ALIBABA CLOUD

# Alibaba Cloud

## Apsara Stack Enterprise

## User Guide - Middleware and Enterprise Applications

Product Version: V3.15.0

Document Version: 20220210

Alibaba Cloud

# Legal disclaimer

Alibaba Cloud reminds you to carefully read and fully understand the terms and conditions of this legal disclaimer before you read or use this document. If you have read or used this document, it shall be deemed as your total acceptance of this legal disclaimer.

1. You shall download and obtain this document from the Alibaba Cloud website or other Alibaba Cloud-authorized channels, and use this document for your own legal business activities only. The content of this document is considered confidential information of Alibaba Cloud. You shall strictly abide by the confidentiality obligations. No part of this document shall be disclosed or provided to any third party for use without the prior written consent of Alibaba Cloud.

2. No part of this document shall be excerpted, translated, reproduced, transmitted, or disseminated by any organization, company or individual in any form or by any means without the prior written consent of Alibaba Cloud.

3. The content of this document may be changed because of product version upgrade, adjustment, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and an updated version of this document will be released through Alibaba Cloud-authorized channels from time to time. You should pay attention to the version changes of this document as they occur and download and obtain the most up-to-date version of this document from Alibaba Cloud-authorized channels.

4. This document serves only as a reference guide for your use of Alibaba Cloud products and services. Alibaba Cloud provides this document based on the "status quo", "being defective", and "existing functions" of its products and services. Alibaba Cloud makes every effort to provide relevant operational guidance based on existing technologies. However, Alibaba Cloud hereby makes a clear statement that it in no way guarantees the accuracy, integrity, applicability, and reliability of the content of this document, either explicitly or implicitly. Alibaba Cloud shall not take legal responsibility for any errors or lost profits incurred by any organization, company, or individual arising from download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, take responsibility for any indirect, consequential, punitive, contingent, special, or punitive damages, including lost profits arising from the use or trust in this document (even if Alibaba Cloud has been notified of the possibility of such a loss).

5. By law, all the contents in Alibaba Cloud documents, including but not limited to pictures, architecture design, page layout, and text description, are intellectual property of Alibaba Cloud and/or its affiliates. This intellectual property includes, but is not limited to, trademark rights, patent rights, copyrights, and trade secrets. No part of this document shall be used, modified, reproduced, publicly transmitted, changed, disseminated, distributed, or published without the prior written consent of Alibaba Cloud and/or its affiliates. The names owned by Alibaba Cloud shall not be used, published, or reproduced for marketing, advertising, promotion, or other purposes without the prior written consent of Alibaba Cloud. The names owned by Alibaba Cloud include, but are not limited to, "Alibaba Cloud", "Aliyun", "HiChina", and other brands of Alibaba Cloud and/or its affiliates, which appear separately or in combination, as well as the auxiliary signs and patterns of the preceding brands, or anything similar to the company names, trade names, trademarks, product or service names, domain names, patterns, logos, marks, signs, or special descriptions that third parties identify as Alibaba Cloud and/or its affiliates.

6. Please directly contact Alibaba Cloud for any errors of this document.

# Document conventions

| Style | Description | Example |
| --- | --- | --- |
| Danger | A danger notice indicates a situation that will cause major system changes, faults, physical injuries, and other adverse results. | **Danger:** Resetting will result in the loss of user configuration data. |
| Warning | A warning notice indicates a situation that may cause major system changes, faults, physical injuries, and other adverse results. | **Warning:** Restarting will cause business interruption. About 10 minutes are required to restart an instance. |
| Notice | A caution notice indicates warning information, supplementary instructions, and other content that the user must understand. | **Notice:** If the weight is set to 0, the server no longer receives new requests. |
| Note | A note indicates supplemental instructions, best practices, tips, and other content. | **Note:** You can use Ctrl + A to select all files. |
| > | Closing angle brackets are used to indicate a multi-level menu cascade. | Click **Settings**> **Network**> **Set network type**. |
| **Bold** | Bold formatting is used for buttons , menus, page names, and other UI elements. | Click **OK**. |
| `Courier font` | Courier font is used for commands | Run the `cd /d C:/window` command to enter the Windows system folder. |
| *Italic* | Italic formatting is used for parameters and variables. | `bae log list --instanceid` *Instance_ID* |
| [] or [a\|b] | This format is used for an optional value, where only one item can be selected. | `ipconfig [-all\|-t]` |
| {} or {a\|b} | This format is used for a required value, where only one item can be selected. | `switch {active\|stand}` |

# Table of Contents

# 1.Message Queue for Apache RocketMQ

## 1.1. User Guide

### 1.1.1. What is Message Queue for Apache RocketMQ?

Message Queue for Apache RocketMQ is a distributed messaging middleware that is developed by Alibaba Cloud based on Apache RocketMQ. This Alibaba Cloud service features low latency, high concurrency, high availability, and high reliability.

Message Queue for Apache RocketMQ is a core service in the enterprise-level Internet architecture. Using the highly available distributed cluster technology, this service offers a series of messaging services in the cloud, including message subscription and delivery, message tracing and query, scheduled or delayed messages, and resource statistics. Message Queue for Apache RocketMQ provides asynchronous decoupling and load shifting for distributed application systems. It also supports various features for Internet applications, including massive message accumulation, high throughput, and reliable message consumption retry. It is one of the core Alibaba Cloud services that are used to support the Double 11 Shopping Festival.

Message Queue for Apache RocketMQ supports access by using TCP. It also supports the Java, C++,and .NET programming languages. This facilitates quick access to Message Queue for Apache RocketMQ for applications that are developed in different programming languages.

### 1.1.2. Updates

This topic describes the updates of Message Queue for Apache RocketMQ from V3.8.0 to V3.8.1 to help you get started with the updated version.

#### Optimization of resource isolation by instance

Message Queue for Apache RocketMQ provides instances for multi-tenancy isolation. Each user can purchase multiple instances and logically isolate them from each other.

To ensure the compatibility with the existing resources of existing users, Message Queue for Apache RocketMQ provides the following types of instances and namespaces:

- Default instances, which are compatible with the existing resources of existing users
  - This type of instance has no separate namespace. Resource names must be globally unique within and across all instances.
  - By default, an instance without a namespace is automatically generated for the existing resources of each existing user. If no existing resources are available, you can create at most one instance without a namespace.

- You can configure the endpoint, which can be obtained from the **Instances** page in the Message Queue for Apache RocketMQ console.

```
// Recommended configuration:
properties.put(PropertyKeyConst.NAMESRV_ADDR, "xxxx");
// Compatible configuration, which is not recommended. We recommend that you update thi
s configuration to the recommended configuration:
properties.put(PropertyKeyConst.ONSAddr, "xxxx");
```

- New instances
  - A new instance has