
Total items: 0 10 / page

Verify consumption

After querying a message, you can click **Verify Consumption** in the operation column to send the message to a specified client for verification. **Using this feature may lead to message duplication.**

Note:

The consumption verification feature is exclusively use to verify the normalcy of the client's consumption logic with no influence on the standard message receiving process. Consequently, information such as the status of message consumption remains unaltered post-verification.



Verify Consumption

After a message is found, you can send it to the specified client for verification. This feature may lead to message repetition.

Group ID * No data yet

Client ID * Please select

Export messages

After querying a message, you can click **Export Message** in the operation column to export details such as the message body, message Tag, message Key, message production time and consumption attributes.

Querying Retry Messages

Last updated：2024-05-14 17:37:50

To support business operations in case of failure and ensure a full lifecycle for the message in the event of message consumption failure, RocketMQ has implemented a policy to retry message consumption upon failure.

If you are using the RocketMQ 4.x client, the number of message retries is based on the settings you specify within the client.

For the RocketMQ 5.x clusters, you can set the number of message retries when you are creating a Group. If you are using a 5.x client, then the number of retries is based on the settings on the server side; if you are using a 4.x client, then the number of retries is still based on the settings within the client.

Overview

When you need to check if there are any retry messages under a specific Topic, you can query messages on the Retry Message Query Page and expand to view details of each retry attempt, including the time and producer address. It also supports exporting the message and viewing its detailed contents, as shown below.

Directions

1. Log in to the RocketMQ console and click Retry Message Query Page on the left sidebar.
2. On the message query page, select the region and then enter the query criteria as prompted on the page.

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.

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

Topic: Select the topic you want to query.

Group: If the cluster you are querying is a 5.x cluster, you need to select the specific group subscribed under that topic. No need to fill in for 4.x clusters.

Query Method: The message query feature supports the following methods.

Query All: This method is suitable for situations where the information about retry messages is unclear, and it is used to query all retry messages under the current topic.

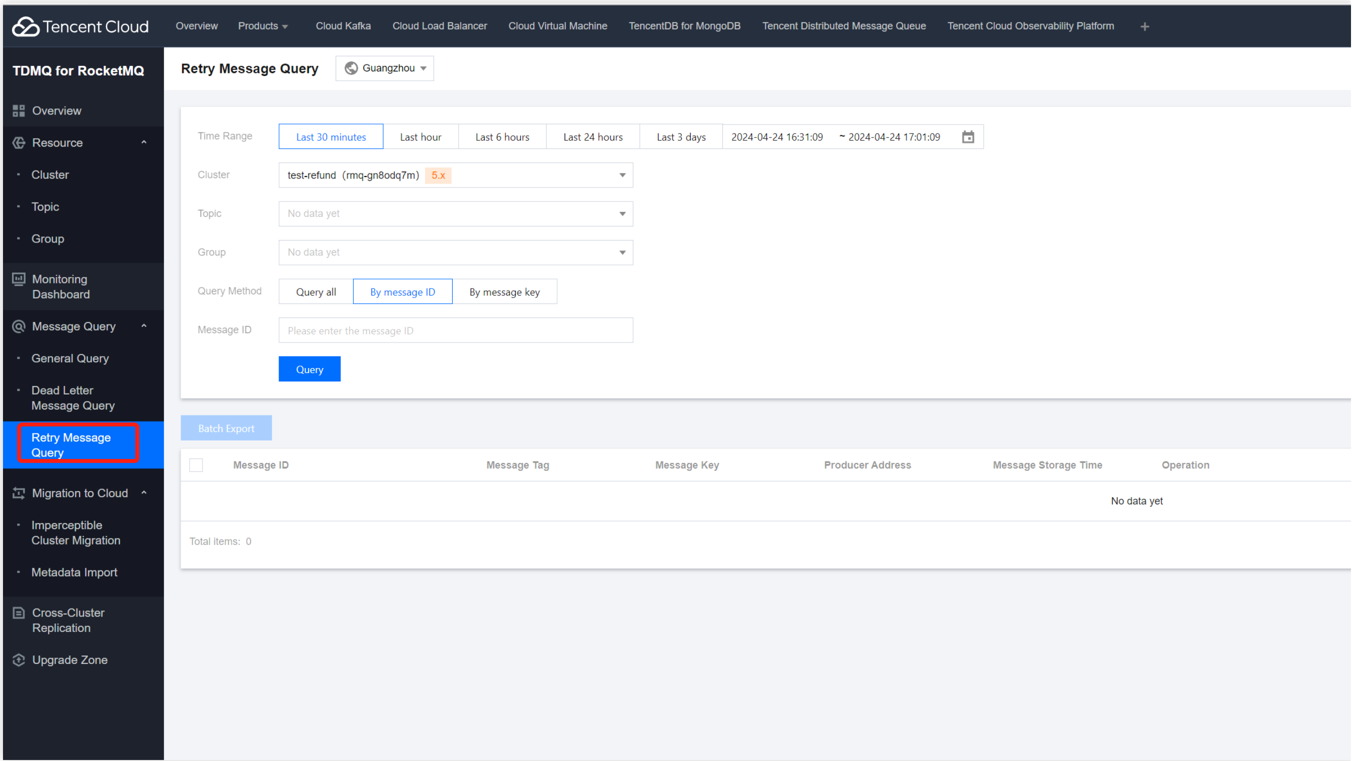
Query by Message ID: This method is an exact query, and it provides a fast and exact match.

Query by Message Key: This method is a fuzzy query, and it is applicable when you do not have a record of the message ID but have set the message key.

Note:

To ensure query speed, when you select Query All, the server will query the most recent messages in chronological order. However, due to query time and display limitations, it may not quickly locate the message you need to query. It is recommended to use more specific search criteria, such as Message ID and Message Key.

3. click **Query**, and the results will be displayed in the list below with pagination.



Time Range: Last 30 minutes, 2024-04-24 16:31:09 ~ 2024-04-24 17:01:09

Cluster: test-refund (rmq-gn8odq7m) 5.x

Topic: No data yet

Group: No data yet

Query Method: By message ID

Message ID: Please enter the message ID

Message ID	Message Tag	Message Key	Producer Address	Message Storage Time	Operation
No data yet					

Total items: 0

4. After the query is completed, you can click a single message to view the retry situation of the current message, such as the number of retries and the producer address, etc. You can also click other options in the operation bar.

Querying Dead Letter Message

Last updated：2024-01-17 16:42:03

Overview

A dead letter queue is a special form of message queue, used to process messages that are unable to be normally consumed in a centralized manner. If a message fails to be properly consumed after a certain number of retry attempts, TDMQ for RocketMQ will determine that this message cannot be consumed in the current situation and send it to the dead letter queue.

In actual situations, messages may become unconsumable due to extended periods of service downtime or network disconnections. In such situations, messages are not discarded immediately, but the dead letter queue engages in extended persistence processing for such messages. Upon identifying an appropriate solution, users can create a consumer subscription to the dead letter queue in order to process messages that cannot be process at the time.

Query Restrictions

You can query messages from the past three days at most.

Feature Description

If a message is sent to the dead letter queue, it will not be consumed by consumers normally. You can query messages from the past three days at most. Please process dead letter messages within three days after they are generated. Otherwise, these messages may be deleted.

A dead letter queue contains all the dead letter messages generated in every topic within a single group. If there are no dead letter messages in a group, no dead letter queues will be created and dead letter messages cannot be found.

Directions:

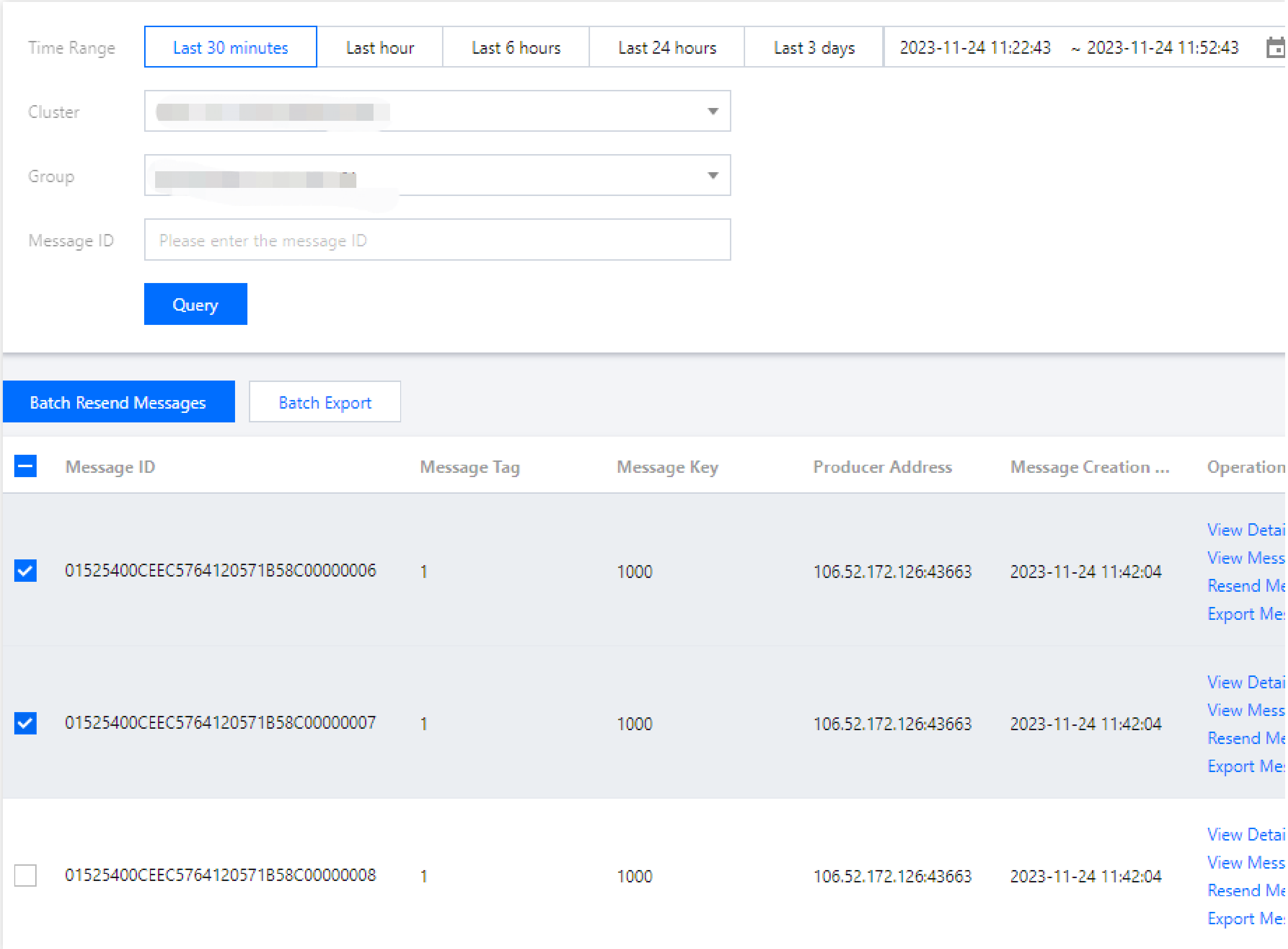
1. Log in to the TDMQ for RocketMQ console, and then click on **Dead Letter Message Query** in the left sidebar.
2. On the message query page, select the region and follow the on-screen instructions to enter the query conditions.

Time Range: Choose your desired time period for conducting the query. You can choose the last 30 minutes, last hour, last 6 hours, last 24 hours, last 3 days, or a customized time range.

Present Cluster: Choose the cluster that contains the dead letter message to be queried.

Group: Choose the group that contains the dead letter message to be queried.

Message ID: It is optional.
If you leave the Message ID field blank, the query is a **fuzzy search**. Queries can be conducted in batch according to Group ID and time range in which the dead letter messages are generated.
If you specify Message ID, the query is an **exact search**. Group ID and Message ID will be used to precisely locate any message.
3. Click **Query**. The list below will show all the results of the search in pages.



Time Range: Last 30 minutes | Last hour | Last 6 hours | Last 24 hours | Last 3 days | 2023-11-24 11:22:43 ~ 2023-11-24 11:52:43

Cluster: [redacted]

Group: [redacted]

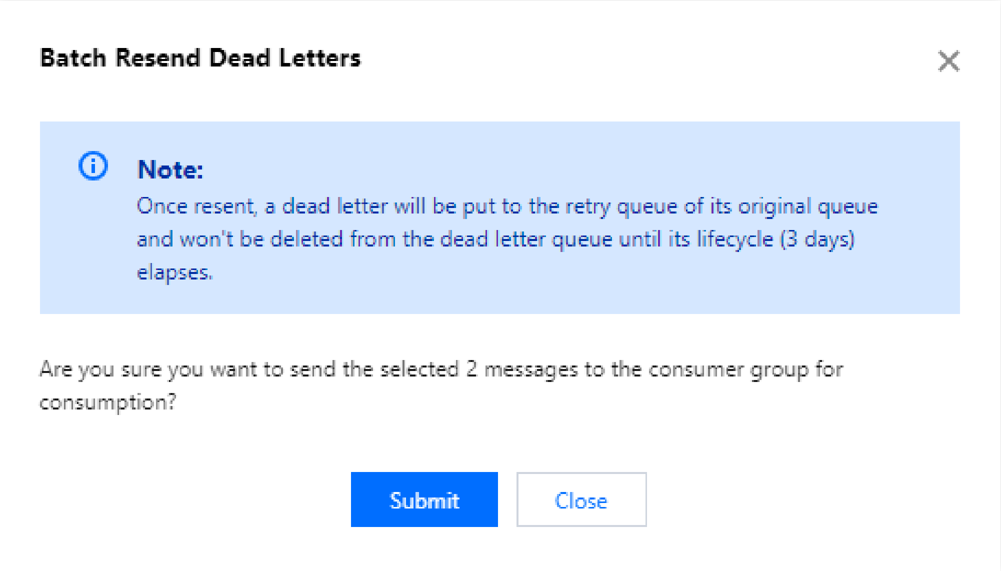
Message ID: Please enter the message ID

Query

Batch Resend Messages | Batch Export

	Message ID	Message Tag	Message Key	Producer Address	Message Creation ...	Operation
☑	01525400CEEC5764120571B58C00000006	1	1000	106.52.172.126:43663	2023-11-24 11:42:04	View Detai View Mess Resend Me Export Me
☑	01525400CEEC5764120571B58C00000007	1	1000	106.52.172.126:43663	2023-11-24 11:42:04	View Detai View Mess Resend Me Export Me
□	01525400CEEC5764120571B58C00000008	1	1000	106.52.172.126:43663	2023-11-24 11:42:04	View Detai View Mess Resend Me Export Me

4. You can select multiple dead letter messages and click **Batch Resend Messages** at the top left corner to resend these messages to the original retry queue. Alternatively, you can click **Resend Message** under the operations column of a single message to resend a particular dead letter message. Messages that have been resent will be delivered to the retry queue of the original queue and will not be immediately deleted from the dead letter queue. These messages will only be deleted after their lifecycle (three days) expires.

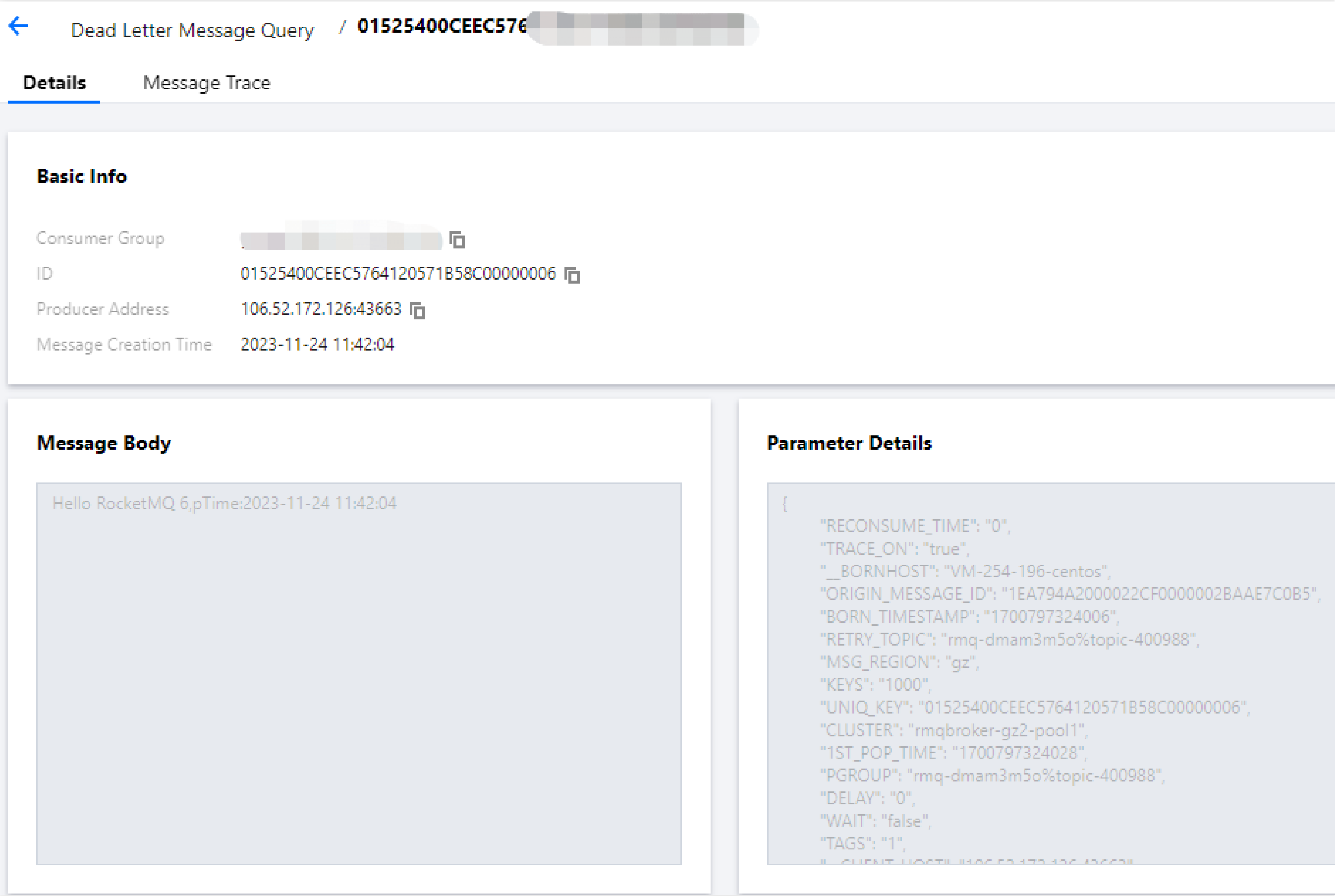


Batch Resend Dead Letters

Note:
Once resent, a dead letter will be put to the retry queue of its original queue and won't be deleted from the dead letter queue until its lifecycle (3 days) elapses.

Are you sure you want to send the selected 2 messages to the consumer group for consumption?

5. Find the message in which you want to view its content or parameters, and click **View Details** under the operation column. Then you can view the basic information, content (message body), and parameters of the message.



/ 01525400CEEC576[redacted]

Basic Info

Consumer Group [redacted]
ID 01525400CEEC5764120571B58C00000006
Producer Address 106.52.172.126:43663
Message Creation Time 2023-11-24 11:42:04

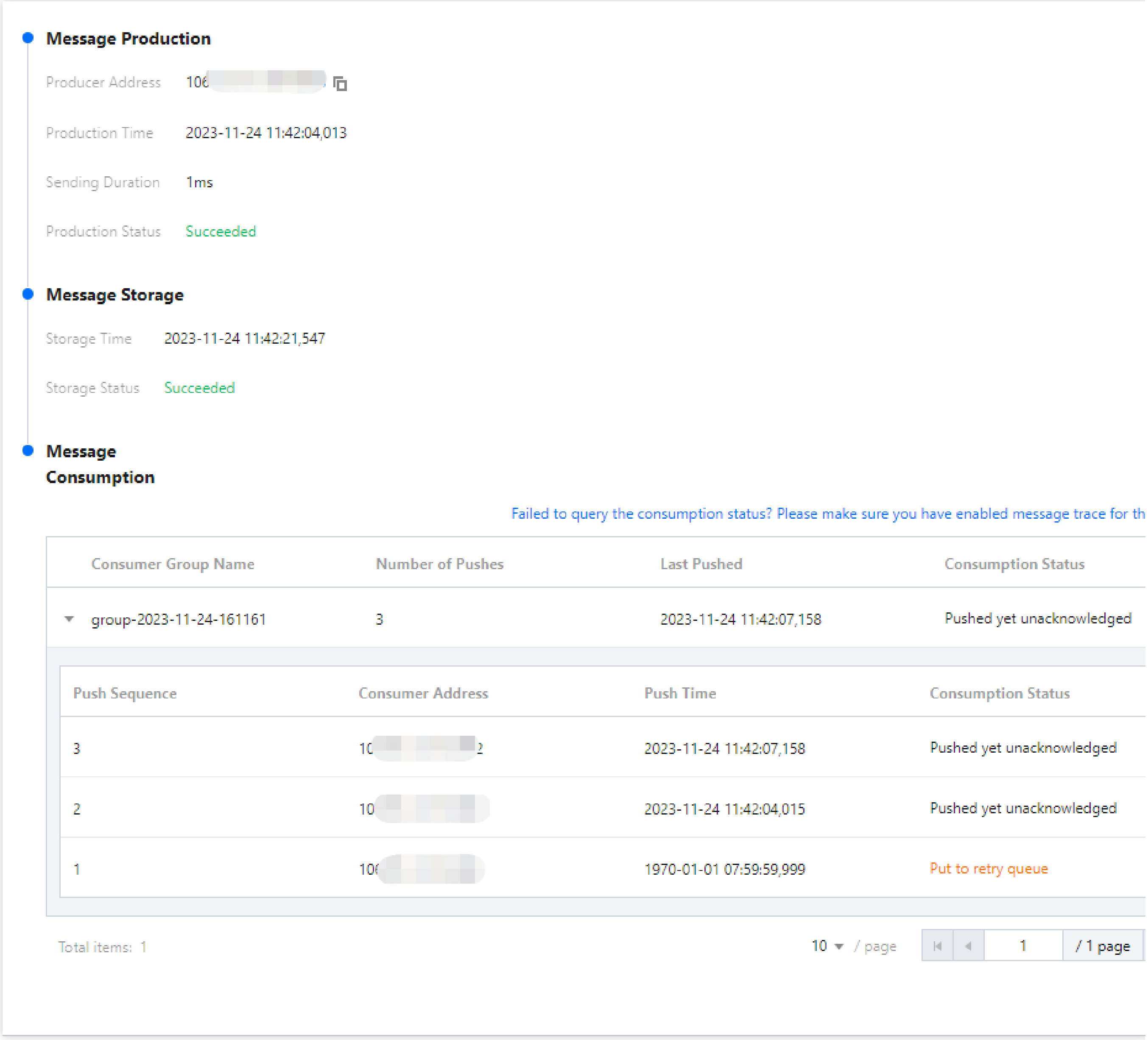
Message Body

Hello RocketMQ 6,pTime:2023-11-24 11:42:04

Parameter Details

```
{
        "RECONSUME_TIME": "0",
        "TRACE_ON": "true",
        "__BORNHOST": "VM-254-196-centos",
        "ORIGIN_MESSAGE_ID": "1EA794A2000022CF0000002BAAE7C0B5",
        "BORN_TIMESTAMP": "1700797324006",
        "RETRY_TOPIC": "rmq-dmam3m5o%topic-400988",
        "MSG_REGION": "gz",
        "KEYS": "1000",
        "UNIQ_KEY": "01525400CEEC5764120571B58C00000006",
        "CLUSTER": "rmqbroker-gz2-pool1",
        "1ST_POP_TIME": "1700797324028",
        "PGROUP": "rmq-dmam3m5o%topic-400988",
        "DELAY": "0",
        "WAIT": "false",
        "TAGS": "1",
```

6. Click **View Message Trace** under the operation column, or select **Message Trace** on the tab bar in the details page. Then you can view the message trace of the message. For details, see the Message Trace Query Results. You will see that when a dead letter message is resent, the consumption status changes to Dead Letter Redelivery Completed.



Message Production

Producer Address 106[redacted]

Production Time 2023-11-24 11:42:04,013

Sending Duration 1ms

Production Status Succeeded

Message Storage

Storage Time 2023-11-24 11:42:21,547

Storage Status Succeeded

Message Consumption

Failed to query the consumption status? Please make sure you have enabled message trace for th

Consumer Group Name	Number of Pushes	Last Pushed	Consumption Status
group-2023-11-24-161161	3	2023-11-24 11:42:07,158	Pushed yet unacknowledged

Push Sequence	Consumer Address	Push Time	Consumption Status
3	10[redacted]2	2023-11-24 11:42:07,158	Pushed yet unacknowledged
2	10[redacted]	2023-11-24 11:42:04,015	Pushed yet unacknowledged
1	10[redacted]	1970-01-01 07:59:59,999	Put to retry queue

Total items: 1

Message Trace Description

Last updated：2024-01-17 16:42:18

A message trace records the entire process of a message from the producer to the TDMQ for RocketMQ server, and ultimately to the consumer, enumerating elements such as the time spent in each stage (accurate to the microsecond), execution outcome, producer IP, and consumer IP.

Prerequisites

You have deployed the producer and consumer services as instructed in the SDK documentation, and there are messages produced and consumed in the last three days.

If you are producing and consuming messages using a client 5.0 or later, there is no need to separately turn on the trace switch on the client.

If you are using a client 4.x, you need to enable the message trace feature on the client itself. The following shows an example of how to implement this setting:

Producer Settings

Push Consumer Settings

Pull Consumer Settings

Spring Boot Starter Access (Version 2.2.2 and Above)

```
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);
```

```
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 the region and follow the on-screen instructions to enter the query conditions.

Time Range: Choose the required time range for the query. Options include the most recent 100 messages (display in chronological order by default), last 30 minutes, last 1 hour, last 6 hours, last 24 hours, last 3 days, or a custom time range.

Cluster: Select the cluster where the Topic you want to query is located.

Topic: Select the Topic you want to query.

Query Method: The following query methods are supported.

Query All: This method will display all messages in the selected Topic within the selected time range.

By message ID: This is a precise and high-speed query method with an exact match.

By message key: This method is a fuzzy query best suited when you have not recorded the message ID but have configured a message key.

3. Click **Query**. The results will be displayed by pages in the list below.

4. Click **View message trace** in the action column, or select **Message Trace** in the details page tab bar to view the corresponding message trace.

Description of Message Trace Query Results

The query results of the message trace are divided into three sections: message production, message storage, and message consumption.

Message Production

Parameter	Description
Production Address	Corresponding producer's address and port.
Production Time	The time when the TDMQ for RocketMQ server acknowledged message receipt, accurate to the millisecond.
Sending duration	The time expended to send the message from the producer to the TDMQ for RocketMQ server, accurate to the microsecond.
Production Status	Whether the message production was successful. A failure generally denotes loss of part of the header data during message transmission, which may result in the preceding fields being NULL.

Message Storage

Parameter	Description
Storage Time	The time when the message is persistently stored.
Storage Duration	The duration between when the message was persistently stored and when the TDMQ for RocketMQ server received the acknowledgment, accurate to the millisecond.
Storage Status	Whether the persistent storage of the message was successful. If the status is failure, the message was not successfully stored, possibly due to disk damage or insufficient capacity. Such situation requires immediate submission of a ticket.

Message Consumption

Message consumption is presented in a list format. TDMQ for RocketMQ supports both cluster consumption and broadcast consumption modes.

The information displayed in the list is as described below:

Parameter	Description
Consumer Group Name	The name of the Consumer Group.

Consumption Mode	The consumption mode of the consumer group, supporting both cluster consumption and broadcast consumption modes.
Number of Pushes	The number of times the TDMQ for RocketMQ server sends the message to the consumer.
Last Pushed Time	The final instance in which the TDMQ for RocketMQ server sent the message to the consumer.
Consumption Status	Unacknowledged: The TDMQ for RocketMQ server has sent the message to the consumer, but has not yet received an acknowledgment from the consumer. Acknowledged: The consumer has sent an acknowledgment (ack) to the TDMQ for RocketMQ server and the server has received the acknowledgment. Retried: The server has not received an acknowledgment and the message is subsequently sent again due to timeout. Retried but Unacknowledged: The TDMQ for RocketMQ server has resent the message to the consumer, but an acknowledgment in response from the consumer has still not been received. Moved to Dead Letter Queue: After a number of unsuccessful consumption attempts, the message has been sent to the dead letter queue. **Note:** If the consumption mode is broadcasting, the only state of consumption is **Pushed**.

Permission Management

Access Authorization for Root Account

Last updated：2024-01-17 16:42:42

Overview

RocketMQ needs to access APIs of other cloud products, so granting RocketMQ permissions to create service roles is required.

Prerequisites

A Tencent Cloud account has been successfully registered.

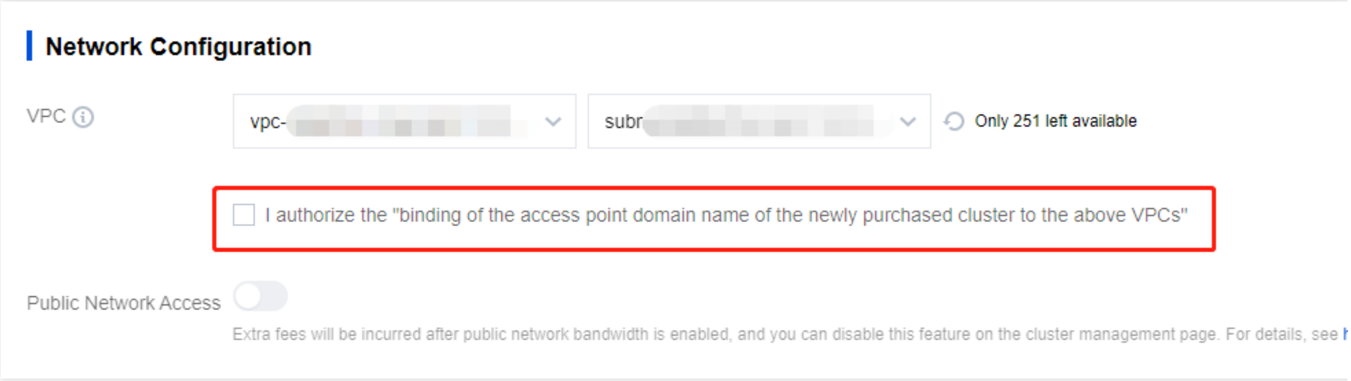
Note:

If you have registered for a Tencent Cloud account, a root account is automatically created by the system for quick access to Tencent Cloud resources.

Directions:

1. Log in to the TDMQ for RocketMQ console, choose **RocketMQ** > **Cluster Management** from the left-hand navigation bar, and then click on **Create Cluster**.

2. During the network configuration on the purchase cluster page, upon selecting a VPC and then ticking the option "I authorize the “binding of the access point domain name of the newly purchased cluster to the above VPCs", a prompt window for authorization will appear.



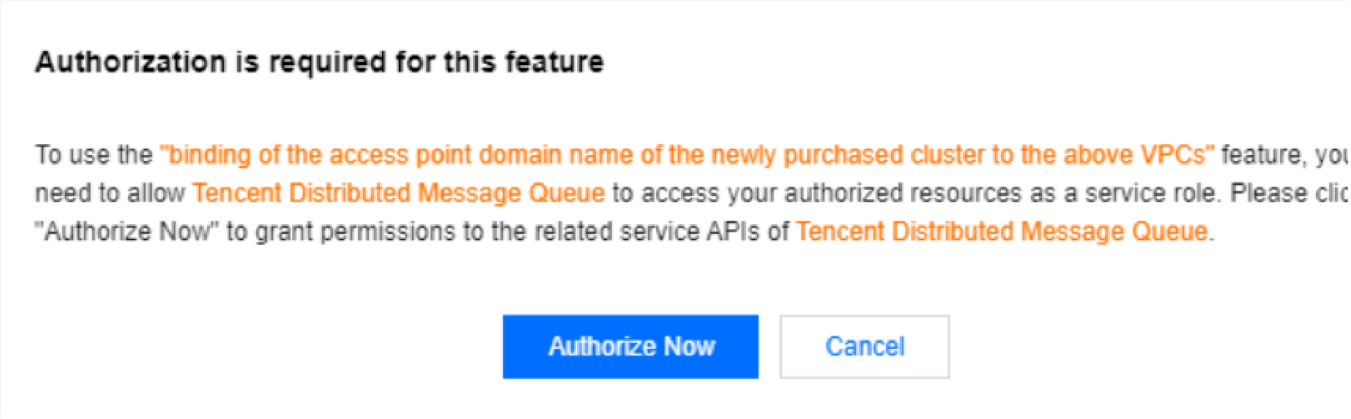
Network Configuration

VPC ⓘ vpc-[redacted] subn[redacted] Only 251 left available

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

Public Network Access ○

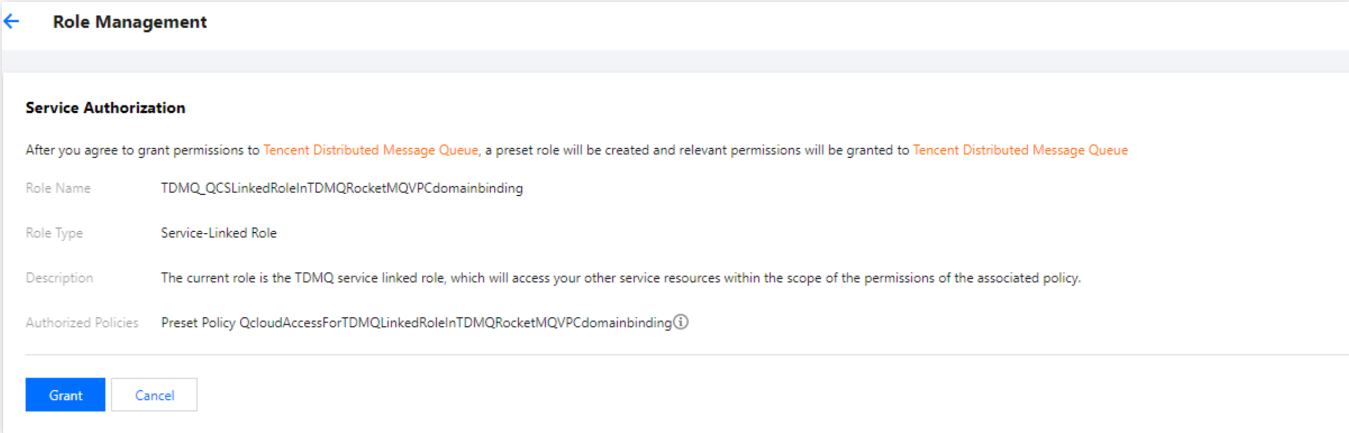
Extra fees will be incurred after public network bandwidth is enabled, and you can disable this feature on the cluster management page. For details, see



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 clic "Authorize Now" to grant permissions to the related service APIs of Tencent Distributed Message Queue.

3. Click on **Authorize** to navigate to the CAM console. Click on **Authorize Now** to assign the TDMQ RocketMQ service a role to access your other cloud service 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 ⓘ

4. After you have completed the authorization process, you can create a RocketMQ cluster and use the relevant services.

Access Authorization for Sub-Account

Granting Sub-Account Access to TDMQ for RocketMQ

Last updated：2024-01-17 16:43:19

Basic Concepts in CAM

The root account authorizes sub-accounts by associating them with policies. The policies can be configured to the granularity of **API, Resource, User/User Group, Allow/Deny, and Condition**.

Account System

Root Account: Possesses all Tencent Cloud resources and has the capability to access any of its resources.
Sub-account: Comprised of sub-users and collaborators.
Sub-user: Created and fully owned by a root account.
Collaborator: Possesses the identity of a root account. If an account is added as a collaborator to a current root account, it is one of the sub-accounts and can switch back to its original root account identity.
Identity Credentials: Includes both login credentials and access certificates. **Login Credentials** refer to usernames and passwords, **and Access Certificates** refer to API keys (SecretId and SecretKey).

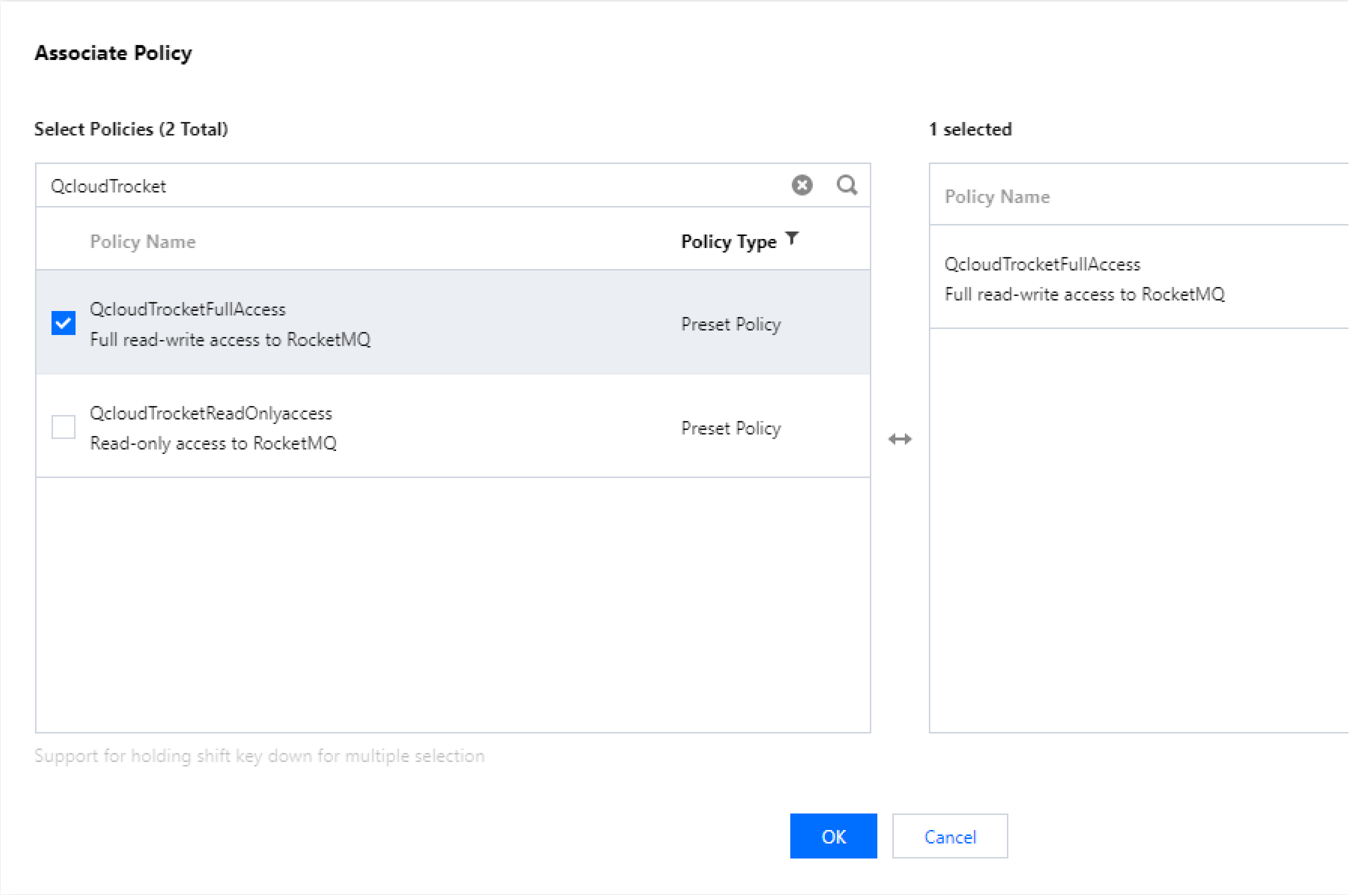
Resources and Permissions

Resources: An object within the cloud service that is subjected to operations, such as a CVM instance, a COS bucket, and a VPC instance.
Permissions: Permissions refer to allowing or rejecting certain users to perform certain actions. By default, **the root account has unrestricted access to all resources under it**, and **a sub-account possesses no access to any resources under its root account**.
Policy: A syntactical guideline that defines and describes one or more permissions. **The root account** performs authorization by **associating policies** with users/user groups.

Sub-account Using RocketMQ

To ensure that a sub-account can successfully use RocketMQ, the root account must grant authorization to the sub-account.
Use the root account to log in to the CAM console, locate the appropriate sub-account within the sub-account list, and then click on **Authorize** in the action column.

RocketMQ offers two preset policies for sub-accounts: QcloudTrocketReadOnlyaccess and QcloudTrocketFullAccess. The former only allows viewing related information in the console, and the latter allows read and write operations on the product console.



Associate Policy

Select Policies (2 Total)

QcloudTrocket

	Policy Name	Policy Type
☑	QcloudTrocketFullAccess Full read-write access to RocketMQ	Preset Policy
□	QcloudTrocketReadOnlyaccess Read-only access to RocketMQ	Preset Policy

1 selected

Policy Name
QcloudTrocketFullAccess Full read-write access to RocketMQ

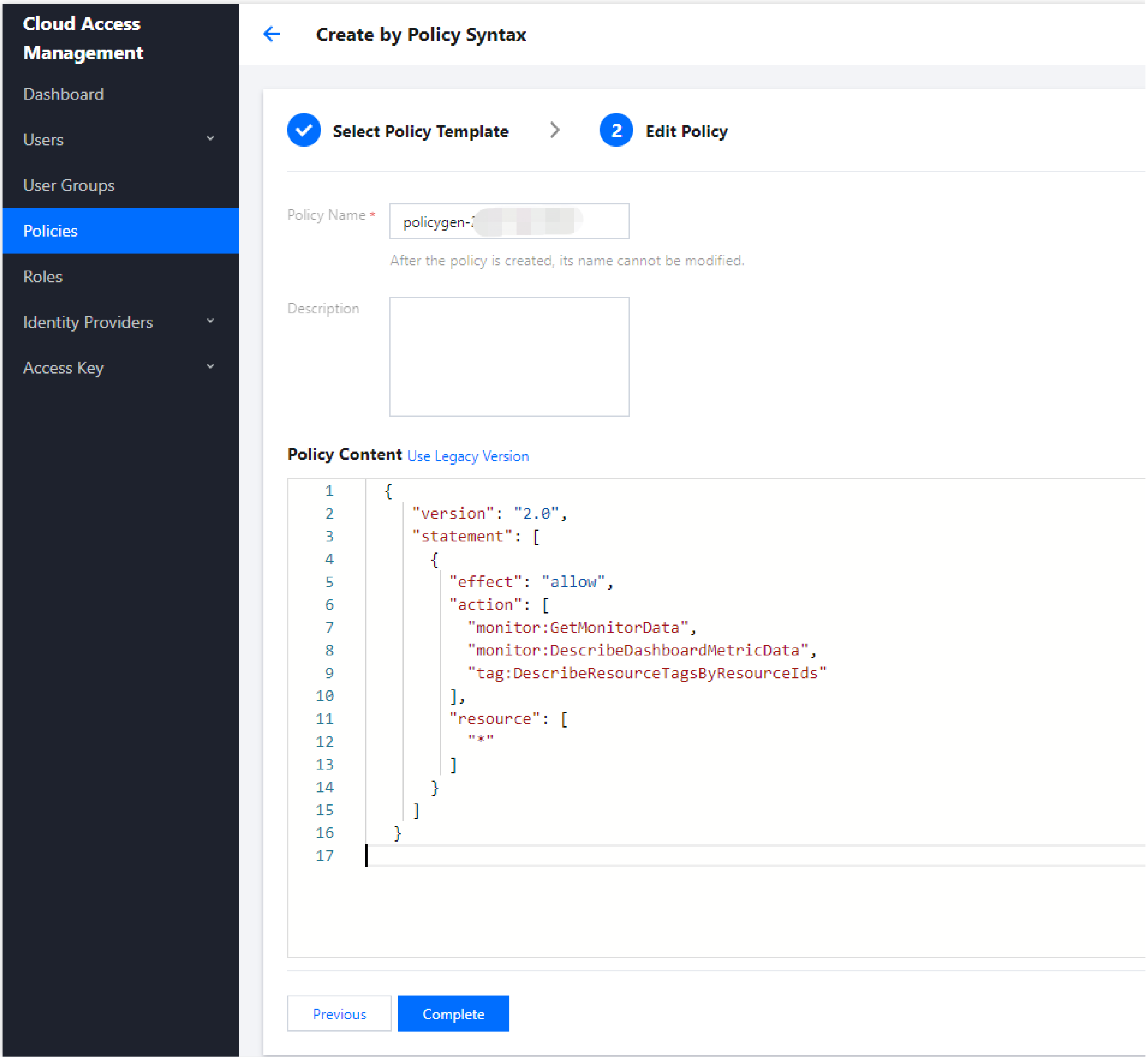
Support for holding shift key down for multiple selection

Apart from the preceding preset policies, the root account also needs to grant the sub-account permissions to call other cloud services as needed to better usage. The use of RocketMQ involves the following corresponding API permissions of cloud services:

Cloud Service	API Name	API Features	Corresponding Features in RocketMQ
TCOP (Monitor)	GetMonitorData	Queries monitoring metric data	Views corresponding monitoring metrics displayed in the console
TCOP (Monitor)	DescribeDashboardMetricData	Queries monitoring metric data	Views corresponding monitoring metrics displayed in the console
Resource tags	DescribeResourceTagsByResourceIds	Queries resource tags	Views resource tags of the cluster

To grant the preceding permissions to a sub-account, the root account needs to perform the **Create Custom Policy** operation on **Policies** page of the CAM Console. After clicking **Create by syntax** for creation, select a **Blank Template** and enter the following policy syntax:

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



Policy Name * policygen-[redacted]

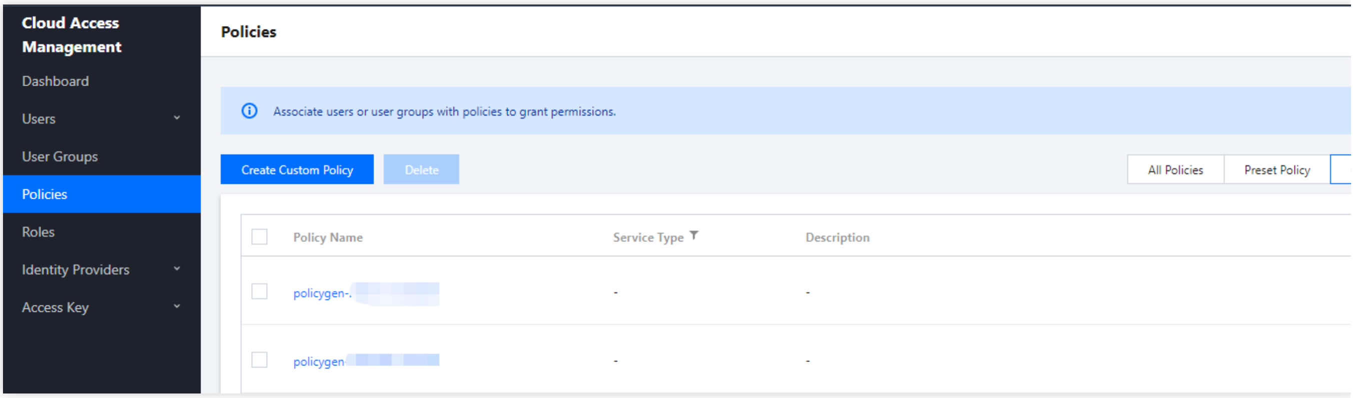
After the policy is created, its name cannot be modified.

Description

Policy Content Use Legacy Version

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

After creating the policy, associate the newly created policy with the sub-account under the operation column. See the following figure:



Associate users or user groups with policies to grant permissions.

Policy Name	Service Type	Description
policygen-.[redacted]	-	-
policygen[redacted]	-	-

##Related Documents

Operational Level Authorization

Resource Level Authorization

Tag-Level Authorization

Granting Operation-Level Permissions to Sub-Accounts

Last updated：2024-01-17 16:43:30

Overview

This documentation describes how to use a Tencent Cloud root account to grant operation-level permissions to a sub-account. You can grant different read/write permissions to the sub-account as needed.

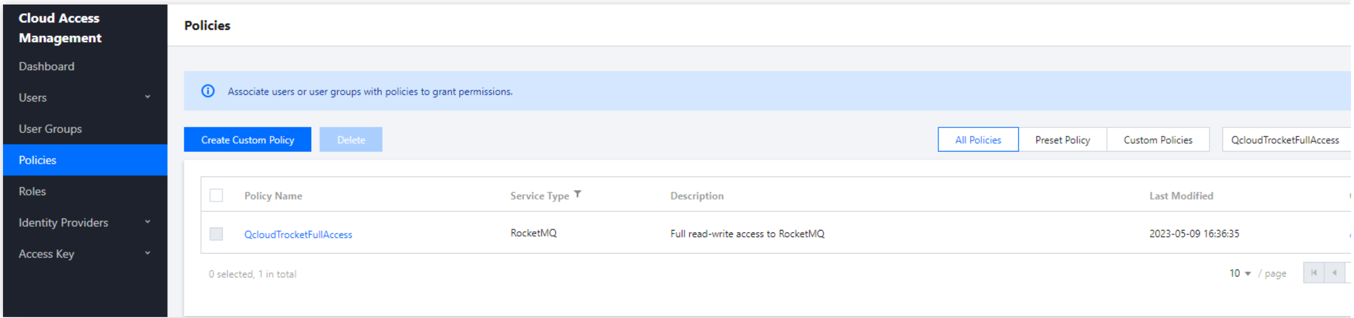
Directions:

Granting Full Read/Write Permissions

Note:

Once a sub-account is granted full read/write permissions, the sub-account will possess **full read/write capabilities** over **all resources** under the root account.

1. Log in to the CAM console with your root account.
2. Click **Policies** in the left sidebar to access the policy management list page.
3. In the search bar on the right, enter **QcloudTrocketFullAccess** and search.

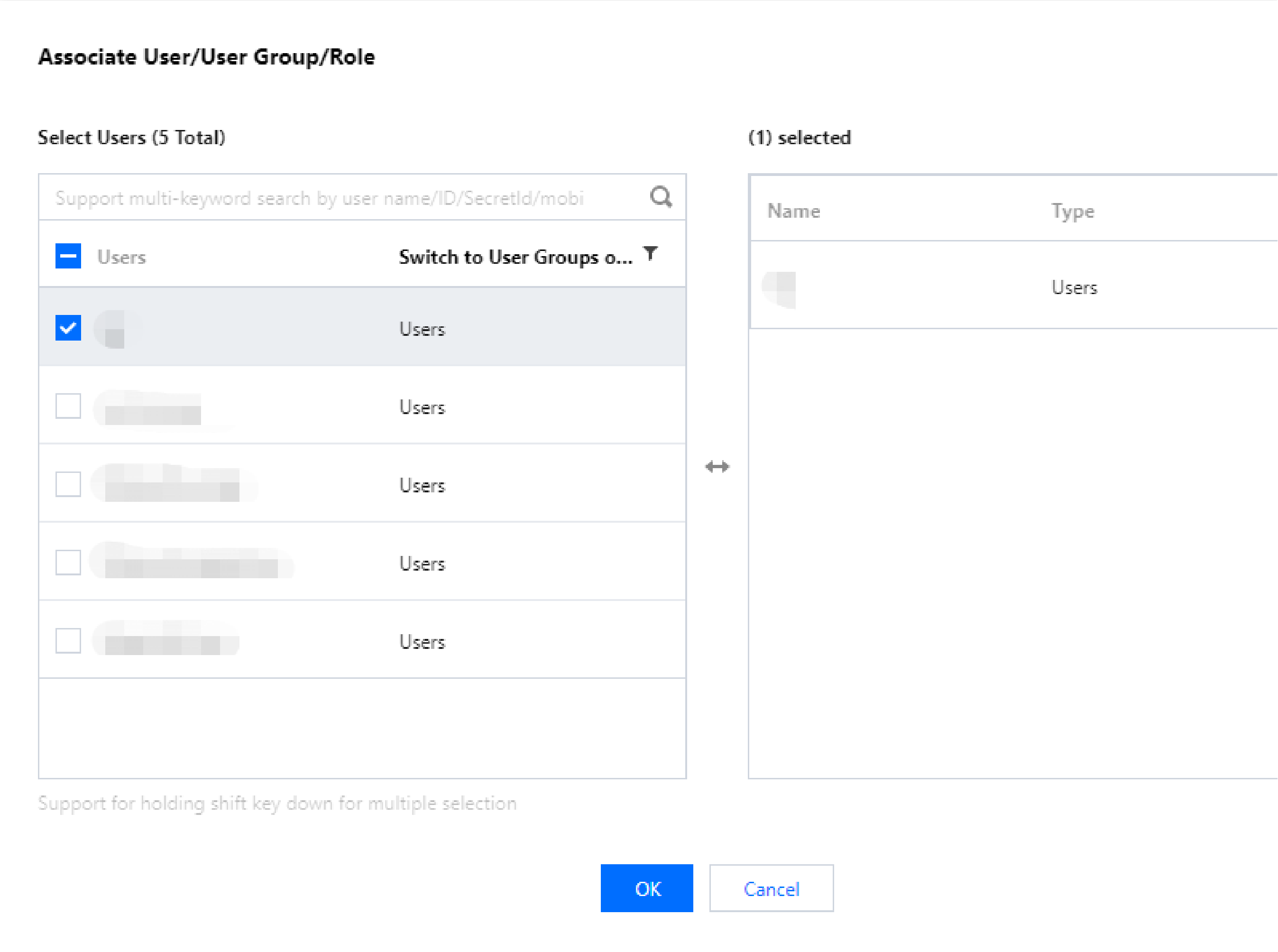


Associate users or user groups with policies to grant permissions.

□	Policy Name	Service Type	Description	Last Modified
□	QcloudTrocketFullAccess	RocketMQ	Full read-write access to RocketMQ	2023-05-09 16:36:35

0 selected, 1 in total

4. In the search results, click the **Associate Users/User Groups/Role** of **QcloudTrocketFullAccess** and choose the sub-account you want to authorize.



Associate User/User Group/Role

Select Users (5 Total)

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

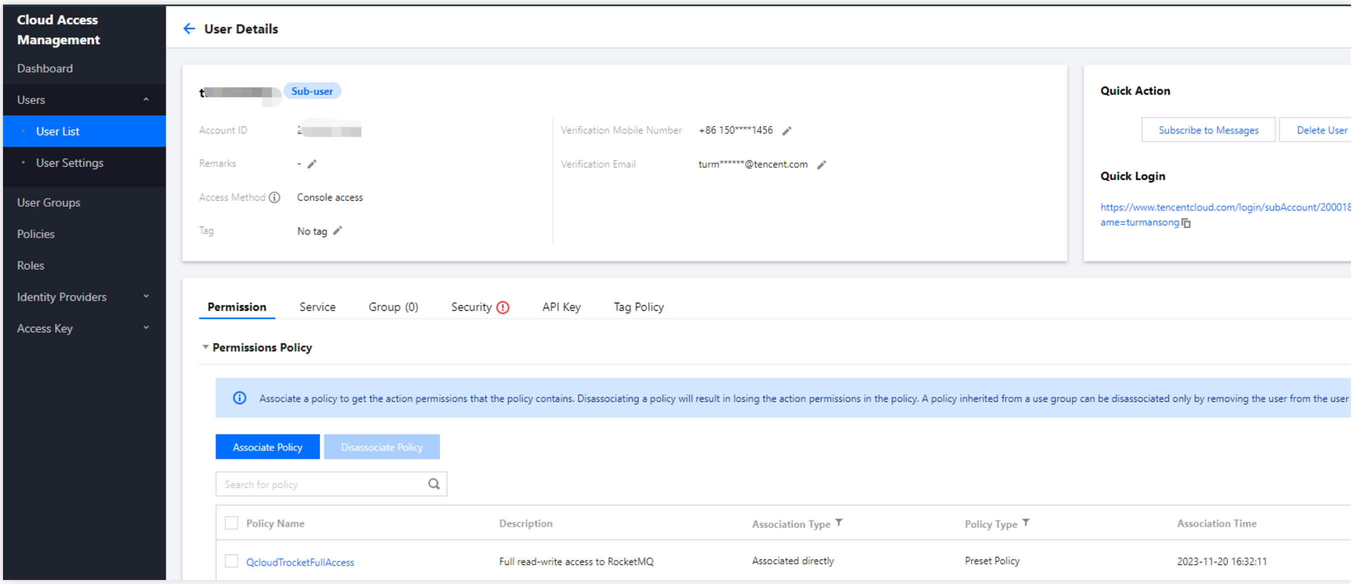
☐	Users	Switch to User Groups o...
☑	[redacted]	Users
☐	[redacted]	Users
☐	[redacted]	Users
☐	[redacted]	Users
☐	[redacted]	Users

Support for holding shift key down for multiple selection

(1) selected

Name	Type
[redacted]	Users

5. Click **OK** to complete the authorization. This policy will then be displayed in the user's policy list.



t[redacted] Sub-user

Account ID 2[redacted]

Remarks -

Access Method Console access

Tag No tag

Verification Mobile Number +86 150****1456

Verification Email turm******@tencent.com

https://www.tencentcloud.com/login/subAccount/200018
ame=turmansong

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 from the user

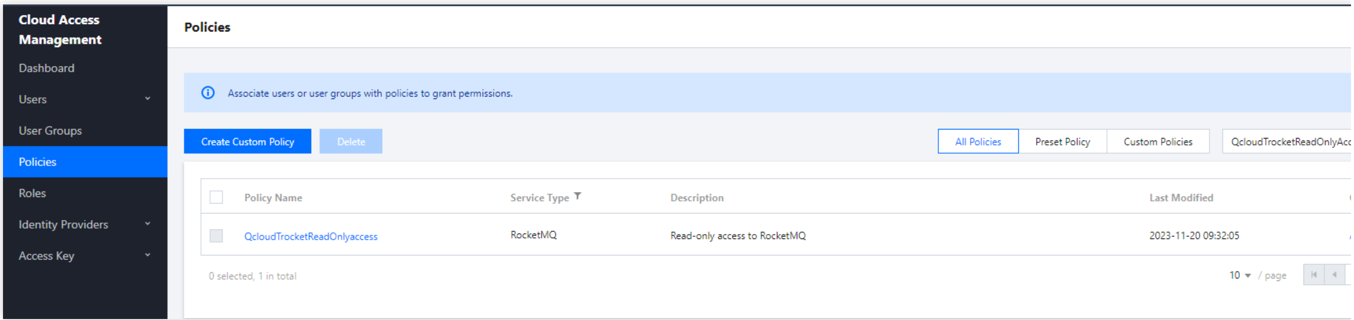
☐ Policy Name	Description	Association Type	Policy Type	Association Time
☐ QcloudTrocketFullAccess	Full read-write access to RocketMQ	Associated directly	Preset Policy	2023-11-20 16:32:11

Granting Read-Only Permission

Note:

Once a sub-account is granted read-only permission, it will possess **read-only capabilities** for **all resources** under the root account.

1. Log in to the CAM console with your root account.
2. Click **Policies** on the left-hand navigation bar to access the policy management list page.
3. In the search bar on the right, enter **QcloudTrocketReadOnlyAccess** and search.

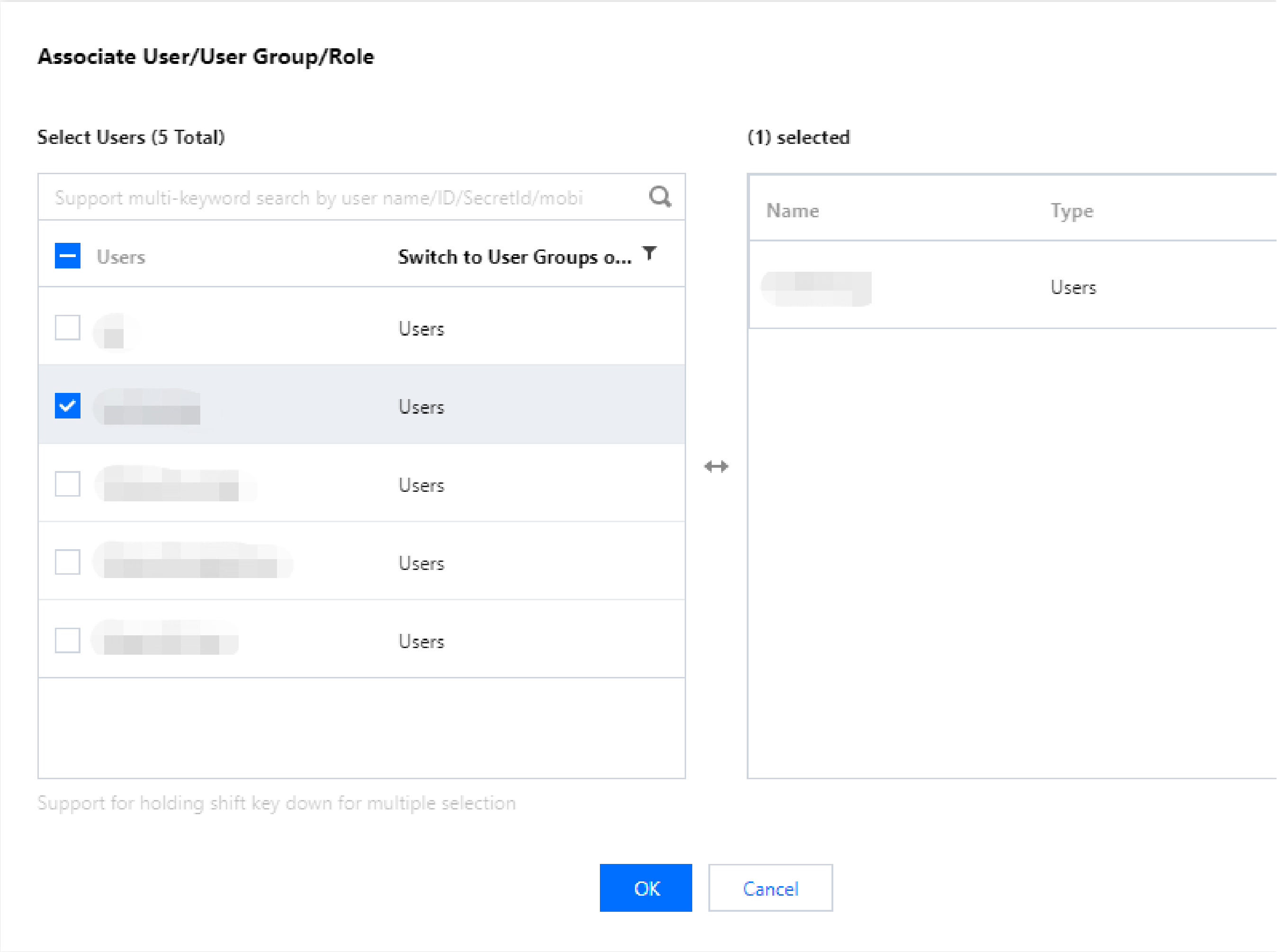


Associate users or user groups with policies to grant permissions.

□	Policy Name	Service Type	Description	Last Modified
□	QcloudTrocketReadOnlyaccess	RocketMQ	Read-only access to RocketMQ	2023-11-20 09:32:05

0 selected, 1 in total

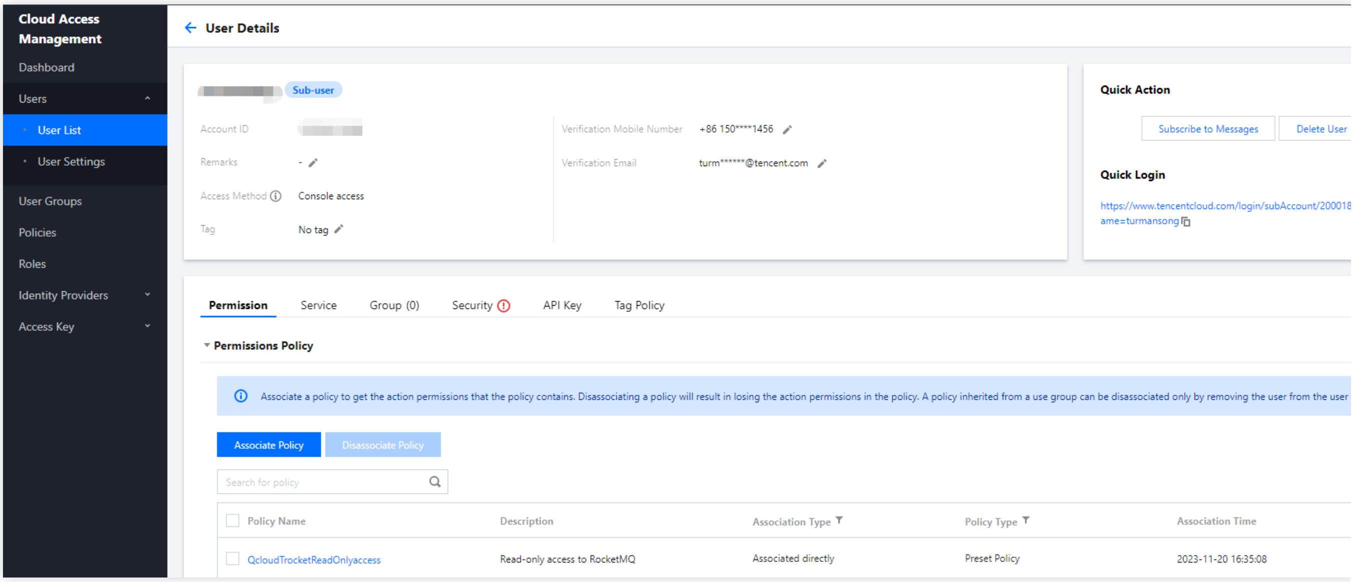
4. In the search results, click on the **Associate Users/User Groups/Role** of **QcloudTrocketReadOnlyAccess** and choose the sub-account you want to authorize.


Associate User/User Group/Role
Select Users (5 Total)
(1) selected
Support multi-keyword search by user name/ID/SecretId/mobi
Users
Switch to User Groups o...
Support for holding shift key down for multiple selection
OK
Cancel

	Users	Switch to User Groups o...
□		Users
☑		Users
□		Users
□		Users
□		Users

Name	Type
	Users

5. Click **OK** to complete the authorization. This policy will then be displayed in the user's policy list.



[redacted] Sub-user

Account ID [redacted]

Remarks -

Access Method Console access

Tag No tag

Verification Mobile Number +86 150****1456

Verification Email turm******@tencent.com

Quick Login

https://www.tencentcloud.com/login/subAccount/200018
ame=turmansong

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 from the user

□ Policy Name	Description	Association Type	Policy Type	Association Time
□ QcloudTrocketReadOnlyaccess	Read-only access to RocketMQ	Associated directly	Preset Policy	2023-11-20 16:35:08

Other Authorization Methods

Resource Level Authorization

Tag-Level Authorization

Granting Resource-Level Permissions to Sub-Accounts

Last updated：2024-01-17 16:43:42

Overview

This document describes how to use the root account to grant a sub-account resource-level permissions. After the authorization, the sub-account will possess control over a specific resource.

Operation Prerequisites

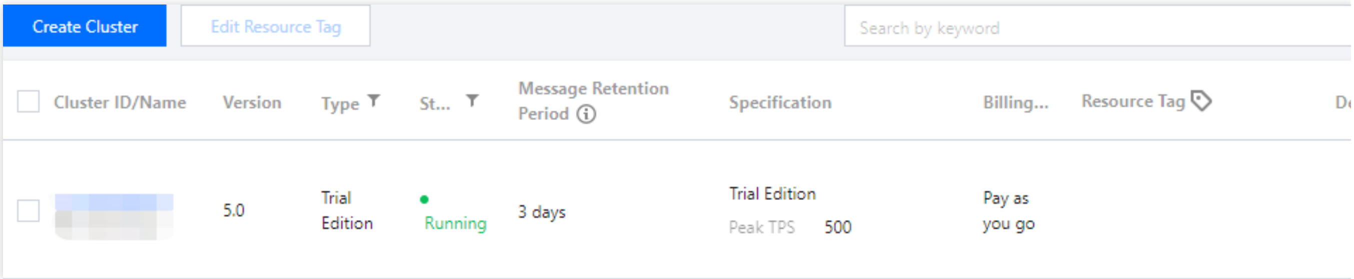
You have a Tencent Cloud root account and have activated the Tencent Cloud CAM service.
The root account should have at least one sub-account, and authorization has been granted according to "Retrieving access permissions for sub-accounts".
You have at least one RocketMQ instance.

Directions:

You can use the policy feature in the CAM console to grant a sub-account permissions of the root account's RocketMQ resources. For details, see **Granting RocketMQ Resources to Sub-Accounts**. This example demonstrates how to grant a cluster resource to a sub-account. The operation for other resource types are similar.

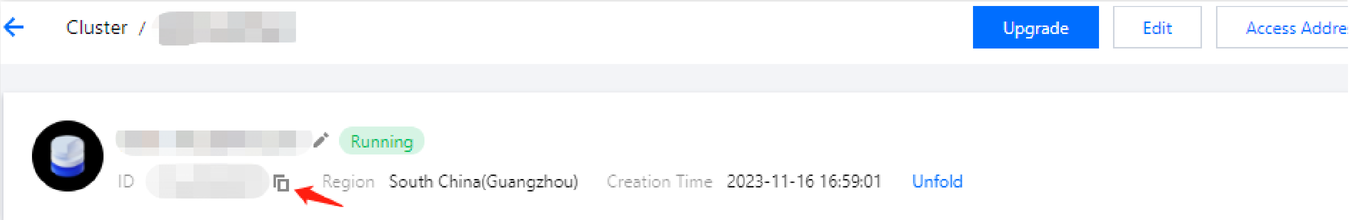
Step 1: Acquiring the Resource ID of the RocketMQ Cluster

1. Use the **root account** to log in to the TDMQ for RocketMQ console, select an existing cluster instance, and click to open the details page.



□ Cluster ID/Name	Version	Type	St...	Message Retention Period ⓘ	Specification	Billing...	Resource Tag	D
□ [redacted]	5.0	Trial Edition	● Running	3 days	Trial Edition Peak TPS 500	Pay as you go		

2. In **Basic Info**, the field **ID** is the ID of the current RocketMQ cluster.

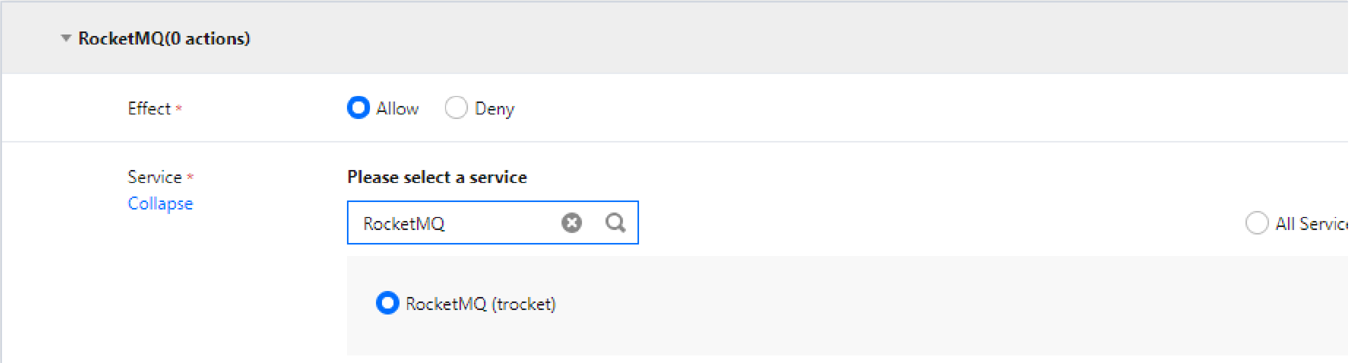


Region South China(Guangzhou)

Creation Time 2023-11-16 16:59:01

Step 2: Creating an Authorization Policy

1. Open the CAM console and click **Policies** on the left sidebar.
2. Click **Create Custom Policy,** and choose **Create by Policy Generator**.
3. In the visual policy generator, keep **Effect** set to **Allow**. In **Service**, enter "rocketmq" to filter and select **RocketMQ (trocket)** from the results.



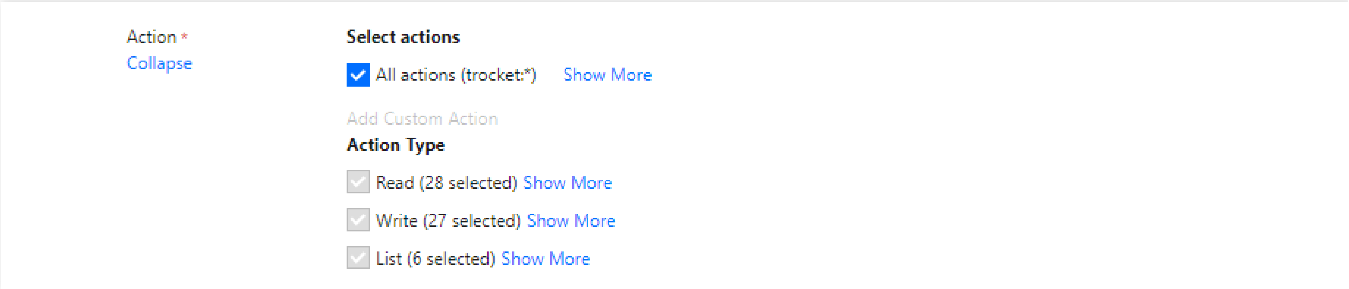
Effect * ● Allow ○ Deny

Service * Please select a service

RocketMQ

● RocketMQ (trocket)

4. Select **All Actions** in **Action**. You can also select action types as needed.



Action * Select actions

☑ All actions (trocket:*) Show More

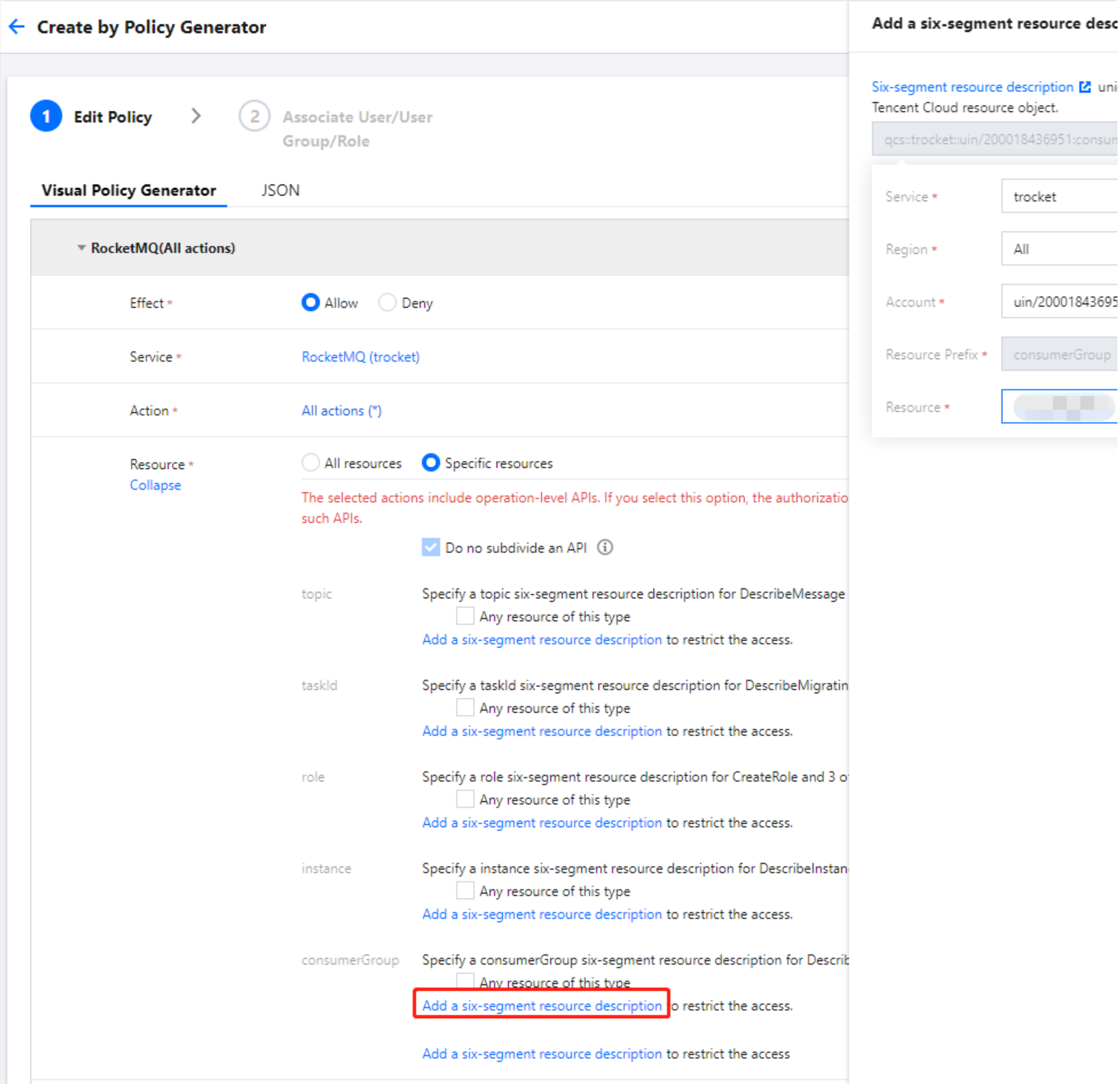
Action Type

☑ Read (28 selected) Show More

☑ Write (27 selected) Show More

☑ List (6 selected) Show More

5. In **Resource**, select **Specific resources**. You can either select **Any resource of this type (grant access to all resources in this category)** on the right, or click on **Add a Six-segment Resource description (authorize specific resources)**.
6. In the displayed sidebar under **Resource**, specify the **ID** of the resource you want to authorize. For the acquisition procedure, see Step 1.



← Create by Policy Generator

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

Visual Policy Generator JSON

RocketMQ(All actions)

Effect * ● Allow ○ Deny

Service * RocketMQ (trocket)

Action * All actions (*)

Resource * Collapse ○ All resources ● Specific resources

The selected actions include operation-level APIs. If you select this option, the authorizatio such APIs.

☑ Do no subdivide an API

topic: Specify a topic six-segment resource description for DescribeMessage
□ Any resource of this type
Add a six-segment resource description to restrict the access.

taskId: Specify a taskId six-segment resource description for DescribeMigratin
□ Any resource of this type
Add a six-segment resource description to restrict the access.

role: Specify a role six-segment resource description for CreateRole and 3 o
□ Any resource of this type
Add a six-segment resource description to restrict the access.

instance: Specify a instance six-segment resource description for DescribeInstan
□ Any resource of this type
Add a six-segment resource description to restrict the access.

consumerGroup: Specify a consumerGroup six-segment resource description for Describ
□ Any resource of this type
Add a six-segment resource description to restrict the access.

Add a six-segment resource description to restrict the access

Add a six-segment resource desc

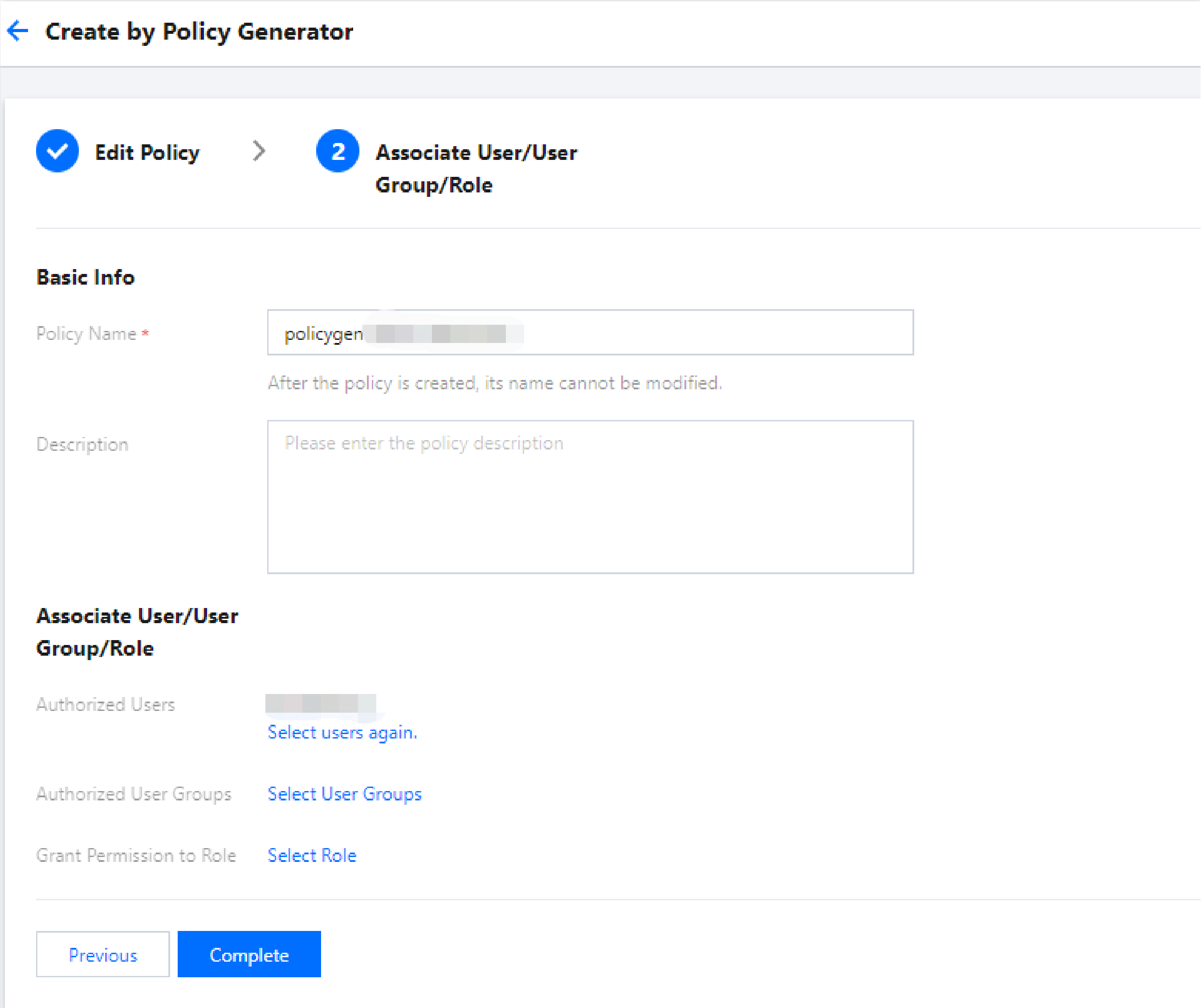
Six-segment resource description uni
Tencent Cloud resource object.

qcs::trocket::uin/200018436951:consun

Service *	trocket
Region *	All
Account *	uin/20001843695
Resource Prefix *	consumerGroup
Resource *	[redacted]

7. Click Next and fill in the policy name as needed.

8. Click **Select Users** or **Select User Groups** to choose the user or user group that needs to be granted resource permissions.



Create by Policy Generator

Edit Policy > 2 Associate User/User Group/Role

Basic Info

Policy Name * policygen[redacted]

After the policy is created, its name cannot be modified.

Description Please enter the policy description

Associate User/User Group/Role

Authorized Users [redacted]
Select users again.

Authorized User Groups Select User Groups

Grant Permission to Role Select Role

9. Click **Complete**. The sub-accounts granted with resource permissions can access the related resources.

Other Authorization Methods

Operational Level Authorization

Tag-Level Authorization

Appendix

List of APIs Supporting Resource-Level Authorization

TDMQ supports resource-level authorization, enabling you to bestow upon a particular sub-account, the API permissions of a specific resource.

The APIs supporting resource-level authorization are as follows:

API Name	API Description	Resource Type	Six-Segment Example of Resource
CreateConsumerGroup	Creates consumer groups	consumerGroup	qcs::trocket:${region}:uin/${uin}:consumerG
CreateInstance	Creates instances	instance	qcs::trocket:${region}:uin/${uin}:instance/*
CreateInstanceEndpoint	Creates access points	instance	qcs::trocket:${region}:uin/${uin}:instance/${i
CreateRole	Adds roles	role	qcs::trocket:${region}:uin/${uin}:role/${instar
CreateTopic	Creates topics	topic	qcs::trocket:${region}:uin/${uin}:topic/${insta
DeleteConsumerGroup	Deletes consumer groups	consumerGroup	qcs::trocket:${region}:uin/${uin}:consumerG
DeleteInstance	Deletes instances	instance	qcs::trocket:${region}:uin/${uin}:instance/${i
DeleteInstanceEndpoint	Deletes access points	instance	qcs::trocket:${region}:uin/${uin}:instance/${i
DeleteRole	Deletes roles	role	qcs::trocket:${region}:uin/${uin}:role/${instar
DeleteTopic	Deletes topics	topic	qcs::trocket:${region}:uin/${uin}:topic/${insta
DescribeConsumerClient	Queries consumer client details	consumerGroup	qcs::trocket:${region}:uin/${uin}:consumerG
DescribeConsumerClientList	Queries client connections under consumer group	consumerGroup	qcs::trocket:${region}:uin/${uin}:consumerG

DescribeConsumerGroup	Queries consumer group details	consumerGroup	qcs::trocket:${region}:uin/${uin}:consumerG
DescribeConsumerGroupList	Queries consumer group lists	consumerGroup	qcs::trocket:${region}:uin/${uin}:consumerG
DescribeInstance	Queries instances	instance	qcs::trocket:${region}:uin/${uin}:instance/${i
DescribeInstanceList	Queries instance lists	instance	qcs::trocket:${region}:uin/${uin}:instance/${i
DescribeInstanceTopUsages	Obtains instance resource consumption ranking	instance	qcs::trocket:${region}:uin/${uin}:instance/${i
DescribeMessage	Queries messages	topic	qcs::trocket:${region}:uin/${uin}:topic/${insta
DescribeMessageList	Queries message lists	topic	qcs::trocket:${region}:uin/${uin}:topic/${insta
DescribeMessageTrace	Queries message traces	topic	qcs::trocket:${region}:uin/${uin}:topic/${insta
DescribeRoleList	Queries role lists	role	qcs::trocket:${region}:uin/${uin}:role/${instar
DescribeTopic	Queries topic details	topic	qcs::trocket:${region}:uin/${uin}:topic/${insta
DescribeTopicList	Queries topic lists	topic	qcs::trocket:${region}:uin/${uin}:topic/${insta
DescribeTopicListByGroup	Obtains topic lists based on the	consumerGroup	qcs::trocket:${region}:uin/${uin}:consumerG

	consumer group		
DescribeTopicStatisticalList	Obtains the number and types of topics under a specified instance	instance	qcs::trocket:${region}:uin/${uin}:instance/${i
ExportMessage	Exports messages	topic	qcs::trocket:${region}:uin/${uin}:topic/${insta
ModifyConsumerGroup	Modifies consumer group attributes	consumerGroup	qcs::trocket:${region}:uin/${uin}:consumerG
ModifyInstance	Modifies instances	instance	qcs::trocket:${region}:uin/${uin}:instance/${i
ModifyInstanceEndpoint	Modifies access points	instance	qcs::trocket:${region}:uin/${uin}:instance/${i
ModifyRole	Modifies roles	role	qcs::trocket:${region}:uin/${uin}:role/${instai
ResetConsumerGroupOffset	Resets consumption offset	consumerGroup	qcs::trocket:${region}:uin/${uin}:consumerG
SendMessage	Sends messages	topic	qcs::trocket:${region}:uin/${uin}:topic/${insta
VerifyMessageConsumption	Verifies message consumption	topic	qcs::trocket:${region}:uin/${uin}:topic/${insta

Granting Tag-Level Permissions to Sub-Accounts

Last updated：2024-01-17 16:43:53

Overview

This document describes how to use a root account to grant a sub-account access to resources under a specific tag through tag authorization. The authorized sub-account can gain control over the resources associated with the corresponding tag.

Prerequisites

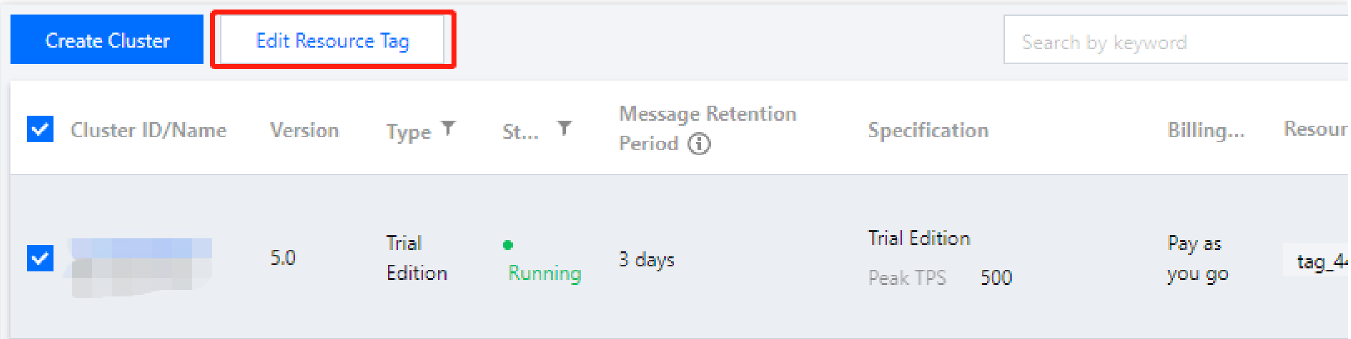
You have a Tencent Cloud root account and have activated the Tencent Cloud CAM service.
The root account should have at least one sub-account, and authorization has been granted according to "Retrieving access permissions for sub-accounts".
You have at least one RocketMQ cluster resource instance.
You have at least one **tag**. If you do not have one, you can go to TAG control panel > **TAG list** to create one.

Directions

You can use the policy feature of the CAM console to grant the sub-account read/write permissions for RocketMQ resources owned by the root account and already bound to a tag, by **authorizing by TAG**. For details, see **Granting Resource Permissions to Sub-Accounts by TAG**.

Step 1: Binding Tags to Resources

1. Use the **root account** to log in to the MQ for RocketMQ console, and navigate to the cluster management page.
2. Select the target cluster and then click **Edit Resource Tag** in the upper left corner to bind a tag to the cluster.



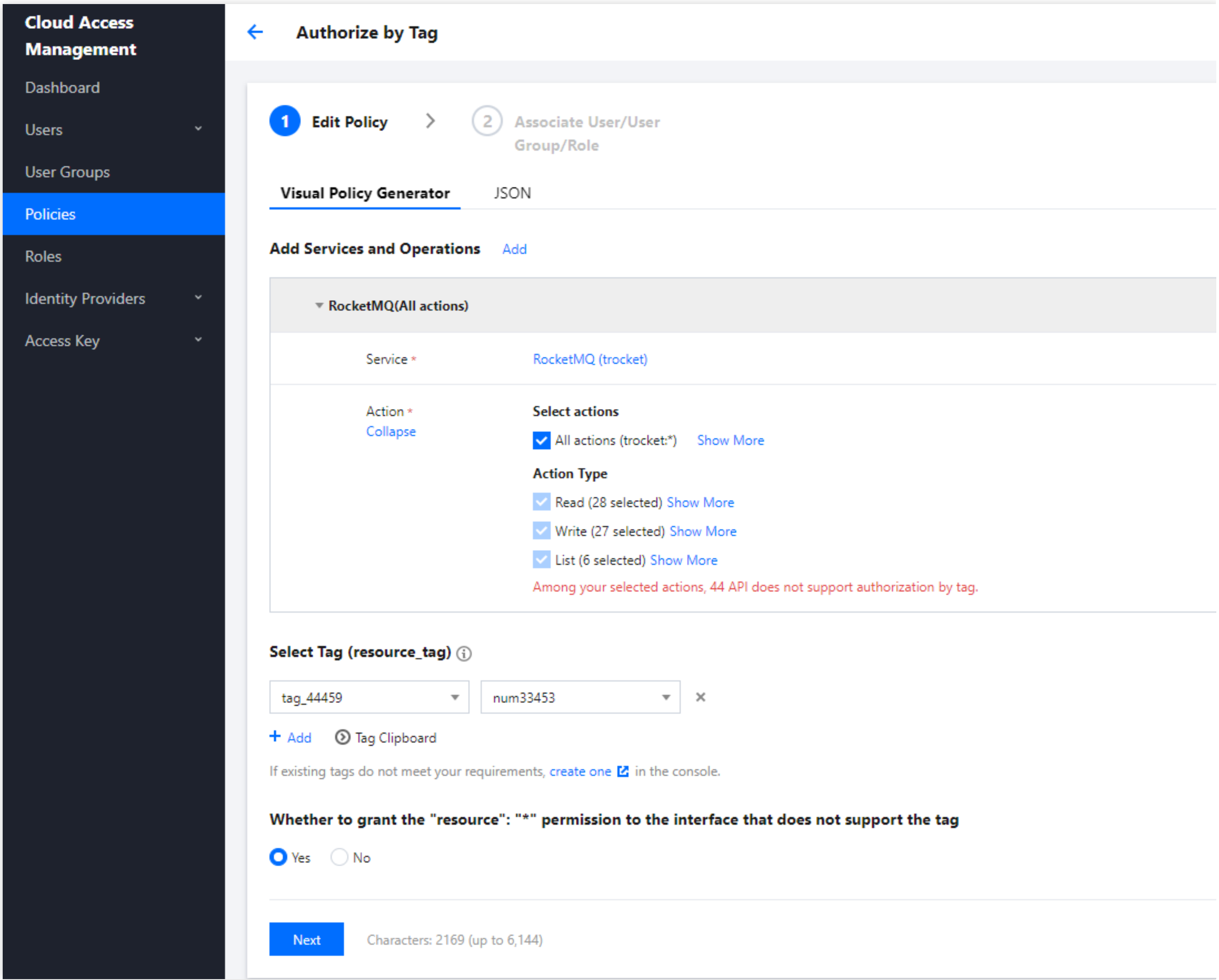
☑ Cluster ID/Name	Version	Type	St...	Message Retention Period	Specification	Billing...	Resour
☑ [redacted]	5.0	Trial Edition	● Running	3 days	Trial Edition Peak TPS 500	Pay as you go	tag_4

Step 2: Authorizing by Tag

1. Open the CAM console and click **Policies** on the left sidebar.

2. Click **Create Custom Policy,** and select **Authorize by TAG**.

3. In the visual policy generator, enter "rocketmq" in the **service** to filter. Select **TROCKET(trocket)** from the results. Select **All actions** in **Action**, or select the corresponding operation as needed.



Add Services and Operations Add

RocketMQ(All actions)

Service * RocketMQ (trocket)

Action * Collapse

Select actions

☑ All actions (trocket:*) Show More

Action Type

☑ Read (28 selected) Show More

☑ Write (27 selected) Show More

☑ List (6 selected) Show More

Among your selected actions, 44 API does not support authorization by tag.

Select Tag (resource_tag)

tag_44459 num33453 ×

+ Add Tag Clipboard

If existing tags do not meet your requirements, create one in the console.

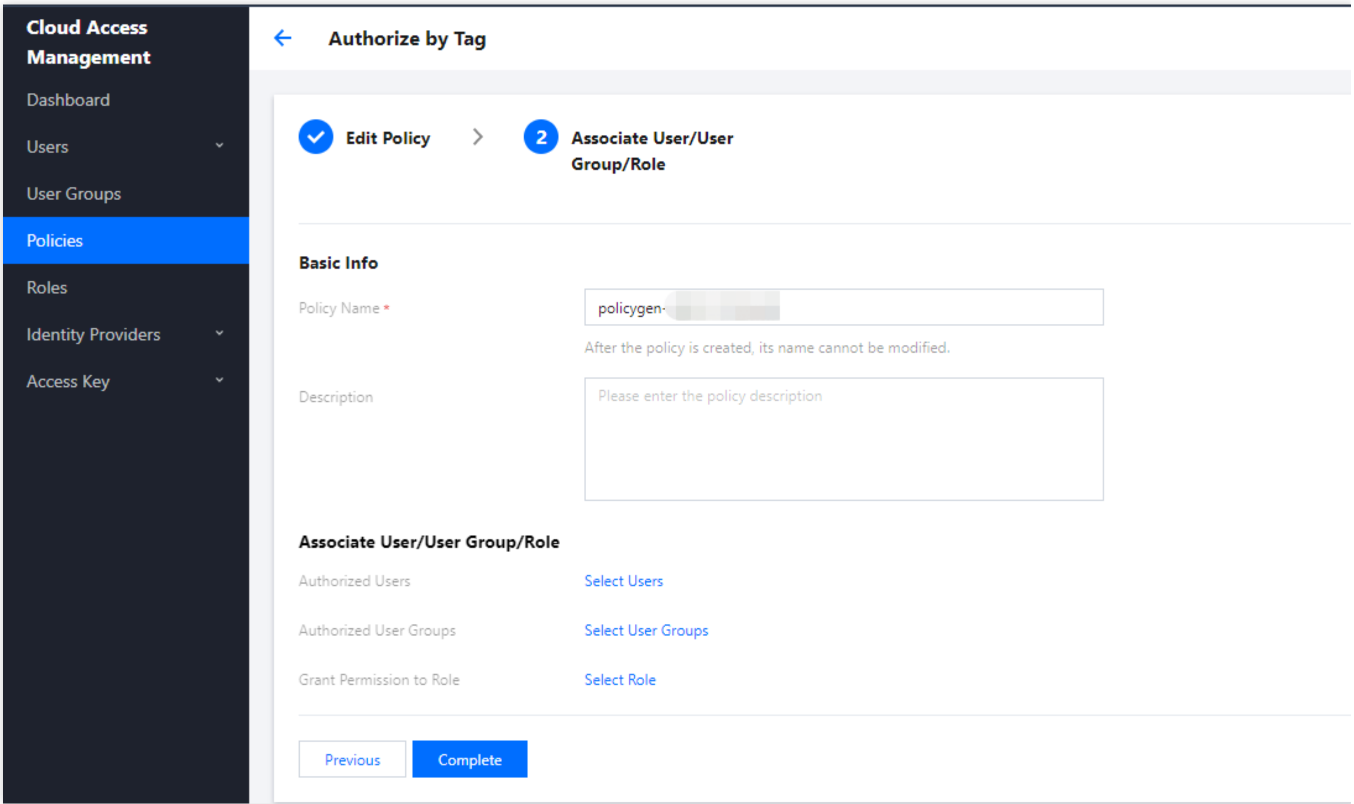
Whether to grant the "resource": "*" permission to the interface that does not support the tag

● Yes ○ No

Next Characters: 2169 (up to 6,144)

4. Click Next and fill in the policy name as needed.

5. Click **Select Users** or **Select User Groups** to choose the user or user group that needs to be granted resource permissions.

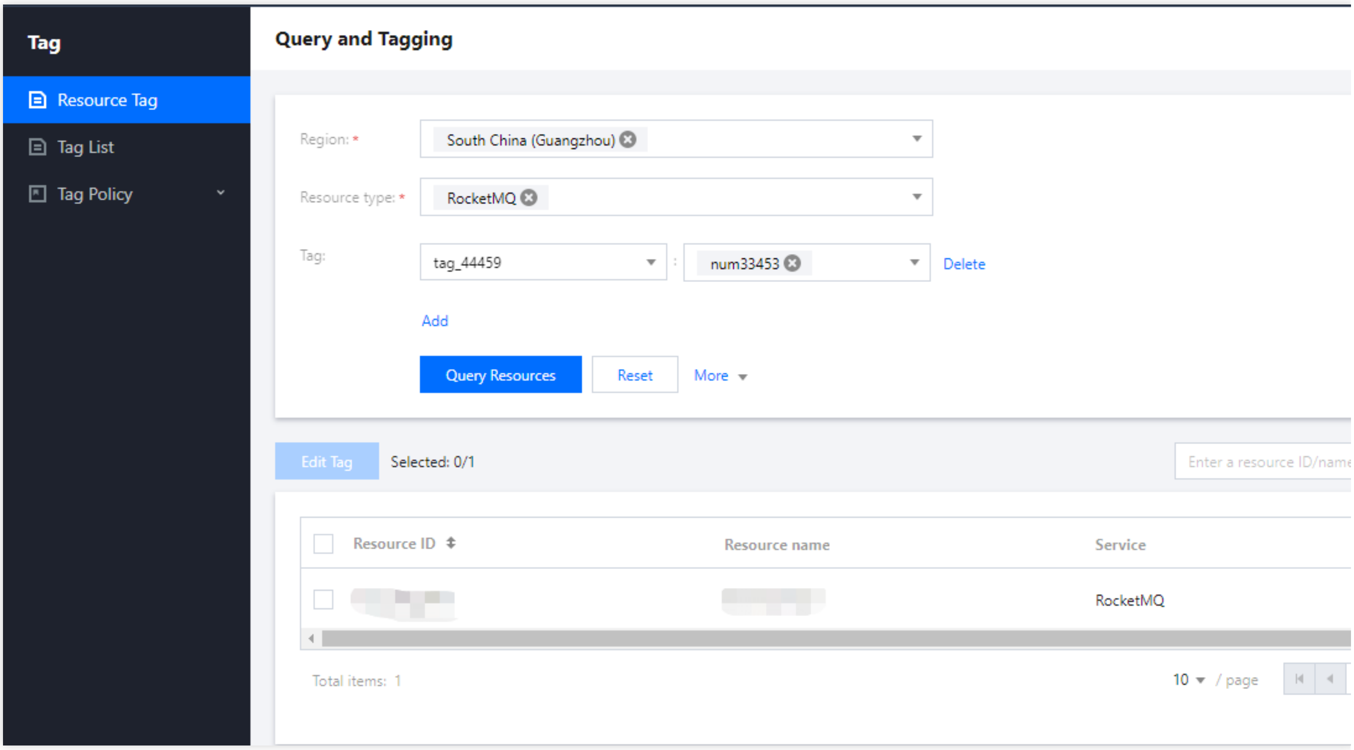


6. Click **Complete**. The corresponding sub-accounts can now control resources under the specified tag according to the policy.

Unified Management of Resource Tags

You can also manage resource tags in a unified manner on the **TAG Console**. The detailed operations are as follows:

1. Log in to the **Tag console**.
2. On the left sidebar, select Resource Tag and choose the query conditions as needed, and then choose **TROCKET** > **RocketMQ Instance** in **Resource type**.
3. Click **Query Resources**.
4. Select the required resources in the results, and click **Edit Tag**. You can bind or unbind tags in batches.



Region: * South China (Guangzhou)

Resource type: * RocketMQ

Tag: tag_44459 : num33453

□	Resource ID	Resource name	Service
□	[redacted]	[redacted]	RocketMQ

Total items: 1

Other Authorization Methods

Operational Level Authorization

Resource Level Authorization

Tag Management

Last updated：2024-01-26 17:07:58

Overview

Tag is a key-value pair provided by Tencent Cloud to identify a resource in the cloud. You can use it to 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 configure. They are only used for you to manage TDMQ for RocketMQ resources.

Use Restrictions

Please be aware of the following restrictions when using tags:

Restriction Type	Restrictions
Quantity limit	Each cloud resource allows up to 50 tags.
Tag key restrictions	`qcloud` , `tencent` , and `project` are system reserved tag keys and cannot be created. A Tag key can only contain `numbers` , `letters` , and `+=.@-` symbols. The maximum length of a tag key is 255 characters.
Tag value restrictions	A tag value can only contain `empty strings or numbers` , `letters` , and `+=.@-` symbols. The maximum length of a tag value is 127 characters.

Operation Procedure and Use Cases

Use Case Description

A company has six TDMQ for RocketMQ clusters on Tencent Cloud. The following shows the department, business scope, and responsible person information of the six clusters:

Queue ID	Department of Usage	Business Scope	Responsible Person
rocketmq-qzga74ov5gw1	E-commerce	Marketing campaigns	John
rocketmq-qzga74ov5gw2	E-commerce	Marketing campaigns	Jack

rocketmq-qzga74ov5gw3	Games	Game A	John
rocketmq-qzga74ov5gw4	Games	Game B	Jack
rocketmq-qzga74ov5gw5	Entertainment	Post-production	Jack
rocketmq-qzga74ov5gw6	Entertainment	Post-production	John

Using 'rocketmq-qzga74ov5gw1' as an example, we can add the following three sets of tags to this instance:

Tag Key	Tag Value
dept	ecommerce
business	mkt
owner	John

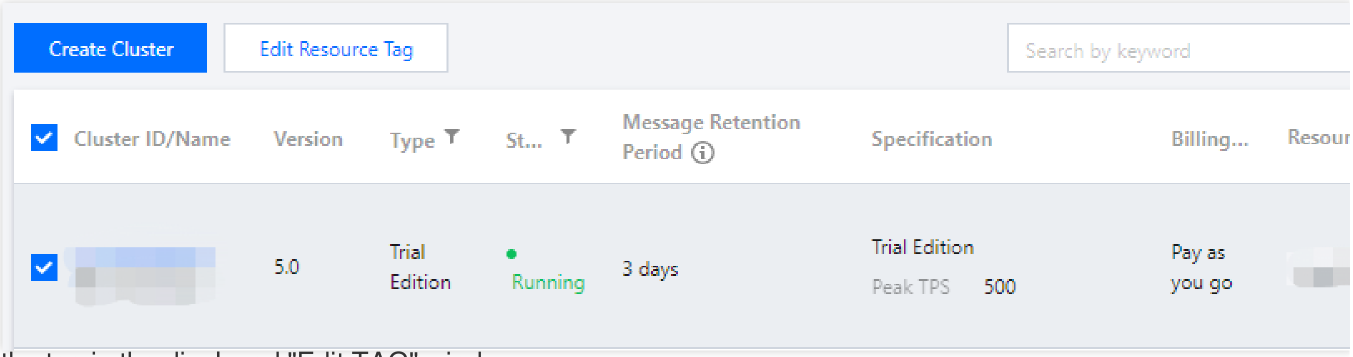
Appropriate tags can also be assigned to other queue resources in the same way according to their department, business scope, and responsible person.

Setting TAG on the TDMQ for RocketMQ Console

The preceding scenario is used as an example. After you design the tag key and tag value, you can log in to the TDMQ for RocketMQ console to configure the tags.

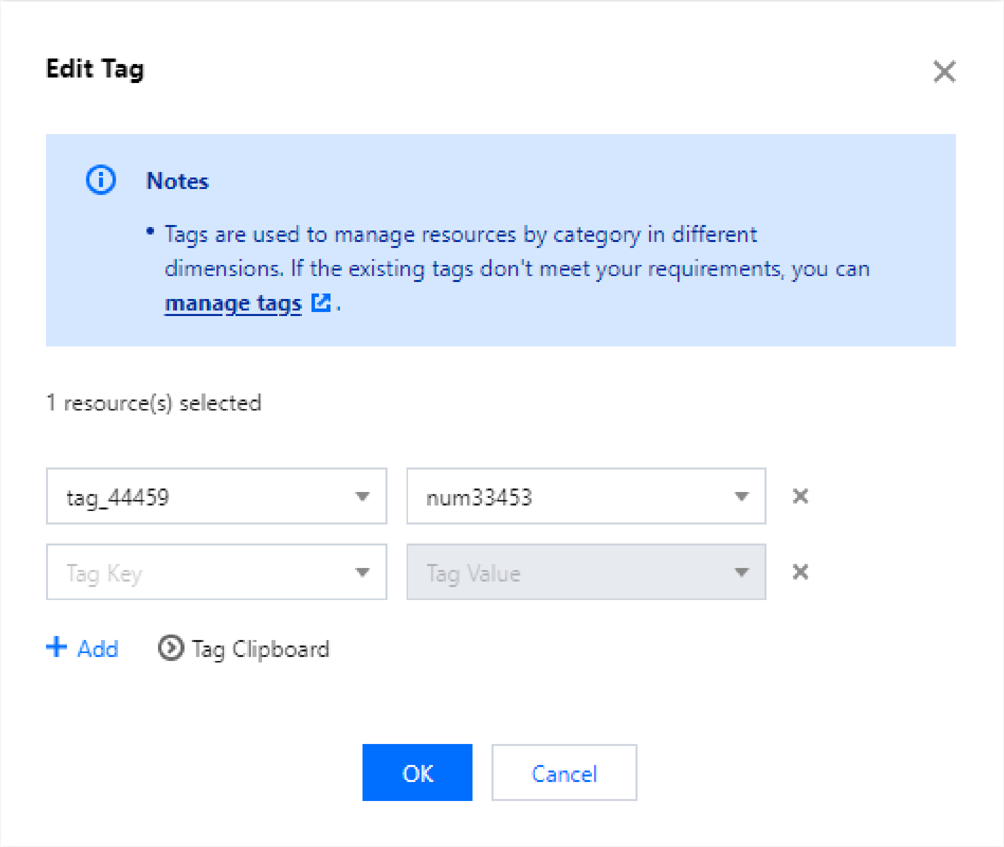
1. Log in to the Console for RocketMQ.

2. In the cluster management list page, select the appropriate region. Then, tick the cluster for which you want to edit the tag and click **Edit Resource TAG** at the top of the page.



☑ Cluster ID/Name	Version	Type	St...	Message Retention Period ⓘ	Specification	Billing...	Resour
☑ [redacted]	5.0	Trial Edition	● Running	3 days	Trial Edition Peak TPS 500	Pay as you go	[redacted]

3. Set the tag in the displayed "Edit TAG" window.



Edit Tag

Notes

- 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

Tag Key Tag Value

Note:

If existing tags do not meet your specifications, go to TAG Management to create new tags.

4. Click **OK**. After receiving a successful modification prompt from the system, view the bound tags in the cluster's resource tag column.

Filtering Resources with a Tag Key

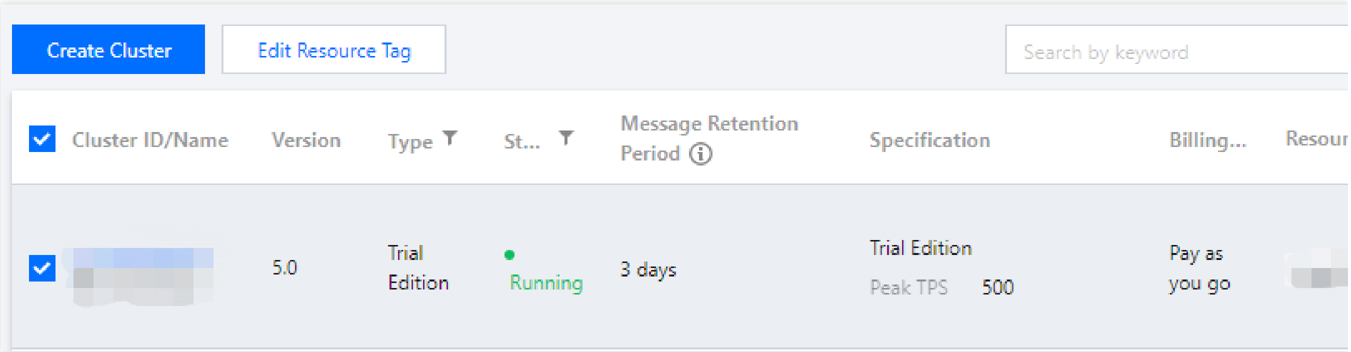
If you want to filter clusters bound to the corresponding tag, perform the following operations.

1. Select **TAG** from the search box located in the upper right corner of the page.

2. In the window that is displayed after **TAG:**, select the tag you want to search for and click **OK** to initiate the search. For example, by selecting `TAG:owner:zhangsan` , you can filter out clusters associated with the tag key `owner:zhangsan` .

Editing Tags

1. In the cluster management list page, select the appropriate region. Then, tick the cluster for which you want to edit the tag and click **Edit Resource TAG** at the top of the page.



☑	Cluster ID/Name	Version	Type	St...	Message Retention Period	Specification	Billing...	Resour
☑	[redacted]	5.0	Trial Edition	● Running	3 days	Trial Edition Peak TPS 500	Pay as you go	[redacted]

Note:

You can batch edit tags for a maximum of 20 resources at a time.

2. In the displayed "Edit TAG" window, add, modify, or delete tags as needed.

Development Guide

Message Type

General Message

Last updated：2023-11-14 16:34:59

A general message is a basic type of message that is delivered to a specified topic by production and consumed by consumers who subscribe to the topic. There is no order in the topic of general messages, and multiple partitions can be used to improve the efficiency of message production and consumption. The performance of general messages is optimal when the throughput is huge.

General messages are different from scheduled and delayed messages, sequential messages, and transactional messages. The topics corresponding to these four message types cannot be mixed and can only be used for sending and receiving messages of the same type. For instance, the topic of a general message can only be used to send and receive general messages and cannot be used to send and receive delayed, sequential, and transactional messages.

Scheduled Message and Delayed Message

Last updated：2024-01-17 16:47:17

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

Concepts

Scheduled message: In actual business, after a message is sent to the server, the consumer is expected to receive it at a later time point rather than immediately. This type of message is called scheduled message.
Delayed message: In actual business, after a message is sent to the server, the consumer is expected to receive it after a period of time rather than immediately. This type of message is called delayed message.
Actually, the delayed message can be regarded as a special type of scheduled message, which is essentially the same thing.

Instructions

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 messages and delayed messages in TDMQ for RocketMQ. You can also view the complete code sample >>.

Scheduled Messages

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(StandardCharse
// 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("2021-10-01
    // 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) {
```

Delayed Messages

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(StandardCharse

// 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 Restrictions

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 one second for scheduled and delayed messages.

The time ranges of scheduled and delayed messages varies based on different cluster specifications. For details, see Product Series.

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：2024-01-17 16:51:45

Sequential message is an advanced type of message offered 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 messages and general messages is as follows:

Message Type	Consumption Sequence	Performance	Use Cases
General message	No sequence	High	Suitable for scenarios with high demands for throughput and no requirement for production and consumption sequence.
Sequential message	Messages in a specified Topic following the FIFO rule	Average	Scenarios having average demands for throughput and requiring publishing and consuming all messages in a specified Topic in strict accordance with the FIFO rule

Sequential messages are often used in the following actual business scenarios:

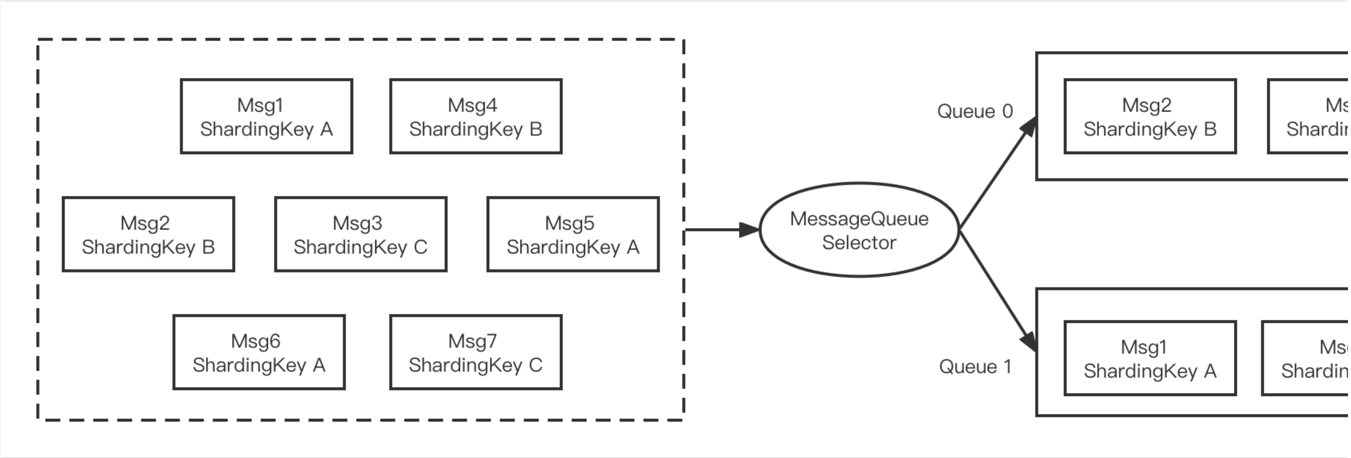
Order creation: In some e-commerce systems, the creation, payment, refund, and logistics messages of an order must be produced or consumed in strict sequence. Otherwise, the order status delivery will be messed up, affecting normal businesses. Therefore, the messages of this order must be produced and consumed in a certain sequence in the client and message queue. In addition, the messages are sequentially dependent, and the processing of the next message depend on the processing result of the preceding message.

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

Financial scenarios: In some matchmaking transaction scenarios such as certain securities trading, priority is given to those who bid first case of the same biding 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. Messages of the same ShardingKey will be assigned to the same queue and consumed in sequence.



The code for sequential messages is as follows:

```
public class Producer {
    public static void main(String[] args) throws UnsupportedEncodingException {
        try {
            DefaultMQProducer producer = new DefaultMQProducer("please_rename_uniqu
            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_CHA
                SendResult sendResult = producer.send(msg, new MessageQueueSelector
                    @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 | Interr
            e.printStackTrace();
        }
    }
}
```

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

The `MessageQueueSelector` API is as follows:

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

Where, mqs is the queue that can be sent, msg is the message, arg is the object passed in the preceding send API, and the queue to which the message needs to be sent is returned. In the preceding sample, orderId is used as the partition classification standard, and the remainder of the number of all queues is obtained to send messages with the same orderId to the same queue.

In the production environment, it is recommended 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:

To ensure high availability of messages, TDMQ for RocketMQ currently does not support "globally sequential messages" in a single queue (users who have already created globally sequential messages can continue to use them normally). If you want to ensure global sequence, you can use a consistent ShardingKey.

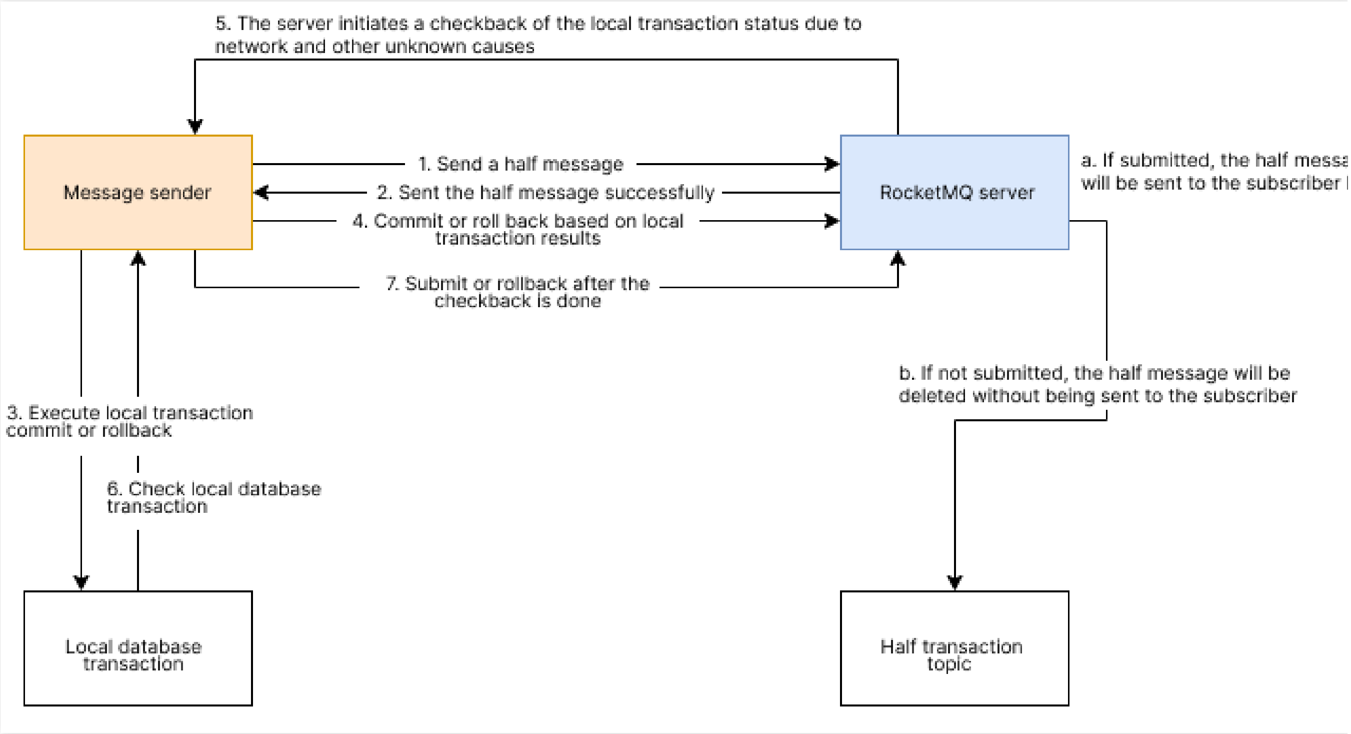
Transactional Message

Last updated：2024-01-17 16:52:56

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

Feature 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.


5. The server initiates a checkback of the local transaction status due to network and other unknown causes
Message sender
1. Send a half message
2. Sent the half message successfully
4. Commit or roll back based on local transaction results
RocketMQ server
a. If submitted, the half messa
will be sent to the subscriber l
7. Submit or rollback after the checkback is done
b. If not submitted, the half message will be deleted without being sent to the subscriber
3. Execute local transaction commit or rollback
6. Check local database transaction
Local database transaction
Half transaction topic

1. The producer sends a message to RocketMQ (1).
2. After receiving the message, the server stores it in the half message topic (2).
3. Local transaction is executed (3).
4. The producer proactively sends the transaction execution result to TDMQ for RocketMQ (4).
5. If the local transaction execution result has not been returned after a certain period of time, TDMQ for RocketMQ will execute the recheck logic (5).
6. After receiving the message recheck, 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. Consumers can consume the message.

TransactionStatus.ROLLBACK: Rolls back the transaction. The message is discarded without being consumed by consumers.

TransactionStatus.UN_KNOW: Unknown status, indicating the waiting of another recheck.

7. When the transaction is successfully executed, TDMQ for 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 billed transaction chain is usually long with a significant chance of error or timeout. TDMQ for RocketMQ'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.

Message Filtering

Last updated：2024-01-17 16:53:09

This document describes the features, application scenarios, and usage instructions of message filtering in TDMQ for RocketMQ.

Feature Description

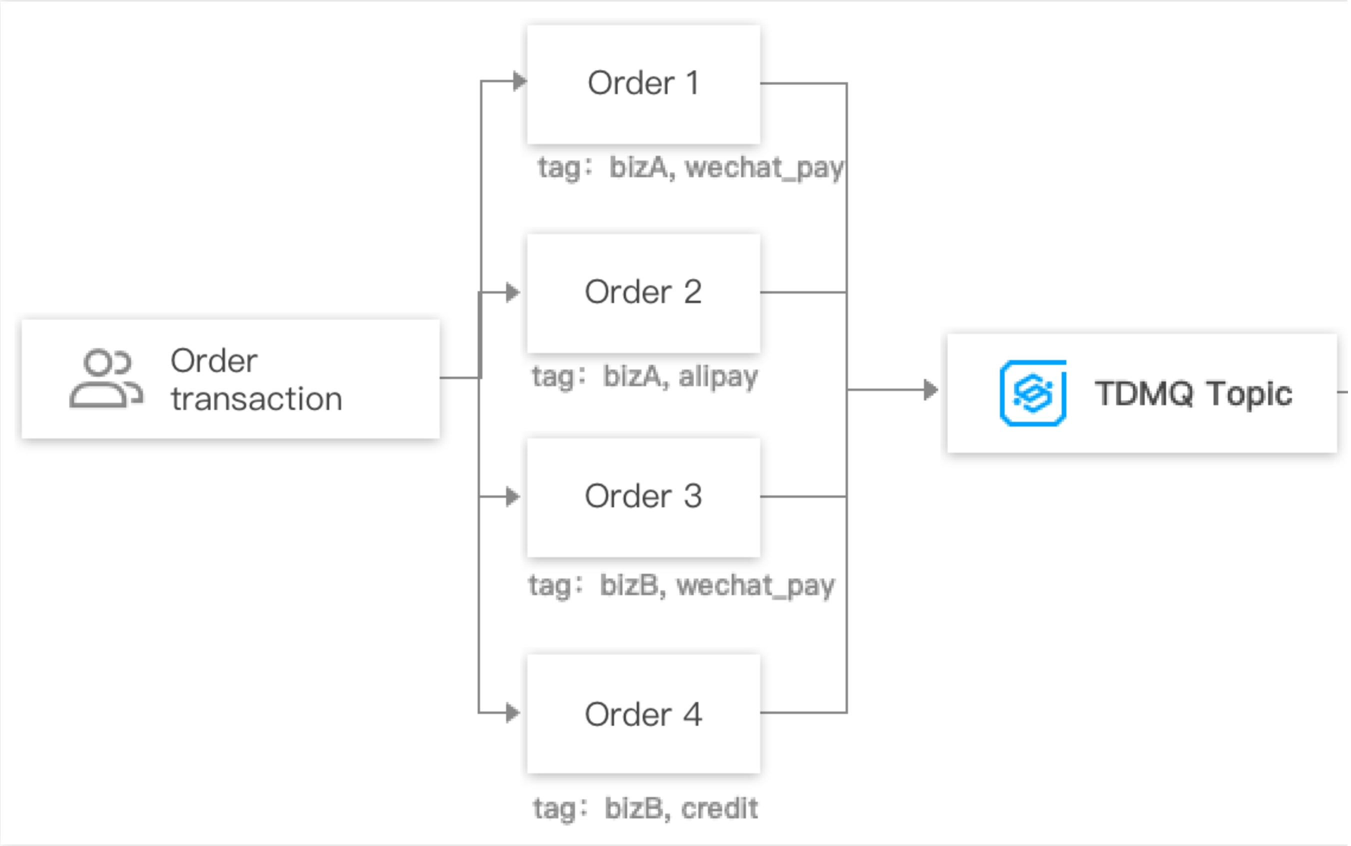
Message filtering indicates that messages are filtered by the message attribute configured by the message producer when the producer sends messages to the topic. The consumer that subscribes to the topic can filter messages based on their attributes so that only eligible messages are delivered to the consumer for consumption.

If a consumer configures no filter conditions when subscribing to a topic, no matter whether filter attributes are configured 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. Users can set one or more tags during message production and subscribe to specified tags during consumption.



Instructions

Filtering by Tag

Sending Messages

Note:

During message sending, tags must be clearly specified for each message.

```
String tag = "yourMessageTagA";
       final Message message = provider.newMessageBuilder()
           // Set topic for the current message.
           .setTopic(topic)
           // Message secondary classifier of message besides topic.
           .setTag(tag)
           // Key(s) of the message, another way to mark message besides message i
           .setKeys("yourMessageKey-1c151062f96e")
           .setBody(body)
           .build();
```

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.

```
String consumerGroup = "yourConsumerGroup";
        String topic = "yourTopic";
        String tag = "*";
        FilterExpression filterExpression = new FilterExpression(tag, FilterExpress
        // In most case, you don't need to create too many consumers, singleton pat
```

```
          PushConsumer pushConsumer = provider.newPushConsumerBuilder()
              .setClientConfiguration(clientConfiguration)
              // Set the consumer group name.
              .setConsumerGroup(consumerGroup)
              // Set the subscription for the consumer.
              .setSubscriptionExpressions(Collections.singletonMap(topic, filterExpre
              .setMessageListener(messageView -> {
                  // Handle the received message and return consume result.
                  log.info("Consume message={}", messageView);
                  return ConsumeResult.SUCCESS;
              })
              .build();
```

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.

```
String consumerGroup = "yourConsumerGroup";
      String topic = "yourTopic";
      String tag = "TAGA";
      FilterExpression filterExpression = new FilterExpression(tag, FilterExpress
      // In most case, you don't need to create too many consumers, singleton pat
      PushConsumer pushConsumer = provider.newPushConsumerBuilder()
          .setClientConfiguration(clientConfiguration)
          // Set the consumer group name.
          .setConsumerGroup(consumerGroup)
          // Set the subscription for the consumer.
          .setSubscriptionExpressions(Collections.singletonMap(topic, filterExpre
          .setMessageListener(messageView -> {
              // Handle the received message and return consume result.
              log.info("Consume message={}", messageView);
              return ConsumeResult.SUCCESS;
          })
          .build();
```

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.

```
 String consumerGroup = "yourConsumerGroup";
        String topic = "yourTopic";
        String tag = "TAGA || TAGB";
        FilterExpression filterExpression = new FilterExpression(tag, FilterExpress
        // In most case, you don't need to create too many consumers, singleton pat
        PushConsumer pushConsumer = provider.newPushConsumerBuilder()
            .setClientConfiguration(clientConfiguration)
            // Set the consumer group name.
            .setConsumerGroup(consumerGroup)
            // Set the subscription for the consumer.
            .setSubscriptionExpressions(Collections.singletonMap(topic, filterExpre
            .setMessageListener(messageView -> {
                // Handle the received message and return consume result.
                log.info("Consume message={}", messageView);
                return ConsumeResult.SUCCESS;
            })
            .build();
```

Filtering by SQL

Sending Messages

The message sending code here is basically the same as the code for sending simple messages. A message is allowed to carry multiple user-defined attributes when you construct the message body.

```
final Message message = provider.newMessageBuilder()
            // Set topic for the current message.
            .setTopic(topic)
            // Message secondary classifier of message besides topic.
            // Key(s) of the message, another way to mark message besides message i
            .setKeys("yourMessageKey-1c151062f96e")
            .setBody(body)
            // Some information for SQL filtering
            .addProperty("key1", "value1")
            .build();
```

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.

```
String consumerGroup = "yourConsumerGroup";
      String topic = "yourTopic";
      String sql = "key1 IS NOT NULL AND key1='value1'";
      // SQL expression
      FilterExpression filterExpression = new FilterExpression(sql, FilterExpress
      // If all is subscribed to
      //FilterExpression filterExpression = FilterExpression.SUB_ALL;
```

```
        // In most case, you don't need to create too many consumers, singleton pat
        PushConsumer pushConsumer = provider.newPushConsumerBuilder()
            .setClientConfiguration(clientConfiguration)
            // Set the consumer group name.
            .setConsumerGroup(consumerGroup)
            // Set the subscription for the consumer.
            .setSubscriptionExpressions(Collections.singletonMap(topic, filterExpre
            .setMessageListener(messageView -> {
                // Handle the received message and return consume result.
                log.info("Consume message={}", messageView);
                return ConsumeResult.SUCCESS;
            })
            .build();
```

Note:

The preceding sections provide introduction to the use instructions of message publishing and subscription. For more operations, see GitHub Demo or RocketMQ Official Documentation.

Message Retry

Last updated：2024-01-17 16:53:20

This document describes the mechanism of message retry and its usage in TDMQ for RocketMQ.

Feature Description

When a message is consumed for the first time by a consumer and does not receive a normal response, or when users request the server to deliver it again, TDMQ for RocketMQ will automatically attempt to deliver this message again 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. In this situation, human intervention is generally required. You can write a dedicated client to subscribe to the dead letter queue to process such failed messages.

Note:

The broker will automatically retry in the cluster consumption mode but not the broadcasting consumption mode.

The following results are considered as consumption failure, and the message will be retried accordingly:

The consumer returns ConsumeResult.FAILURE.

The consumer returns null.

The consumer actively/passively throws an exception.

Maximum 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

No special processing is required, the 5.0 SDK follows the preceding rules.

Dead Letter Message

Last updated：2024-01-17 16:53:55

This document describes the dead letter queue in TDMQ for RocketMQ and how to use it.

Feature Description

When a message is consumed for the first time by a consumer and does not receive a normal response, or when users request the server to deliver it again, TDMQ for RocketMQ will automatically attempt to deliver this message again 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.

If messages enter the dead letter queue, TDMQ for RocketMQ can no longer automatically process them. In this situation, human intervention is required. You can confirm it via message export, or via specified message resending on the control console.

Feature Description

Different from the retry queue, which supports automatic consumption, messages within the dead letter queue require manual intervention.

The validity of messages also adheres to the rule of deletion after three days by default.

The dead letter queue starts with `%DLQ%` and corresponds to the consumer group one by one. Therefore, a dead letter queue contains all dead letter messages corresponding to the group ID, regardless of their originating topic.

Consumption Mode

Cluster Consumption

Last updated：2024-01-17 16:54:18

Introduction to Cluster Consumption Mode

When the cluster consumption mode is used, any message only needs to be processed by just one consumer within the same subscription group.

Use Cases

This is suitable to scenarios where each message only needs to be processed once.

How to Use

The 5.0 SDK uses the cluster consumption mode by default, requiring no special configuration.

Note:

Please ensure consistency in the subscription relationships of all consumer instances under the same group ID.

Broadcast Consumption

Last updated：2024-01-17 16:54:51

Introduction to Broadcasting Consumption Mode

When the broadcasting consumption mode is used, each message is pushed to all registered consumers within the cluster, ensuring that each message is consumed at least once by every consumer.

Use Cases

This mode is suitable for scenarios where every message needs to be processed by every consumer within the cluster.

How to Use

The 5.0 SDK no longer supports broadcasting consumption. However, a similar feature can be achieved by creating a distinct subscription group for every consumer if it is required.

Note:

Please ensure consistency in the subscription relationships of all consumer instances under the same group ID.

SDK Documentation

Compatibility Description

Last updated：2024-01-17 16:55:18

RocketMQ 5.0 introduces a brand-new 5.x SDK based on the gRPC protocol. The new SDK version offers a more lightweight framework and better multilingual support. We highly recommend that you use this version. Moreover, TDMQ for RocketMQ 5.x series continues to support the access through the 4.x SDK for existing businesses. The compatibility details are as follows:

Server Version	Client Version		Compatibility
5.x	5.x SDK		Fully compatible
	4.x SDK	Versions of 4.9.5 or later	PushConsumer does not support the broadcasting consumption mode yet PushConsumer CONSUME_FROM_TIMESTAMP is currently ineffective (The offset can be reset on the console).
		Versions earlier than 4.9.5	PushConsumer does not support the broadcasting consumption mode yet PushConsumer CONSUME_FROM_TIMESTAMP is currently ineffective (The offset can be reset on the console). PullConsumer consumption is not supported yet.

5.x SDK

Use of Java SDK

Last updated：2024-01-17 16:55:30

Overview

This document describes how to use an open-source SDK to send and receive messages with the SDK for Java serving as example, for you to better understand the complete procedure involved in message sending and receiving.

Note:

The Java client is used as an example. For clients of other languages, see the SDK Documentation.

Prerequisites

You have created and prepared the required resources.

You have installed JDK 1.8 or later.

You have installed Maven 2.5 or later.

You have downloaded the demo.

Directions:

Step 1: Installing the Java Dependency Library

Incorporate the relevant dependencies in the Java project. A Maven project is used as an example. Add the following dependencies to pom.xml:

```
<dependencies>
        <dependency>
            <groupId>org.apache.rocketmq</groupId>
            <artifactId>rocketmq-client-java</artifactId>
            <version>5.0.5</version>
        </dependency>
</dependencies>
```

Step 2: Producing Messages

```
public class NormalMessageSyncProducer {
    private static final Logger log = LoggerFactory.getLogger(NormalMessageSyncProd

    private NormalMessageSyncProducer() {
    }

    public static void main(String[] args) throws ClientException, IOException {
        final ClientServiceProvider provider = ClientServiceProvider.loadService();

        //Adding ak and sk in the configuration
        String accessKey = "yourAccessKey"; //ak
        String secretKey = "yourSecretKey"; //sk
        SessionCredentialsProvider sessionCredentialsProvider =
            new StaticSessionCredentialsProvider(accessKey, secretKey);

        // Fill in the access location provided by Tencent Cloud
        String endpoints = "rmq-xxx.rocketmq.xxxtencenttdmq.com:8080";
        ClientConfiguration clientConfiguration = ClientConfiguration.newBuilder()
            .setEndpoints(endpoints)
            .enableSsl(false)
            .setCredentialProvider(sessionCredentialsProvider)
            .build();
        String topic = "yourNormalTopic";
        // In most case, you don't need to create too many producers, singleton pat
        final Producer producer = provider.newProducerBuilder()
            .setClientConfiguration(clientConfiguration)
            // Set the topic name(s), which is optional but recommended. It makes p
            // route before message publishing.
            .setTopics(topic)
            // May throw {@link ClientException} if the producer is not initialized
            .build();
        // Define your message body.
        byte[] body = "This is a normal message for Apache RocketMQ".getBytes(Stand
        String tag = "yourMessageTagA";
        final Message message = provider.newMessageBuilder()
            // Set topic for the current message.
            .setTopic(topic)
            // Message secondary classifier of message besides topic.
            .setTag(tag)
            // Key(s) of the message, another way to mark message besides message i
            .setKeys("yourMessageKey-1c151062f96e")
            .setBody(body)
            .build();
        try {
            final SendReceipt sendReceipt = producer.send(message);
            log.info("Send message successfully, messageId={}", sendReceipt.getMess
        } catch (Throwable t) {
            log.error("Failed to send message", t);
        }
        // Close the producer when you don't need it anymore.
        producer.close();
    }
}
```

Step 3: Consuming Messages

TDMQ for RocketMQ 5.x series by Tencent Cloud supports two types of clients: Push Consumer and Simple Consumer.

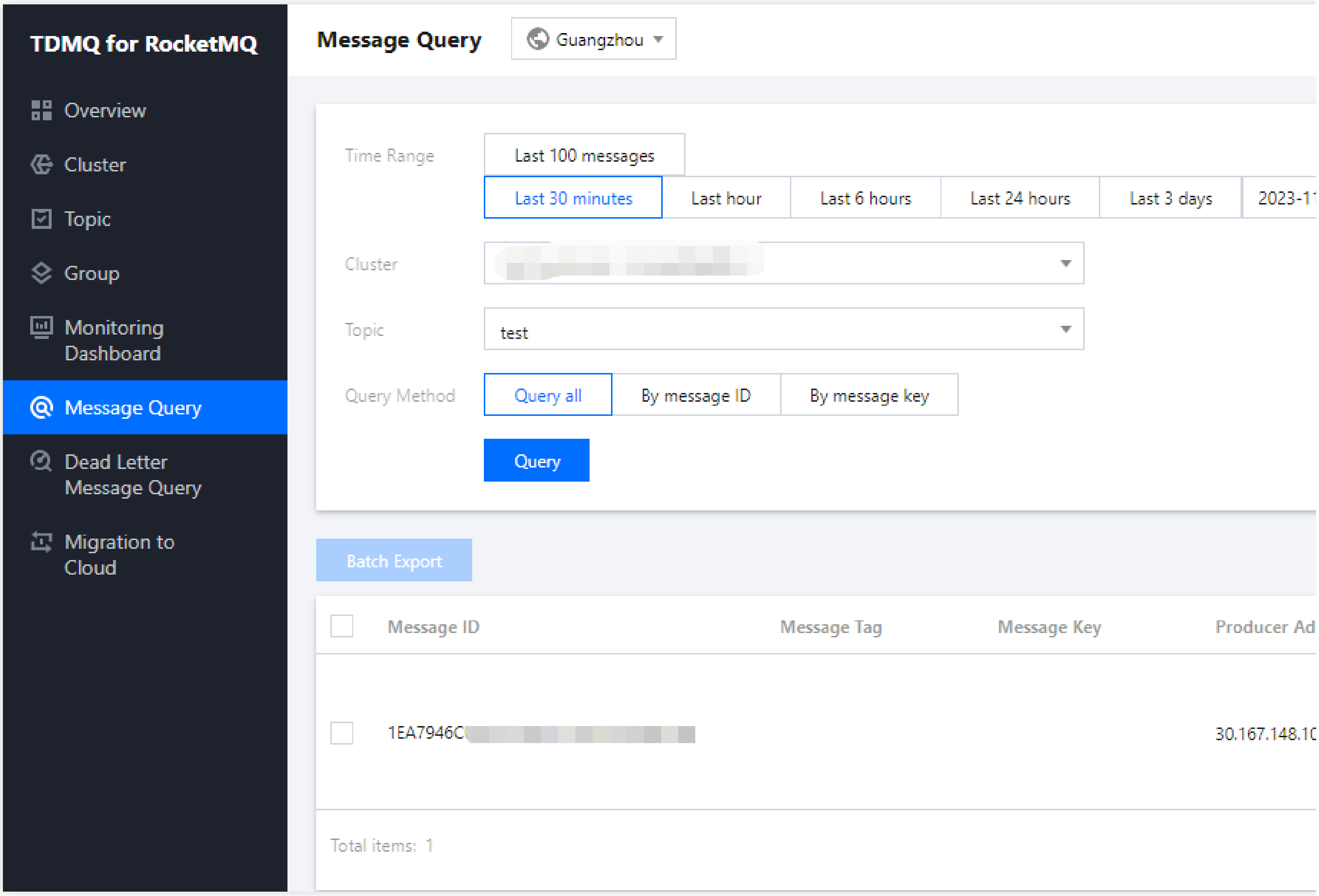
The following code is an example based on Push Consumer:

```
public class NormalPushConsumer {
    private static final Logger log = LoggerFactory.getLogger(NormalPushConsumer.cl

    private NormalPushConsumer() {
    }

    public static void main(String[] args) throws ClientException, IOException, Int
```

```
        final ClientServiceProvider provider = ClientServiceProvider.loadService();

        //Adding ak and sk in the configuration
        String accessKey = "yourAccessKey"; //ak
        String secretKey = "yourSecretKey"; //sk
        SessionCredentialsProvider sessionCredentialsProvider =
            new StaticSessionCredentialsProvider(accessKey, secretKey);

        // Fill in the access location provided by Tencent Cloud
        String endpoints = "rmq-xxx.rocketmq.xxxtencenttdmq.com:8080";
        ClientConfiguration clientConfiguration = ClientConfiguration.newBuilder()
            .setEndpoints(endpoints)
            .enableSsl(false)
            .setCredentialProvider(sessionCredentialsProvider)
            .build();
        String tag = "*";
        FilterExpression filterExpression = new FilterExpression(tag, FilterExpress
        String consumerGroup = "yourConsumerGroup";
        String topic = "yourTopic";
        // In most case, you don't need to create too many consumers, singleton pat
        PushConsumer pushConsumer = provider.newPushConsumerBuilder()
            .setClientConfiguration(clientConfiguration)
            // Set the consumer group name.
            .setConsumerGroup(consumerGroup)
            // Set the subscription for the consumer.
            .setSubscriptionExpressions(Collections.singletonMap(topic, filterExpre
            .setMessageListener(messageView -> {
                // Handle the received message and return consume result.
                log.info("Consume message={}", messageView);
                return ConsumeResult.SUCCESS;
            })
            .build();
        // Block the main thread, no need for production environment.
        Thread.sleep(Long.MAX_VALUE);
        // Close the push consumer when you don't need it anymore.
        pushConsumer.close();
    }
}
```

Step 4: Viewing Message Details

After the message is sent, you will receive a message ID (messageID). Developers can query the recently sent messages on the Message Query page, as shown in the following figure. Information such as details and traces for specific messages is also available. For details, see Message Query.



Message Query

Time Range: Last 30 minutes

Cluster: [redacted]

Topic: test

Query Method: Query all

□	Message ID	Message Tag	Message Key	Producer Ad
□	1EA7946C[redacted]			30.167.148.1C

Total items: 1

Use of Go SDK

Last updated：2024-01-17 16:55:52

Overview

This document describes how to use an open-source SDK to send and receive messages with the Golang SDK serving as example, for you to better understand the complete process of message sending and receiving.

Note:

The Golang client is used as an example. For clients of other languages, see the SDK Documentation.

Prerequisites

You have created and prepared the required resources.

You have installed Golang version 1.13 or higher.

You have downloaded the demo.

Directions:

Step 1: Installing the Golang Dependency Library

Incorporate the relevant dependencies in the Golang project. `go get` is used as example. Run the following command:

```
go get github.com/apache/rocketmq-clients/golang/v5
```

Step 2: Producing Messages

```
package main

import (
    "context"
    "fmt"
    "log"
    "os"
    "strconv"
    "time"

    rmq_client "github.com/apache/rocketmq-clients/golang/v5"
    "github.com/apache/rocketmq-clients/golang/v5/credentials"
)

const (
    Topic = "xxxxxx"
    // Set Endpoint to the access address provided by Tencent Cloud
    Endpoint = "xxxxxx"
    // Add the configured ak to AccessKey
    AccessKey = "xxxxxx"
    // Add the configured sk to SecretKey
    SecretKey = "xxxxxx"
)

func main() {
    os.Setenv("mq.consoleAppender.enabled", "true")
    rmq_client.ResetLogger()
    // In most case, you don't need to create many producers, singleton pattern is
    producer, err := rmq_client.NewProducer(&rmq_client.Config{
        Endpoint: Endpoint,
        Credentials: &credentials.SessionCredentials{
            AccessKey:    AccessKey,
            AccessSecret: SecretKey,
        },
    },
        rmq_client.WithTopics(Topic),
    )
    if err != nil {
        log.Fatal(err)
    }
    // start producer
    err = producer.Start()
    if err != nil {
        log.Fatal(err)
    }
    // graceful stop producer
    defer producer.GracefulStop()

    for i := 0; i < 10; i++ {
        // new a message
        msg := &rmq_client.Message{
             Topic,
            Body:  []byte("this is a message : " + strconv.Itoa(i)),
        }
        // set keys and tag
        msg.SetKeys("a", "b")
        msg.SetTag("ab")
        // send message in sync
        resp, err := producer.Send(context.TODO(), msg)
        if err != nil {
            log.Fatal(err)
        }
        for i := 0; i < len(resp); i++ {
            fmt.Printf("%#v\\n", resp[i])
        }
        // wait a moment
        time.Sleep(time.Second * 1)
    }
}
```

Step 3: Consuming Messages

TDMQ for RocketMQ 5.x series by Tencent Cloud supports two types of clients: Push Consumer and Simple Consumer.

Note:

At this time, the community version of the Golang SDK only supports Simple Consumer.

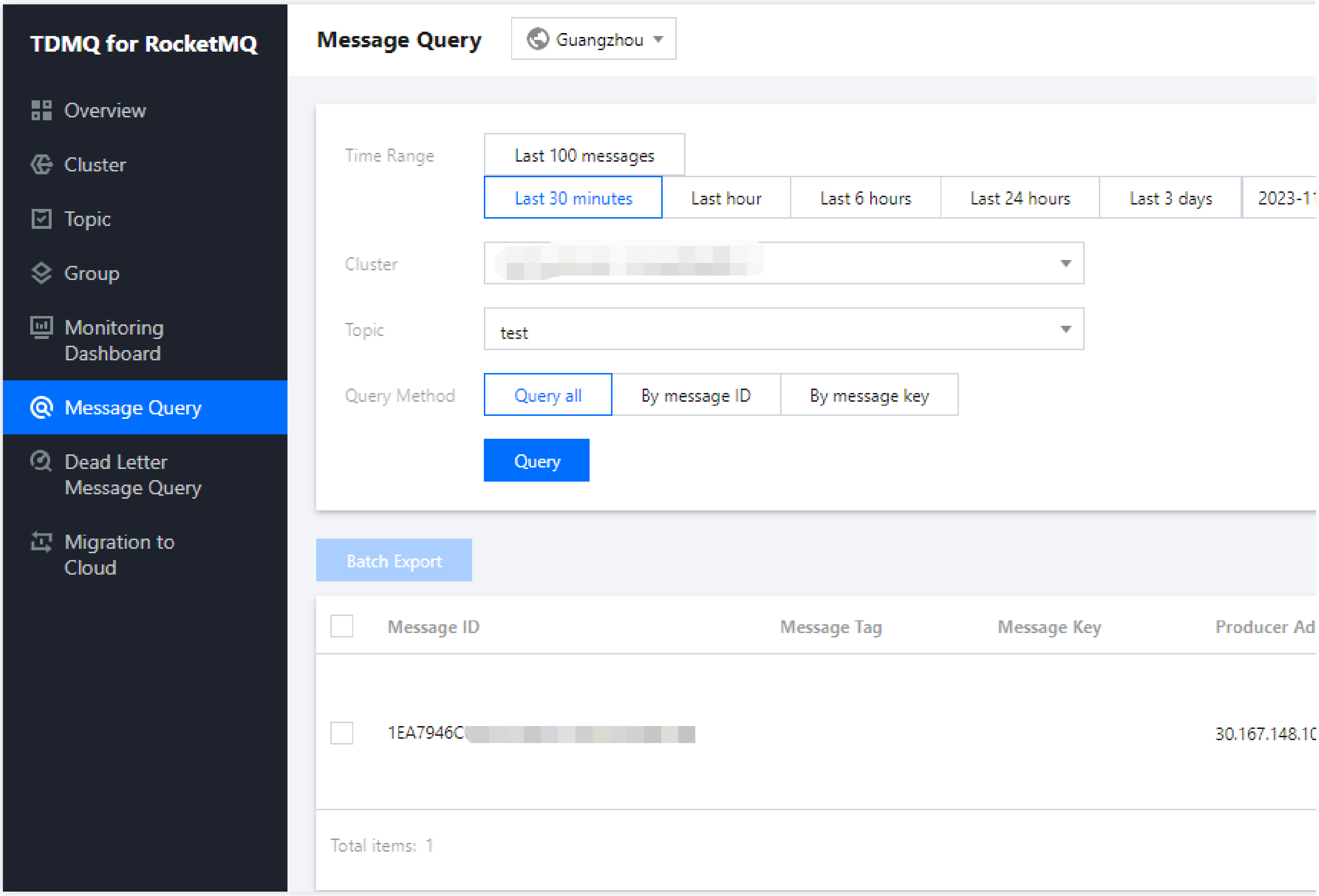
The following sample code uses Simple Consumer as an example:

```
package main

import (
    "context"
    "fmt"
    "log"
    "os"
    "time"

    rmq_client "github.com/apache/rocketmq-clients/golang/v5"
    "github.com/apache/rocketmq-clients/golang/v5/credentials"
```

```
)

const (
    Topic         = "xxxxxx"
    ConsumerGroup = "xxxxxx"
    // Set Endpoint to the access address provided by Tencent Cloud
    Endpoint = "xxxxxx"
    // Add the configured ak to AccessKey
    AccessKey = "xxxxxx"
    // Add the configured sk to SecretKey
    SecretKey = "xxxxxx"
)

var (
    // maximum waiting time for receive func
    awaitDuration = time.Second * 5
    // maximum number of messages received at one time
    maxMessageNum int32 = 16
    // invisibleDuration should > 20s
    invisibleDuration = time.Second * 20
    // receive messages in a loop
)

func main() {
    // log to console
    os.Setenv("mq.consoleAppender.enabled", "true")
    rmq_client.ResetLogger()
    // In most case, you don't need to create many consumers, singleton pattern is
    simpleConsumer, err := rmq_client.NewSimpleConsumer(&rmq_client.Config{
        Endpoint:      Endpoint,
        ConsumerGroup: ConsumerGroup,
        Credentials: &credentials.SessionCredentials{
            AccessKey:    AccessKey,
            AccessSecret: SecretKey,
        },
    },
        rmq_client.WithAwaitDuration(awaitDuration),
        rmq_client.WithSubscriptionExpressions(map[string]*rmq_client.FilterExpress
            Topic: rmq_client.SUB_ALL,
        }),
    )
    if err != nil {
        log.Fatal(err)
    }
    // start simpleConsumer
    err = simpleConsumer.Start()
    if err != nil {
        log.Fatal(err)
    }
    // graceful stop simpleConsumer
    defer simpleConsumer.GracefulStop()
    for {
        fmt.Println("start receive message")
        mvs, err := simpleConsumer.Receive(context.TODO(), maxMessageNum, invisible
        if err != nil {
            fmt.Println(err)
        }
        // ack message
        for _, mv := range mvs {
            simpleConsumer.Ack(context.TODO(), mv)
            fmt.Println(mv)
        }
        fmt.Println("wait a moment")
        fmt.Println()
        time.Sleep(time.Second * 1)
    }
}
```

Step 4: Viewing Message Details

After the message is sent, you will receive a message ID (messageID). Developers can query the recently sent messages on the Message Query page, as shown in the following figure. Information such as details and traces for specific messages is also available. For details, see Message Query.



Time Range: Last 30 minutes

Cluster: [redacted]

Topic: test

Query Method: Query all

☐	Message ID	Message Tag	Message Key	Producer Ad
☐	1EA7946C[redacted]			30.167.148.1C

Total items: 1

Use of C++ SDK

Last updated：2024-01-17 16:56:06

Overview

This document describes how to use an open-source SDK to send and receive messages with the SDK for C++ serving as example, for you to better understand the complete procedure involved in message sending and receiving.

Note:

The C++ client is used as an example. For clients of other languages, see the SDK Documentation.

Prerequisites

You have created and prepared the required resources.

You have installed a compiler suite supporting C++11.

You have installed Bazel version 5.2.0 or CMake version 3.13 and later.

If you use CMake for compilation, gRPC version 1.46.3 is recommended due to incompatibilities between higher versions and the SDK.

You have downloaded the demo.

Directions:

Step 1: Installing the SDK for C++

Install the SDK.

Note:

TDMQ for RocketMQ 5.x series does not currently support TLS. A patch must be applied.

Step 2: Producing Messages

```
#include <algorithm>
#include <atomic>
#include <iostream>
#include <memory>
#include <random>
#include <string>
#include <system_error>

#include "rocketmq/CredentialsProvider.h"
#include "rocketmq/Logger.h"
#include "rocketmq/Message.h"
```

```
#include "rocketmq/Producer.h"

using namespace ROCKETMQ_NAMESPACE;

const std::string &alphaNumeric() {
    static std::string alpha_numeric("0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHI
    return alpha_numeric;
}

std::string randomString(std::string::size_type len) {
    std::string result;
    result.reserve(len);
    std::random_device rd;
    std::mt19937 generator(rd());
    std::string source(alphaNumeric());
    std::string::size_type generated = 0;
    while (generated < len) {
        std::shuffle(source.begin(), source.end(), generator);
        std::string::size_type delta = std::min({len - generated, source.length()})
        result.append(source.substr(0, delta));
        generated += delta;
    }
    return result;
}

static const std::string topic = "xxx";
// Enter the access address provided by Tencent Cloud
static const std::string access_point = "rmq-xxx.rocketmq.xxxtencenttdmq.com:8081";
// Add the configured ak and sk
static const std::string access_key = "xxx";
static const std::string access_secret = "xxx";
static const uint32_t total = 32;
static const int32_t message_body_size = 128;


int main(int argc, char *argv[]) {
    CredentialsProviderPtr credentials_provider;
    if (!access_key.empty() && !access_secret.empty()) {
        credentials_provider = std::make_shared<StaticCredentialsProvider>(access_k
    }

    // In most case, you don't need to create too many producers, singletion patter
    auto producer = Producer::newBuilder()
            .withConfiguration(Configuration::newBuilder()
                                       .withEndpoints(access_point)
                                       .withCredentialsProvider(credentials_provide
                                       .withSsl(false)
                                       .build())
            .withTopics({topic})
            .build();

    std::atomic_bool stopped;
    std::atomic_long count(0);

    auto stats_lambda = [&] {
        while (!stopped.load(std::memory_order_relaxed)) {
            long cnt = count.load(std::memory_order_relaxed);
            while (count.compare_exchange_weak(cnt, 0)) {
                break;
            }
            std::this_thread::sleep_for(std::chrono::seconds(1));
            std::cout << "QPS: " << cnt << std::endl;
        }
    };

    std::thread stats_thread(stats_lambda);

    std::string body = randomString(message_body_size);

    try {
        for (std::size_t i = 0; i < total; ++i) {
            auto message = Message::newBuilder()
                    .withTopic(topic)
                    .withTag("TagA")
                    .withKeys({"Key-" + std::to_string(i)})
                    .withBody(body)
                    .build();
            std::error_code ec;
            SendReceipt send_receipt = producer.send(std::move(message), ec);
            if (ec) {
                std::cerr << "Failed to publish message to " << topic << ". Cause:
            } else {
                std::cout << "Publish message to " << topic << " OK. Message-ID: "
                          << std::endl;
                count++;
            }
        }
    } catch (...) {
        std::cerr << "Ah...No!!!" << std::endl;
    }
    stopped.store(true, std::memory_order_relaxed);
    if (stats_thread.joinable()) {
        stats_thread.join();
    }
    return EXIT_SUCCESS;
}
```

Step 3: Consuming Messages

TDMQ for RocketMQ 5.x series of Tencent Cloud supports two consumption modes: Push Consumer and Simple Consumer.

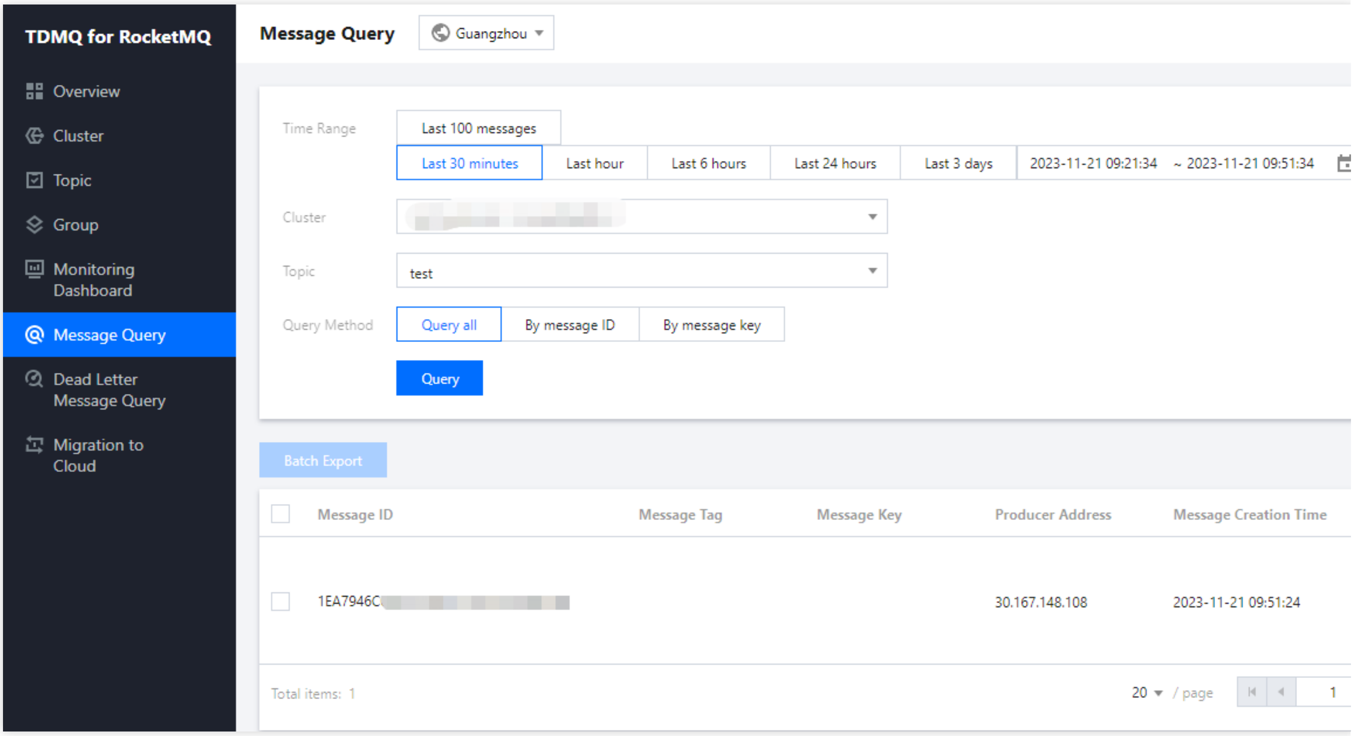
The following code is an example based on Push Consumer:

```
#include <chrono>
```

```
#include <iostream>
#include <mutex>
#include <thread>

#include "rocketmq/Logger.h"
#include "rocketmq/PushConsumer.h"

using namespace ROCKETMQ_NAMESPACE;

static const std::string topic = "xxx";
// Enter the access address provided by Tencent Cloud
static const std::string access_point = "rmq-xxx.rocketmq.xxxtencenttdmq.com:8081";
// Add the configured ak and sk
static const std::string access_key = "xxx";
static const std::string access_secret = "xxx";
static const std::string group = "group-xxx";

int main(int argc, char *argv[]) {
    auto &logger = getLogger();
    logger.setConsoleLevel(Level::Info);
    logger.setLevel(Level::Info);
    logger.init();

    std::string tag = "*";

    auto listener = [](const Message &message) {
        std::cout << "Received a message[topic=" << message.topic() << ", MsgId=" <
        return ConsumeResult::SUCCESS;
    };

    CredentialsProviderPtr credentials_provider;
    if (!access_key.empty() && !access_secret.empty()) {
        credentials_provider = std::make_shared<StaticCredentialsProvider>(access_k
    }

    // In most case, you don't need to create too many consumers, singletion patter
    auto push_consumer = PushConsumer::newBuilder()
            .withGroup(group)
            .withConfiguration(Configuration::newBuilder()
                                       .withEndpoints(access_point)
                                       .withRequestTimeout(std::chrono::seconds(3))
                                       .withCredentialsProvider(credentials_provide
                                       .withSsl(false)
                                       .build())
            .withConsumeThreads(4)
            .withListener(listener)
            .subscribe(topic, tag)
            .build();

    std::this_thread::sleep_for(std::chrono::minutes(30));

    return EXIT_SUCCESS;
}
```

Step 4: Viewing Message Details

After the message is sent, you will receive a message ID (messageID). Developers can query the recently sent messages on the Message Query page, as shown in the following figure. Information such as details and traces for specific messages is also available. For details, see Message Query.



Time Range: Last 30 minutes 2023-11-21 09:21:34 ~ 2023-11-21 09:51:34

Cluster: [redacted]

Topic: test

Query Method: Query all

☐	Message ID	Message Tag	Message Key	Producer Address	Message Creation Time
☐	1EA7946C[redacted]			30.167.148.108	2023-11-21 09:51:24

Total items: 1 20 / page 1

4.x SDK

Use of Java SDK

Last updated：2024-01-17 16:56:33

Overview

This document describes how to use an open-source SDK to send and receive messages with the SDK for Java 4.0 serving as example, for you to better understand the complete procedure involved in message sending and receiving.

Note:

The Java client is used as an example. For clients of other languages, see the SDK Documentation.

Prerequisites

You have created and prepared the required resources.

You have installed JDK 1.8 or later.

You have installed Maven 2.5 or later.

You have downloaded the demo.

Directions:

Step 1: Installing the Java Dependency Library

Incorporate the relevant dependencies in the Java project. A Maven project is used as an example. Add the following dependencies to pom.xml:

```
<!-- in your <dependencies> block -->
  <dependency>
      <groupId>org.apache.rocketmq</groupId>
      <artifactId>rocketmq-client</artifactId>
      <version>4.9.7</version>
  </dependency>

  <dependency>
      <groupId>org.apache.rocketmq</groupId>
      <artifactId>rocketmq-acl</artifactId>
      <version>4.9.7</version>
```

```
    </dependency>
```

Step 2: Producing Messages

```
// Instantiate the message producer
   DefaultMQProducer producer = new DefaultMQProducer(
       groupName,
       new AclClientRPCHook(new SessionCredentials(accessKey, secretKey)) // ACL pe
   );
   // Set the NameServer's address. The address is in the format of an access addre
```

```
    producer.setNamesrvAddr(nameserver);
    // Start the producer instance.
    producer.start();

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

Step 3: Consuming Messages

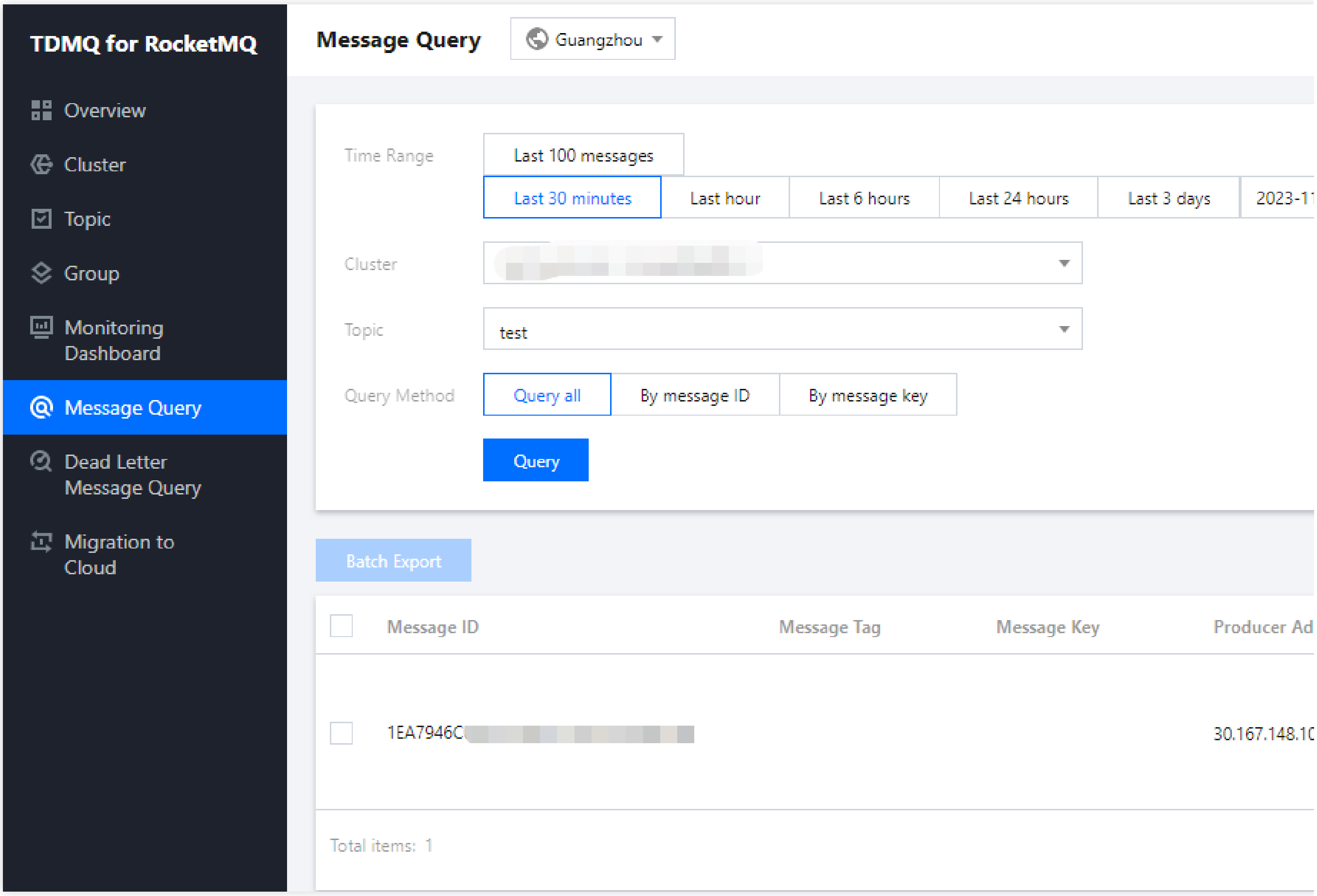
The following code sample uses Push Consumer as example. For other codes, see the more detailed 4.x documentation.

```
// Instantiate the consumer
DefaultMQPushConsumer pushConsumer = new DefaultMQPushConsumer(
            groupName,
            new AclClientRPCHook(new SessionCredentials(accessKey, secretKey))); //
        // Set the NameServer address
pushConsumer.setNamesrvAddr(nameserver);
        // Subscribe to a topic
pushConsumer.subscribe(topic_name, "*");
      // Register a callback implementation class to process messages pulled from t
pushConsumer.registerMessageListener((MessageListenerConcurrently) (msgs, context)
          // Message processing logic
```

```
            System.out.printf("%s Receive New Messages: %s %n", Thread.currentThread(
            // Mark the message as successfully consumed, and return consumption stat
            return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
        });
        // Start the consumer instance
pushConsumer.start();
```

Step 4: Viewing Message Details

After the message is sent, you will receive a message ID (messageID). Developers can query the recently sent messages on the Message Query page, as shown in the following figure. Information such as details and traces for specific messages is also available. For details, see Message Query section.



Time Range: Last 30 minutes

Cluster: [redacted]

Topic: test

Query Method: Query all

□	Message ID	Message Tag	Message Key	Producer Ad
□	1EA7946C[redacted]			30.167.148.1C

Total items: 1

Use of Go SDK

Last updated：2024-01-17 16:56:46

Overview

This document describes how to use an open-source SDK to send and receive messages with the Golang SDK serving as example, for you to better understand the complete process of message sending and receiving.

Prerequisites

You have created the required resources.

You have installed Go.

You have downloaded the demo.

Directions:

1. Execute the following command in the client environment to download the relevant 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 standard messages, modify the corresponding parameters in the `syncSendMessage.go` file.

Currently, delayed messages support arbitrary precision delay, unaffected by the delay level.

General message

Delayed Messages

```
// Service access address (Note: http:// or https:// must be appended before the ac
   var serverAddress = "https://rocketmq-xxx.rocketmq.ap-bj.public.tencenttdmq.com:
   // Authorize the role name
   var secretKey = "admin"
   // Authorize the key for the role
   var accessKey = "eyJrZXlJZC...."
   // 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 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{"http://r
        // Set ACL permissions
        producer.WithCredentials(primitive.Credentials{
            SecretKey: "admin",
            AccessKey: "eyJrZXlJZC......",
        }),
        // Set the producer group
        producer.WithGroupName(groupName),

        // 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 Client! Th
        // Specify the delay level
        // Relationship between level and time:
        // 1s, 5s, 10s, 30s, 1m, 2m, 3m, 4m, 5m, 6m, 7m, 8m, 9m, 10m, 20m, 30m, 1h,
        // 1    2    3     4     5    6   7    8   9   10   11   12  13   14    15
        // If you want to use the delay level, set the following method

        msg.WithDelayTimeLevel(3)
        // If you want to use arbitrary delay messages, set the following method an
        delayMills := int64(10 * 1000)
        msg.WithProperty("__STARTDELIVERTIME", strconv.FormatInt(time.Now().Unix()+
        // 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())
    }
```

3. Message sending is the same as above (taking the synchronous sending as an example).

```
// Topic name
   var topicName = "topic1"
   // Construct message content
   msg := &primitive.Message{
       Topic: topicName, // Set the topic name
       Body:  []byte("Hello RocketMQ Go Client! This is a new message."),
   }
   // Set the tag
   msg.WithTag("TAG")
   // Set the key
   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 Topic tab on the Cluster page on the console.
TAG	Message tag identifier.
yourKey	Business message key.

Release the resources.

```
// Shut down the producer
   err = p.Shutdown()
   if err != nil {
       fmt.Printf("shutdown producer error: %s", err.Error())
   }
```

Note:

For more information on asynchronous sending and one-way sending, see the demo or RocketMQ-Client-Go Examples.

4. Create a consumer.

```
// Service access address (Note: http:// or https:// must be appended before the ac
    var serverAddress = "http://rocketmq-xxx.rocketmq.ap-bj.public.tencenttdmq.com:8
    // Authorize the role name
    var secretKey = "admin"
    // Authorize the key for the role
    var accessKey = "eyJrZXlJZC...."
    // 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 consumption from the start offset
        consumer.WithConsumeFromWhere(consumer.ConsumeFromFirstOffset),
        // Set the consumption mode (cluster mode 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)
    }
```

5. Consume the message.

```
// Topic name
   var topicName = "topic1"
   // Set the tag of messages that are subscribed to
   selector := consumer.MessageSelector{
       Type:       consumer.TAG,
       Expression: "TagA || TagC",
   }
   // Define the delay level for retrying consumption. There are 18 delay levels in
   // 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
```

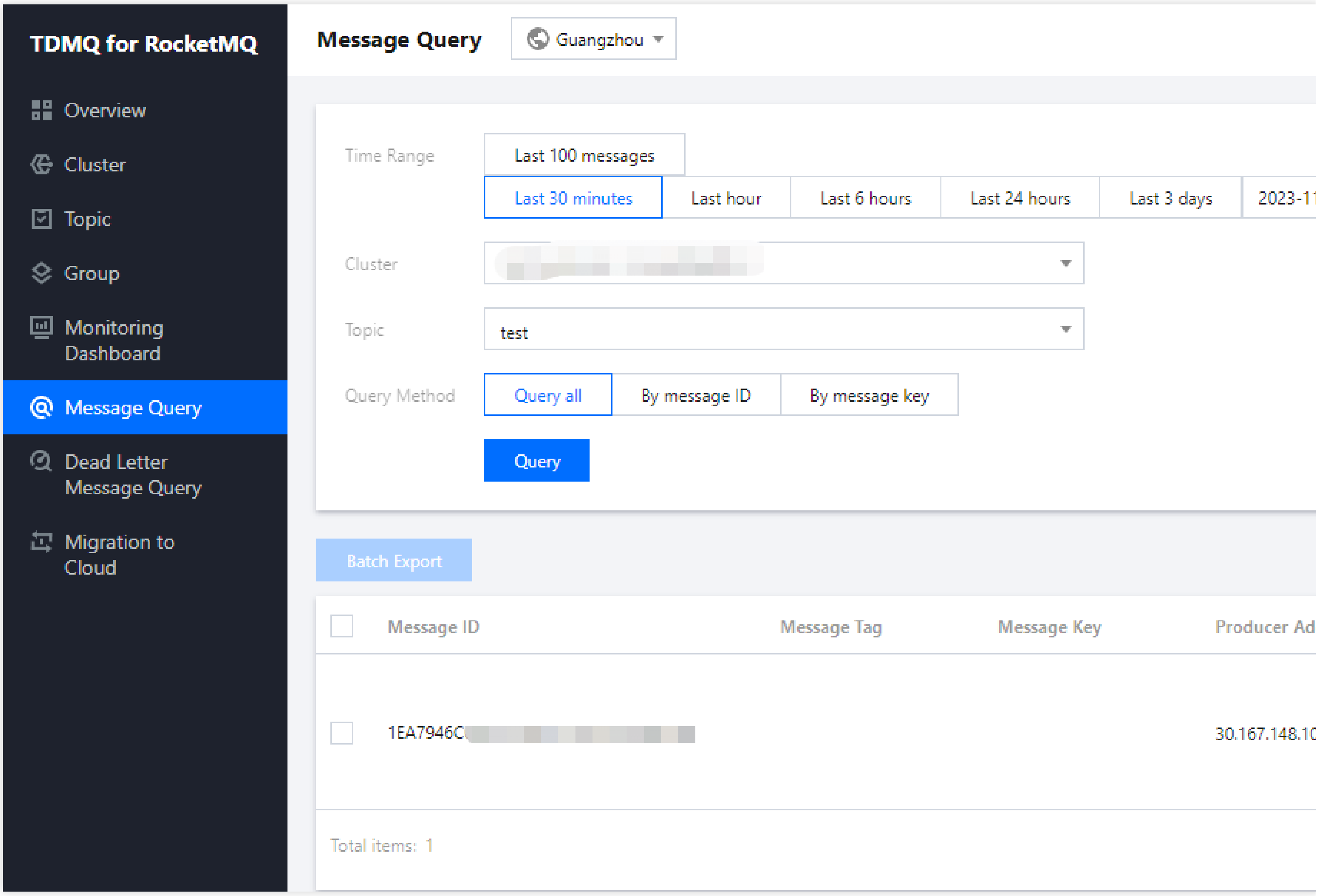
```
    err = c.Subscribe(topicName, selector, func(ctx context.Context,
                                                                  msgs ...*primitive
        fmt.Printf("subscribe callback len: %d \\n", len(msgs))
        // Set the delay level for the next consumption
        concurrentCtx, _ := 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 and respond with a retry
        return consumer.ConsumeRetryLater, nil
    })
    if err != nil {
        fmt.Println(err.Error())
    }
```

Parameter	Description
topicName	The name of the topic, copied from the Topic page on the console.
Expression	Message tag identifier.
delayLevel	Configure the delay level for re-consumption. A total of 18 delay levels are supported.

6. Consume the message (The consumer must consume the message after subscription).

```
// Initiate 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. Check consumption details. After the message is sent, you will receive a message ID (messageID). Developers can query the recently sent messages on the Message Query page, as shown in the following figure. Information such as details and traces for specific messages is also available. For details, see Message Query section.



Time Range: Last 30 minutes

Cluster:

Topic: test

Query Method: Query all

	Message ID	Message Tag	Message Key	Producer Ad
□	1EA7946C			30.167.148.1(

Total items: 1

Note:

This document briefly describes sending and receiving messages using the Golang client. For more operations, see the demo or the RocketMQ-Client-Go Examples.

Use of C++ SDK

Last updated：2024-01-17 16:56:58

Overview

This document describes how to use an open-source SDK to send and receive messages with the SDK for C++ serving as example, for you to better understand the complete procedure involved in message sending and receiving.

Prerequisites

You have installed GCC.

You have downloaded the demo.

Directions:

1. Prepare the environment.

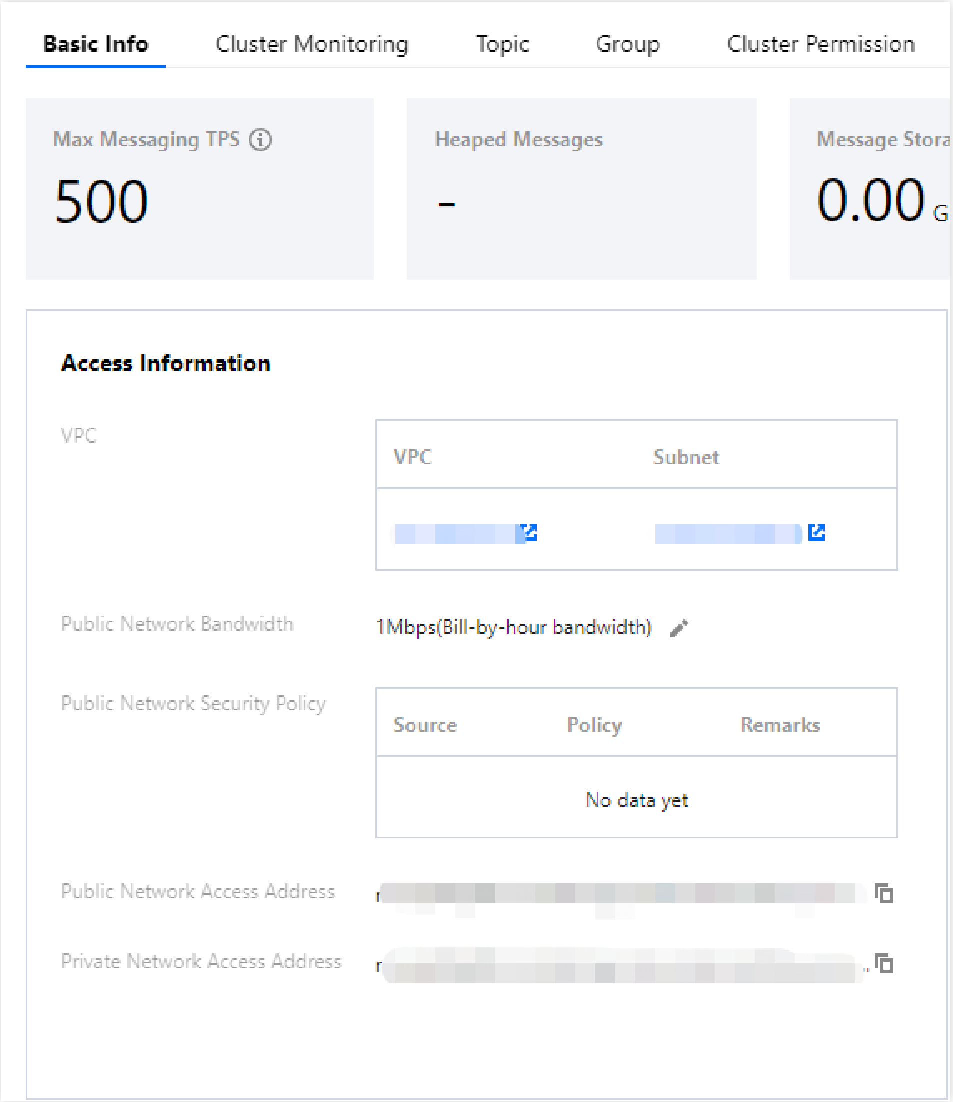
1.1 You need to install the RocketMQ-Client-CPP library in the client environment. Follow the official guide to Install the CPP Dynamic Library.**The master branch build is recommended**.

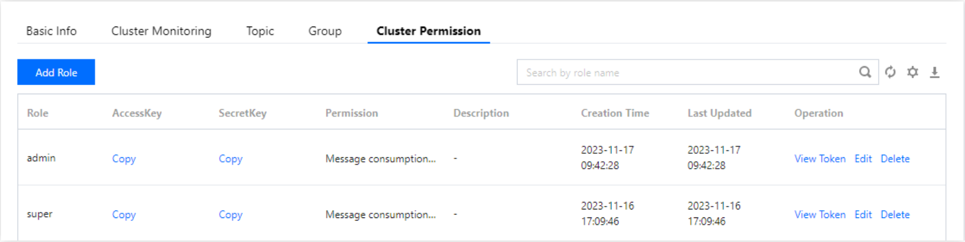
1.2 Incorporate the associated header files and dynamic libraries of RocketMQ-Client-CPP into the project.

2. Initialize 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 key
    secretKey, // Role name
    "");
// Set the namespace (full namespace name)
producer.setNameSpace(namespace);
```

```
// Ensure that the parameters are set before initiation
producer.start();
```

Parameter	Description
groupName	Producer group name, which can be obtained from the Group tab of cluster management on the console.
nameserver	Cluster access address in the basic information of the cluster. Select either the private network or public network access address as needed.  Basic Info · Cluster Monitoring · Topic · Group · Cluster Permission Max Messaging TPS: 500 Heaped Messages: - Message Stora… 0.00 G Access Information VPC: VPC [redacted] · Subnet [redacted] Public Network Bandwidth: 1Mbps(Bill-by-hour bandwidth) Public Network Security Policy: Source · Policy · Remarks — No data yet Public Network Access Address: [redacted] Private Network Access Address: [redacted]
secretKey	Role name, which can be copied from SecretKey on the Cluster Permission page.
accessKey	Role key, which can be copied from AccessKey on the Cluster Permission page.

Role	AccessKey	SecretKey	Permission	Description	Creation Time	Last Updated	Operation
admin	Copy	Copy	Message consumption...	-	2023-11-17 09:42:28	2023-11-17 09:42:28	View Token Edit Delete
super	Copy	Copy	Message consumption...	-	2023-11-16 17:09:46	2023-11-16 17:09:46	View Token Edit Delete

3. Send the message.

```
// Initialize message content
```

```
    MQMessage msg(
        topicName,  // Topic name
        TAGS,        // Message tag
        KEYS,        // Business 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 from the Topic tab on the console.
TAGS	Used to set the message tag.
KEYS	Used to configure the message business key.

4. Release the resources.

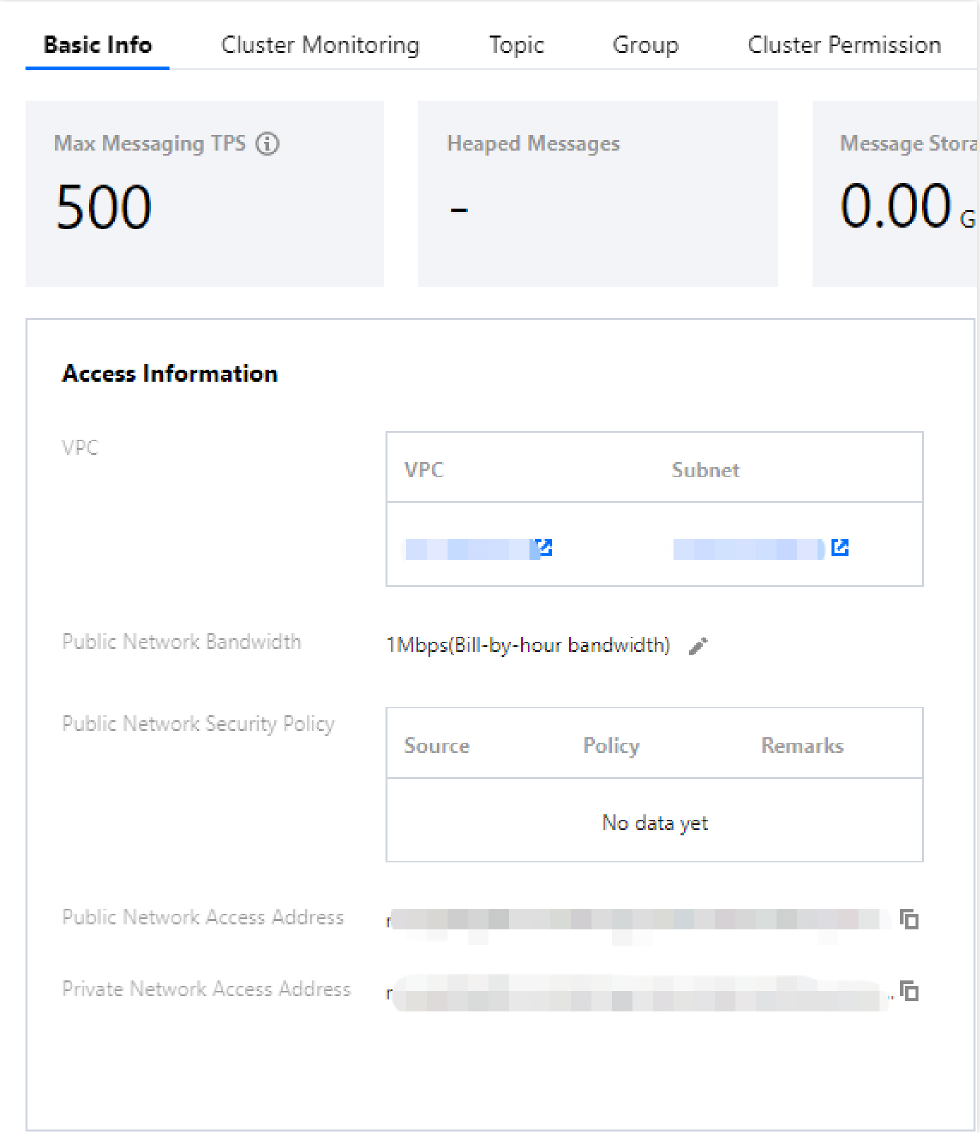
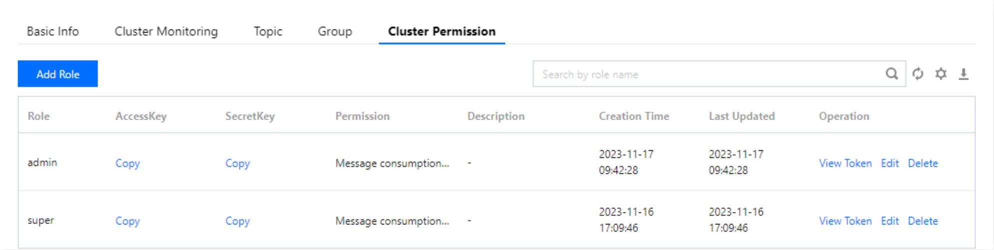
```
// Release resources
   producer.shutdown();
```

5. Initialize the consumer.

```
// Monitor 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` when consumption is successful
           return CONSUME_SUCCESS;
```

```
            // Return `RECONSUME_LATER` when consumption fails. 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");

    // Please register the custom listening function to process the received message
    ExampleMessageListener *messageListener = new ExampleMessageListener();
    // Subscribe to the message
    consumer->subscribe(topicName, TAGS);
    // Set the message listener
    consumer->registerMessageListener(messageListener);

    // After the preparations are complete, the startup function must be called for
    consumer->start();
```

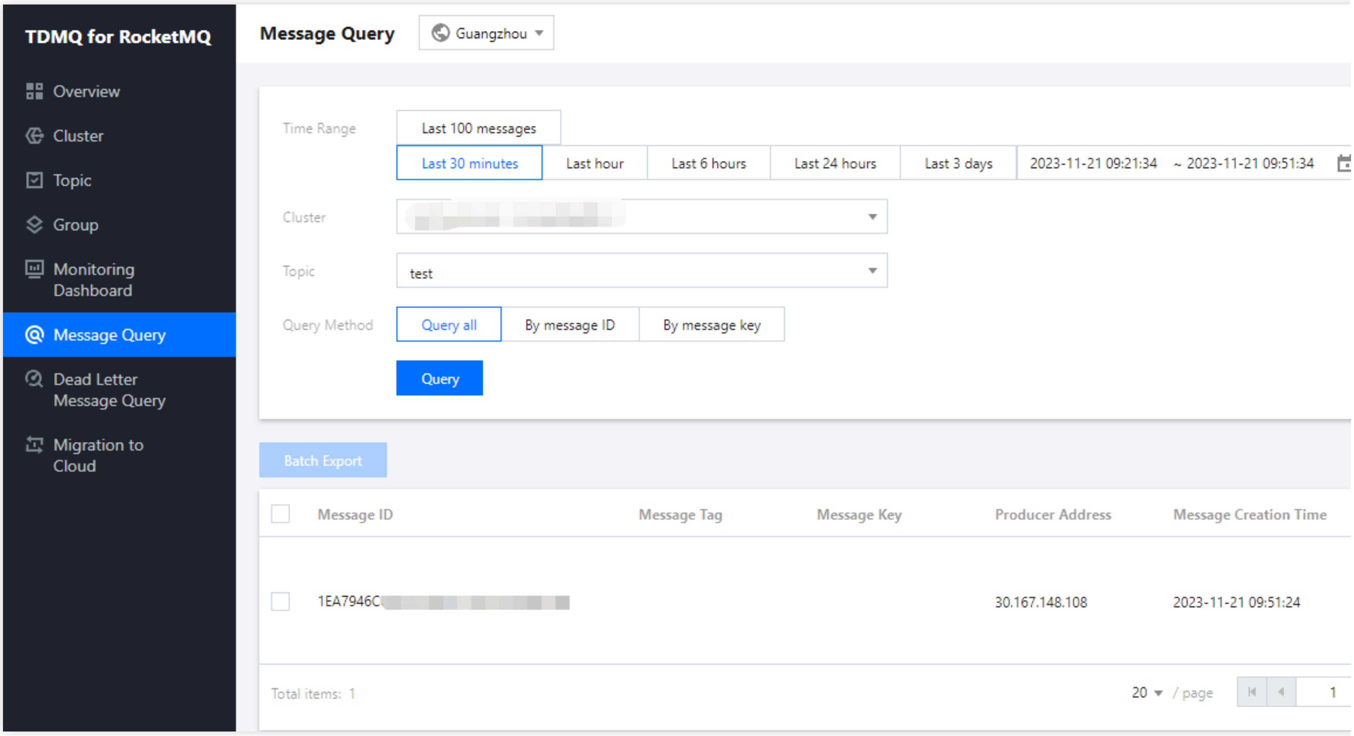
Parameter	Description
groupName	Consumer group name, which can be obtained from the Group tab of cluster management on the console.
nameserver	Cluster access address, which can be obtained by clicking on Get Access Address in the Operation column on the Cluster Management page.

<table>
<tr><td></td><td>

Basic Info | Cluster Monitoring | Topic | Group | Cluster Permission
Max Messaging TPS ⓘ 500
Heaped Messages -
Message Stora 0.00 G
<b>Access Information</b>
<table>
<tr><td>VPC</td><td><table><tr><th>VPC</th><th>Subnet</th></tr><tr><td>[redacted]</td><td>[redacted]</td></tr></table></td></tr>
<tr><td>Public Network Bandwidth</td><td>1Mbps(Bill-by-hour bandwidth)</td></tr>
<tr><td>Public Network Security Policy</td><td><table><tr><th>Source</th><th>Policy</th><th>Remarks</th></tr><tr><td colspan="3">No data yet</td></tr></table></td></tr>
<tr><td>Public Network Access Address</td><td>[redacted]</td></tr>
<tr><td>Private Network Access Address</td><td>[redacted]</td></tr>
</table>
</td></tr>
<tr><td>secretKey</td><td>Role name, which can be copied from SecretKey on the Cluster Permission page.</td></tr>
<tr><td>accessKey</td><td>Role key, which can be copied from AccessKey on the Cluster Permission page.

Basic Info | Cluster Monitoring | Topic | Group | <b>Cluster Permission</b>
Add Role | Search by role name
<table>
<tr><th>Role</th><th>AccessKey</th><th>SecretKey</th><th>Permission</th><th>Description</th><th>Creation Time</th><th>Last Updated</th><th>Operation</th></tr>
<tr><td>admin</td><td>Copy</td><td>Copy</td><td>Message consumption...</td><td>-</td><td>2023-11-17 09:42:28</td><td>2023-11-17 09:42:28</td><td>View Token Edit Delete</td></tr>
<tr><td>super</td><td>Copy</td><td>Copy</td><td>Message consumption...</td><td>-</td><td>2023-11-16 17:09:46</td><td>2023-11-16 17:09:46</td><td>View Token Edit Delete</td></tr>
</table>
</td></tr>
<tr><td>topicName</td><td>Topic name, which can be copied from the Topic tab on the console.</td></tr>
<tr><td>TAGS</td><td>Used to set the tag of the subscribed messages.</td></tr>
</table>

6. Release the resources.

```
// Release resources
consumer->shutdown();
```

7. After the message is sent, you will receive a message ID (messageID). Developers can query the recently sent messages on the Message Query page, as shown in the following figure. Information such as details and traces for specific messages is also available. For details, see Message Query.



Time Range: Last 30 minutes 2023-11-21 09:21:34 ~ 2023-11-21 09:51:34

Cluster: [redacted]

Topic: test

Query Method: Query all

□	Message ID	Message Tag	Message Key	Producer Address	Message Creation Time
□	1EA7946C[redacted]			30.167.148.108	2023-11-21 09:51:24

Total items: 1

Note:

The preceding sections briefly describe how to publish and subscribe to messages. For more operations, see the Demo or the RocketMQ-Client-CPP Examples.

Use of Python SDK

Last updated：2024-01-17 16:57:19

Overview

This document describes how to use an open-source SDK to send and receive messages with the SDK for Python serving as example, for you to better understand the complete procedure involved in message sending and receiving.

Prerequisites

You have created the required resources.

You have installed Python.

You have installed pip.

You have downloaded the demo.

Directions:

Step 1: Preparing the Environment

Rocketmq-client Python packaging is performed based on rocketmq-client-cpp. Therefore, `librocketmq` needs to be installed in advance.

Note:

Currently, the Python client only supports Linux and macOS operating systems, and Windows is not supported.

1. Install librocketmq (version 2.0.0 or higher). For installation instructions, see librocketmq Installation.
2. Execute the following command to install rocketmq-client-python.

```
pip install rocketmq-client-python
```

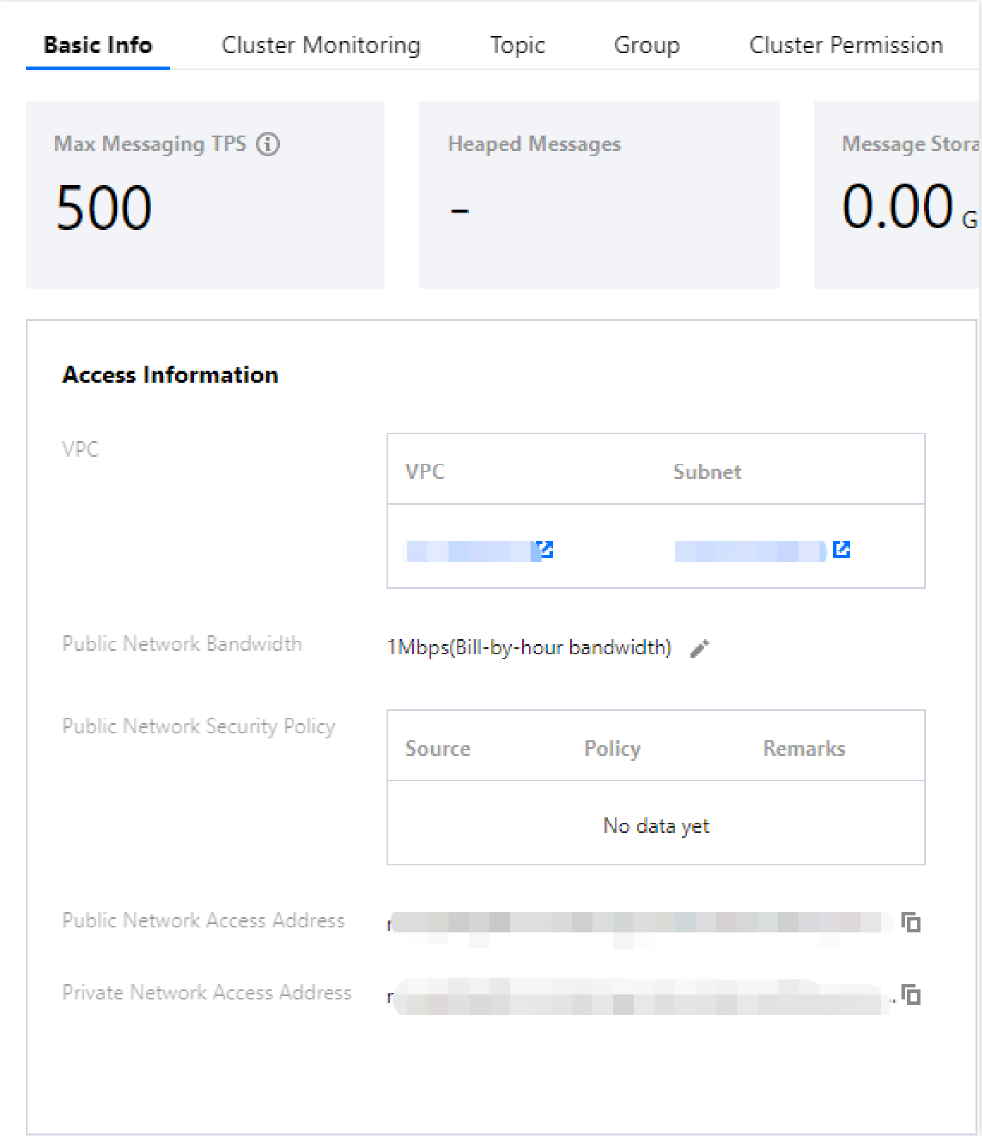
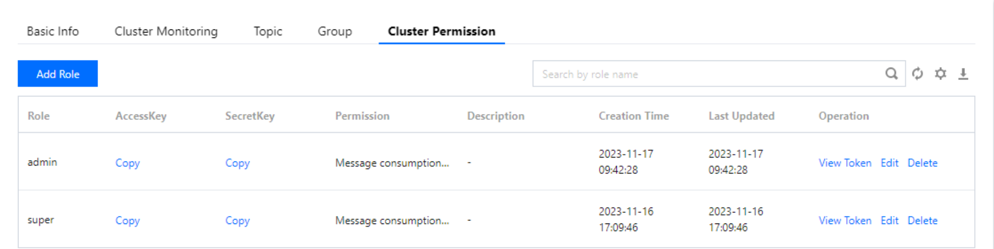
Step 2: Producing a Message

Write codes for producing the message.

```
from rocketmq.client import Producer, Message

   # Initialize the producer and set group information. group1
   producer = Producer(groupName)
   # Set the service address
   producer.set_name_server_address(nameserver)
   # Set permissions (role name and key)
   producer.set_session_credentials(
        accessKey,  # Role key
       secretKey,  # Role name
       ''
```

```
    )
    # Start the producer
    producer.start()

    # Assemble the message. The topic name can be copied from the Topic tab on the c
    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 synchronous messages
    ret = producer.send_sync(msg)
    print(ret.status, ret.msg_id, ret.offset)
    # Release resources
    producer.shutdown()
```

Parameter	Description
groupName	Producer group name, which is obtained from the Group tab of cluster management on the console.
nameserver	Cluster access address in the basic information of the cluster. Select either the private network or public network access address as needed.

	 Basic Info · Cluster Monitoring · Topic · Group · Cluster Permission Max Messaging TPS: 500 Heaped Messages: - Message Stora 0.00 G **Access Information** VPC: VPC [redacted] · Subnet [redacted] Public Network Bandwidth: 1Mbps(Bill-by-hour bandwidth) Public Network Security Policy: Source · Policy · Remarks · No data yet Public Network Access Address: [redacted] Private Network Access Address: [redacted]
secretKey	Role name, which can be copied from SecretKey on the Cluster Permission page.
accessKey	Role key, which can be copied from AccessKey on the Cluster Permission page.  Basic Info · Cluster Monitoring · Topic · Group · Cluster Permission Add Role · Search by role name Role · AccessKey · SecretKey · Permission · Description · Creation Time · Last Updated · Operation admin · Copy · Copy · Message consumption... · - · 2023-11-17 09:42:28 · 2023-11-17 09:42:28 · View Token Edit Delete super · Copy · Copy · Message consumption... · - · 2023-11-16 17:09:46 · 2023-11-16 17:09:46 · View Token Edit Delete
topicName	Topic name, which can be copied from the Topic tab on the console.
TAGS	Used to set the message tag.
KEYS	Used to configure the message business key.

The Python client of the open-source community has certain defects in message production, resulting in an uneven load distribution among different queues of the same topic. For details, see Defect Details.

Step 3: Consuming Messages

Create, compile, and execute a consumption message program.

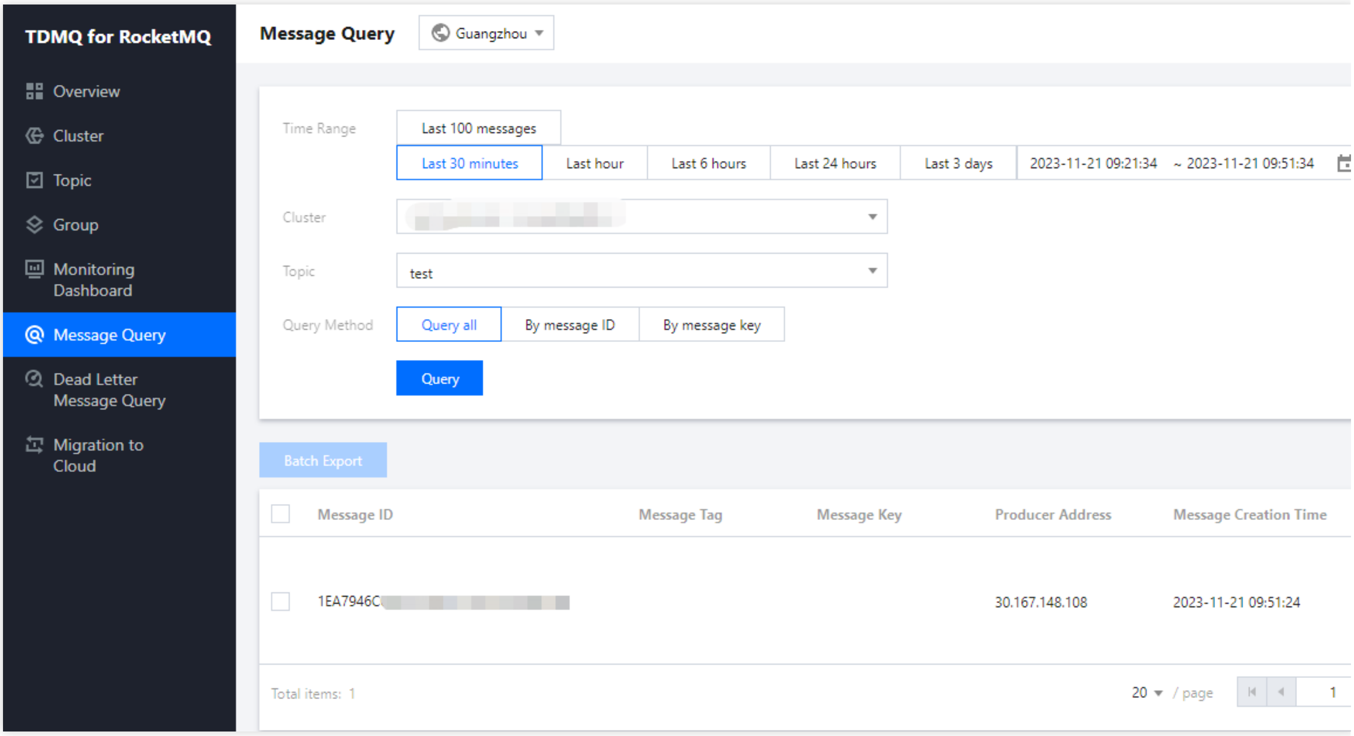
```
import time

    from rocketmq.client import PushConsumer, ConsumeStatus
    # Callback for message processing.
    def callback(msg):
        # Simulate the business.
        print('Received message. messageId: ', msg.id, ' body: ', msg.body)
        # Return `CONSUME_SUCCESS` if the consumption is successful.
        return ConsumeStatus.CONSUME_SUCCESS
        # Return the status of the message upon successful consumption.
        # 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 key)
    consumer.set_session_credentials(
        accessKey,  # Role key
        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 from the Group tab on the console.
nameserver	The same as the producer address.
secretKey	The same as the method of acquiring produced messages.
accessKey	The same as the method of acquiring produced messages.
topicName	Topic name, which can be copied from the Topic tab on the console.

TAGS	Set the tag of subscribed messages, which is set to * by default, indicating the subscription to all messages.

Step 4: Viewing Consumption Details

After the message is sent, you will receive a message ID (messageID). Developers can query the recently sent messages on the Message Query page, as shown in the following figure. Information such as details and traces for specific messages is also available. For details, see Message Query.



Time Range: Last 30 minutes 2023-11-21 09:21:34 ~ 2023-11-21 09:51:34

Cluster: [redacted]

Topic: test

Query Method: Query all

	Message ID	Message Tag	Message Key	Producer Address	Message Creation Time
□	1EA7946C[redacted]			30.167.148.108	2023-11-21 09:51:24

Total items: 1

Note:

The preceding sections briefly describe how to publish and subscribe to messages. For more operations, see the Demo or the RocketMQ Client Python Examples.

Use of Spring Cloud Stream

Last updated：2024-01-17 16:57:33

Overview

This document describes how to send and receive messages with the Spring Cloud Stream serving as example, for you to better understand the complete procedure involved in message sending and receiving.

Prerequisites

You have created the required resources.
You have installed JDK 1.8 or later.
You have installed Maven 2.5 or later.
You have downloaded the demo or visited the GitHub project.

Directions:

Step 1: Incorporating Dependencies

Incorporate the spring-cloud-starter-stream-rocketmq dependency in the pom.xml file. The current recommended version is 2021.0.5.0, and it is necessary to exclude dependencies, using SDK 4.9.7.

```
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-stream-rocketmq</artifactId>
    <version>2021.0.5.0</version>
   <exclusions>
     <exclusion>
         <groupId>org.apache.rocketmq</groupId>
         <artifactId>rocketmq-client</artifactId>
     </exclusion>
     <exclusion>
         <groupId>org.apache.rocketmq</groupId>
```

```
                    <artifactId>rocketmq-acl</artifactId>
                 </exclusion>
              </exclusions>
           </dependency>
           <dependency>
               <groupId>org.apache.rocketmq</groupId>
               <artifactId>rocketmq-client</artifactId>
               <version>4.9.7</version>
           </dependency>
           <dependency>
               <groupId>org.apache.rocketmq</groupId>
               <artifactId>rocketmq-acl</artifactId>
               <version>4.9.7</version>
           </dependency>
```

Step 2: Adding Configurations

Add the corresponding RocketMQ configurations to the configuration file.

```
spring:
  cloud:
    stream:
      rocketmq:
        binder:
          # Full name of the service address
          name-server: rmq-xxx.rocketmq.ap-bj.public.tencenttdmq.com:8080
          # Role name
          secret-key: admin
          # Role key
          access-key: eyJrZXlJZ...
```

```
            # producer group
            group: producerGroup
          bindings:
            # Channel name, corresponding to the channel name under spring.cloud.stre
            Topic-TAG1-Input:
              consumer:
                # Subscribed tag type, configured according to real consumer conditio
                subscription: TAG1
            # Channel name
            Topic-TAG2-Input:
              consumer:
                subscription: TAG2
        bindings:
          # Channel name
          Topic-send-Output:
            # Specify topic, corresponding to the created topic name
            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
```

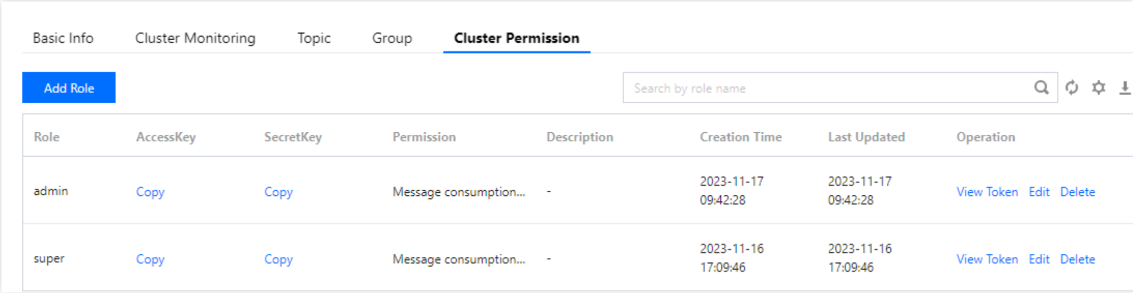
Note:

In terms of configuration, the subscription configuration item for `2.2.5-RocketMQ-RC1` and `2.2.5.RocketMQ.RC2` is `subscription` , and the configuration item for other lower versions is `tags` . The complete configuration item reference for other versions is as follows:

```
spring:
     cloud:
       stream:
         rocketmq:
           bindings:
             # Channel name, corresponding to the channel name under spring.cloud.s
             Topic-test1:
               consumer:
                 # Subscribed tag type, configured according to real consumer condi
                 tags: TAG1
             # Channel name
```

```
              Topic-test2:
                consumer:
                  tags: TAG2
            binder:
              # Full name of the service address
              name-server: rocketmq-xxx.rocketmq.ap-bj.public.tencenttdmq.com:8080
              # Role name
              secret-key: admin
              # Role key
              access-key: eyJrZXlJZ...
          bindings:
            # Channel name
            Topic-send:
              # Specified topic
              destination: topic1
              content-type: application/json
              # Use the full name of the group
              group: group1
            # Channel name
            Topic-test1:
              destination: topic1
              content-type: application/json
              group: group1
            # Channel name
            Topic-test2:
              destination: topic1
              content-type: application/json
              group: group2
```

Parameter	Description
name-server	Cluster access address, which can be copied from Access Address in the Operation column on the page on the console. Namespace access addresses in new version shared or exclusive clusters can copied from the namespace list.
secret-key	Role name, which can be copied from SecretKey on the Cluster Permission page.
access-key	Role key, which can be copied from AccessKey on the Cluster Permission page.

	
group	Producer group name, which can be copied from the Group tab on the console.
destination	Topic name, which can be copied from the Topic tab on the console.

Role	AccessKey	SecretKey	Permission	Description	Creation Time	Last Updated	Operation
admin	Copy	Copy	Message consumption...	-	2023-11-17 09:42:28	2023-11-17 09:42:28	View Token Edit Delete
super	Copy	Copy	Message consumption...	-	2023-11-16 17:09:46	2023-11-16 17:09:46	View Token Edit Delete

Step 3: Configuring the Channel

A channel consists of input and output. These can be individually configured as needed.

```
/**
 * Custom channel binder
 */
public interface CustomChannelBinder {

    /**
     * Send the message (message producer)
     * Bind the channel name specified in the configuration settings.
     */
    @Output("Topic-send-Output")
    MessageChannel sendChannel();
    /**
     * Receive Message 1 (Consumer 1)
     * Bind the channel name specified in the configuration settings.
     */
    @Input("Topic-TAG1-Input")
    MessageChannel testInputChannel1();

    /**
     * Receive Message 2 (Consumer 2)
     * Bind the channel name specified in the configuration settings.
     */
    @Input("Topic-TAG2-Input")
    MessageChannel testInputChannel2();
}
```

Step 4: Adding Annotations

Add relevant annotations to the configuration or boot class. If there are multiple configured binder configurations, each must be specifically specified within these annotations.

```
@EnableBinding({CustomChannelBinder.class})
```

Step 5: Sending the Messages

1. Inject `CustomChannelBinder` into the class of the message to be sent.

```
@Autowired
     private CustomChannelBinder channelBinder;
```

2. Send the messages by calling the corresponding output stream channel.

```
Message<String> message = MessageBuilder.withPayload("This is a new message.").buil
          channelBinder.sendChannel().send(message);
```

Step 6: Consuming the Messages

```
@Service
public class StreamConsumer {
    private final Logger logger = LoggerFactory.getLogger(StreamDemoApplication.cla

    /**
     * Monitor channel (designated by channel name in configuration)
     *
     * @param messageBody message content
     */
    @StreamListener("Topic-TAG1-Input")
    public void receive(String messageBody) {
        logger.info("Receive1: Message received via stream, messageBody = {}", mess
    }

    /**
     * Monitor channel (designated by channel name in configuration)
     *
     * @param messageBody message content
     */
    @StreamListener("Topic-TAG2-Input")
    public void receive2(String messageBody) {
        logger.info("Receive2: Message received via stream, messageBody = {}", mess
    }
}
```

Step 7: Local Test

After the project is initiated locally, a successful startup notification will be displayed on the console.

Visit `http://localhost:8080/test-simple` via a browser. You can see a successful transmission. Keep an eye on the output log of your 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 with the TAG1 has been sent, and only the subscriber of TAG1 has received the message.

Note:

For specific usage, see the GitHub Demo or Spring Cloud Stream Official Website.

Use of Spring Boot Starter

Last updated：2024-01-17 16:57:41

Overview

This document describes how to use the open-source SDK to send and receive messages with the Spring Boot Starter SDK serving as example, for you to better understand the complete procedure involved in message sending and receiving.

Prerequisites

You have created the required resources.
You have installed JDK 1.8 or later.
You have installed Maven 2.5 or later.
You have downloaded the Demo or visited the GitHub Project.

Directions:

Step 1: Incorporating Dependencies

Introduce dependencies of to pom.xml.

```
<dependency>
    <groupId>org.apache.rocketmq</groupId>
    <artifactId>rocketmq-spring-boot-starter</artifactId>
    <version>2.2.2</version>
     <exclusions>
             <exclusion>
                 <groupId>org.apache.rocketmq</groupId>
                 <artifactId>rocketmq-client</artifactId>
             </exclusion>
             <exclusion>
                 <groupId>org.apache.rocketmq</groupId>
```

```
                   <artifactId>rocketmq-acl</artifactId>
                </exclusion>
        </exclusions>
  </dependency>
   <dependency>
              <groupId>org.apache.rocketmq</groupId>
              <artifactId>rocketmq-client</artifactId>
              <version>4.9.7</version>
          </dependency>
          <dependency>
              <groupId>org.apache.rocketmq</groupId>
              <artifactId>rocketmq-acl</artifactId>
              <version>4.9.7</version>
   </dependency>
```

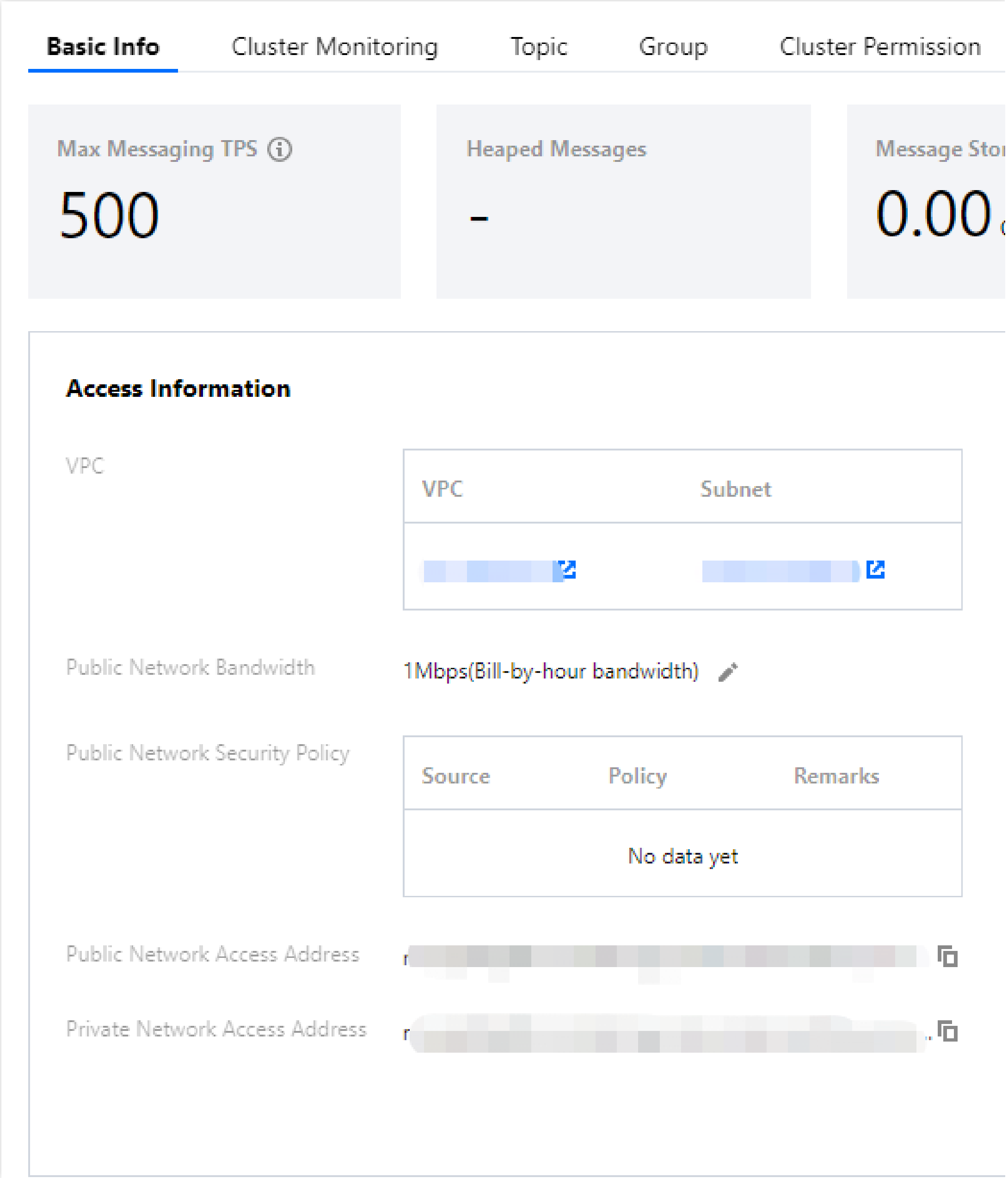
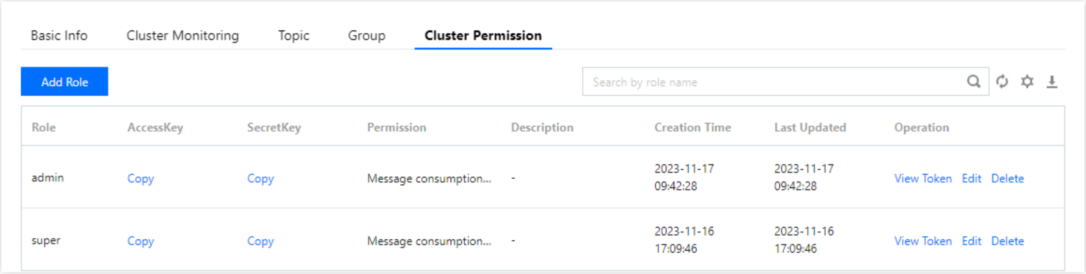
Step 2: Preparing Configurations

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:8080
     # Producer configurations
     producer:
       # Producer group name
       group: group111
```

```
      # Role key
      access-key: eyJrZXlJZC....
      # Name of the authorized role
      secret-key: admin
    # Common configurations for the consumer
    consumer:
      # Role key
      access-key: eyJrZXlJZC....
      # Name of the authorized role
      secret-key: admin

    # Custom configurations

    producer1:
      topic: testdev1
    consumer1:
      group: group111
      topic: testdev1
      subExpression: TAG1
    consumer2:
      group: group222
      topic: testdev1
      subExpression: TAG2
```

Parameter	Description
name-server	Cluster access address in the basic information of the cluster. Select either the private network o network access address as needed.

	 Basic Info · Cluster Monitoring · Topic · Group · Cluster Permission Max Messaging TPS: 500 Heaped Messages: - Message Stor: 0.00 **Access Information** VPC: VPC [redacted] · Subnet [redacted] Public Network Bandwidth: 1Mbps(Bill-by-hour bandwidth) Public Network Security Policy: Source · Policy · Remarks · No data yet Public Network Access Address: [redacted] Private Network Access Address: [redacted]
group	Producer group name, which can be copied from the Group tab on the console.
secret-key	Role name, which can be copied from SecretKey on the Cluster Permission page.
access-key	Role key, which can be copied from accessKey on the Cluster Permission page.  Basic Info · Cluster Monitoring · Topic · Group · **Cluster Permission** Add Role · Search by role name Role · AccessKey · SecretKey · Permission · Description · Creation Time · Last Updated · Operation admin · Copy · Copy · Message consumption... · - · 2023-11-17 09:42:28 · 2023-11-17 09:42:28 · View Token Edit Delete super · Copy · Copy · Message consumption... · - · 2023-11-16 17:09:46 · 2023-11-16 17:09:46 · View Token Edit Delete

topic	Topic name, which can be copied from the Topic tab on the console.
subExpression	Used to set the message tag.

Step 3: Sending the Message

1. Inject `RcoketMQTemplate` into the class in the message that needs to be sent.

```
@Value("${rocketmq.producer1.topic}")
      private String topic;  // Topic name
      @Autowired
      private RocketMQTemplate rocketMQTemplate;
```

2. Send the message. The message body can either be a custom object or a message object (from the org.springframework.messaging package).

```
SendResult sendResult = rocketMQTemplate.syncSend(destination, message);
/*------------------------------------------------------------------------*/
rocketMQTemplate.syncSend(destination, MessageBuilder.withPayload(message).build())
```

3. The following is a complete sample.

```
/**
* Description: Message producer
*/
@Service
public class SendMessage {
// Concatenate the topic name because the full name is required. Alternatively, you
 @Value("${rocketmq.producer1.topic}")
 private String topic;
     @Autowired
 private RocketMQTemplate rocketMQTemplate;
         /**
    * Synchronously sending
    *
    * @param message message content
    * @param tags    subscription tags
    */
 public void syncSend(String message, String tags) {
         // springboot does not support the use of headers to pass tags. Tags 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:

This example is for synchronous transmission. For information on asynchronous transmission, one-way transmission, and so on, see the Demo or visit the GitHub Project.

Step 4: Consuming the Message

```
@Service
   @RocketMQMessageListener(
           consumerGroup = "${rocketmq.consumer1.group}",  // Consumer group. Forma
           // Concatenate full topic name because the full name is required. Altern
           topic = "${rocketmq.consumer1.topic}",
           selectorExpression = "${rocketmq.consumer1.subExpression}" // Subscripti
   )
   public class MessageConsumer implements RocketMQListener<String> {

       @Override
       public void onMessage(String message) {
```

```
                System.out.println("Tag1Consumer has received the message:" + message);
            }
        }
```

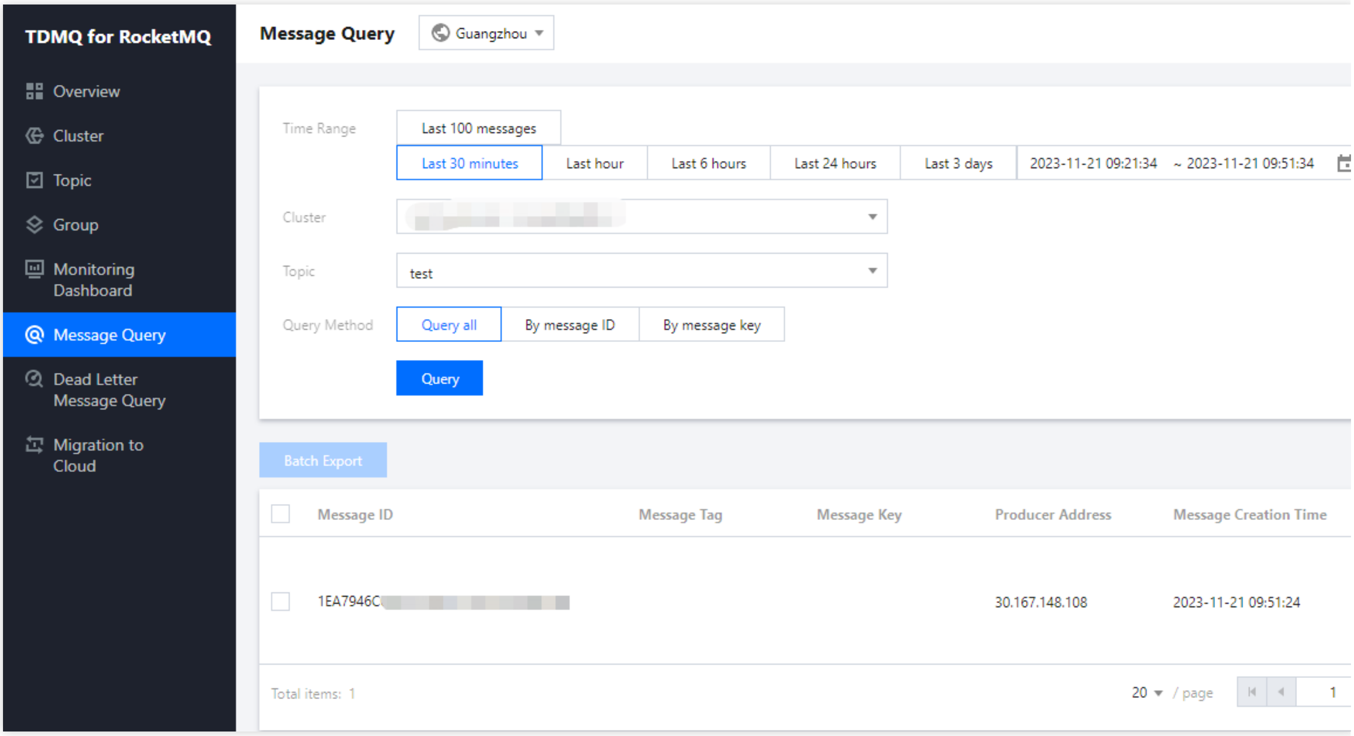
Multiple consumers can be configured based on your business needs. Other consumer configurations can be set as needed.

Note:

For the complete example, see Download Demo or visit the GitHub project.

Step 5: Checking Consumption Details

After the message is sent, you will receive a message ID (messageID). Developers can query the recently sent messages on the Message Query page, as shown in the following figure. Information such as details and traces for specific messages is also available. For details, see the Message Query section.



Message Query Guangzhou

Time Range: Last 30 minutes 2023-11-21 09:21:34 ~ 2023-11-21 09:51:34

Cluster: [redacted]

Topic: test

Query Method: Query all

	Message ID	Message Tag	Message Key	Producer Address	Message Creation Time
□	1EA7946C[redacted]			30.167.148.108	2023-11-21 09:51:24

Total items: 1 20 / page 1

Best Practice

Naming Conventions for Common Concepts of RocketMQ

Last updated：2024-03-18 14:53:27

This document introduces the naming conventions and usage norms for common concepts in RocketMQ.

Naming Conventions

topic

It must not be empty, and can only contain letters, digits, (-), and (_), 3 to 64 characters.

Suggested format: String.format (tp_%s_%s, system name, and business name)

For example: tp_order_check

tag

Can be empty, as long as it is a string, used for secondary message filtering under a topic.

Suggested format: String.format (tag_%s, business action or category)

For example: tag+business action, e.g., the tag for order creation is tag_create; the tag for order closure is tag_close

keys

Can be empty, recommended setting, as long as it is a string or an array of strings, used for querying messages or message traces in the console.

Suggested format: String.format (%s and unique business ID)

For example: order ID, transaction ID or serial number, TraceID, and other unique IDs

producer group

It must not be empty, 3 to 64 characters, and can only contain letters, digits, (-), and (_).

Suggested format: String.format (pg_%s_%s, system name, and business name)

For example: pg_transfer_check

consumer group

It must not be empty, 3 to 64 characters, and can only contain letters, digits, (-), and (_).

Suggested format: String.format (cg_%s_%s, system name, and business name)

For example: cg_transfer_check

role

It must not be empty, supports only digits, upper and lower case letters, and delimiters (_, -), and cannot exceed 32 characters.
Markers for different business read and write permissions, suggested format: business name + send/consume
For example: role_order_send, role_order_consume, and role_order_all

clientId

clientId represents a client instance ID, which must be unique across different clients. clientId cannot be directly set on the client. instanceName is an optional component of clientId, which can modify the clientId by adjusting instanceName.

Classification	Generating Strategy
Producer	${currentIP}@${instanceName}
Cluster Consumer	${currentIP}@${instanceName}
Broadcast Consumer	${currentIP}@${instanceName}

```
public String buildMQClientId() {
        StringBuilder sb = new StringBuilder();
        sb.append(this.getClientIP());

        sb.append("@");
        sb.append(this.getInstanceName());
        if (!UtilAll.isBlank(this.unitName)) {
            sb.append("@");
            sb.append(this.unitName);
        }
        if (enableStreamRequestType) {
            sb.append("@");
            sb.append(RequestType.STREAM);
        }

        return sb.toString();
    }
```

instanceName

Instance names, under default scenarios, do not require special settings by the user. The system will generate a unique value randomly through the following code by default.

```
public void changeInstanceNameToPID() {
      if (this.instanceName.equals("DEFAULT")) {
          this.instanceName = UtilAll.getPid() + "#" + System.nanoTime();
      }
  }
```

Broadcast consumers, upon every startup, need to keep the instance name constant to read the local progress file on the client. It is required to set the instanceName explicitly, and broadcast consumption must ensure the current Client IP remains unchanged before and after startup. If deploying in a container, it is necessary to set a fixed pod IP for scheduling; otherwise, broadcast messages during the restart period will be missed.

Suggested format: String.format (instance-%s-%s, system name, and business name).

Usage Guidelines

Producer

[Mandatory] A **domain** service can only have one topic.

[Mandatory] When sending messages, a **domain** service must set the tag according to the business action.

[Mandatory] The keys must be set when the producer sends a message.

[Mandatory] Logs must be printed whether a message sending succeeded or failed, with both the SendResult and key fields required to be printed.

[Recommended] After a message-sending failure, it is recommended to store the message in a database, and then use a timer-like thread for periodic retries to ensure the message can be delivered to the broker.

[Recommended] For business scenarios where reliability is not a high requirement, the oneway messages can be used.

[Mandatory] When creating a new producer, a producer group must be specified.

Consumer

[Mandatory] When creating a new consumer, a producer group must be specified.

[Mandatory] Message consumers cannot avoid duplicate messages, so the business service needs to ensure idempotent message consumption.

[Recommended] To improve consumption parallelism, you can start multiple Consumer instances under the same ConsumerGroup or increase the parallel consumption capacity of a single Consumer by modifying ConsumeThreadMin and ConsumeThreadMax.

[Recommended] To increase business throughput, you can batch the consume messages by setting the consumer's consumeMessageBatchMaxSize.

[Recommended] In the event of message accumulation, if the business does not have high data requirements, you can choose to discard unimportant messages.

[Recommended] If the message volume is small, it is recommended to print messages and consumption time, etc. at the entry point of consumption to facilitate subsequent troubleshooting.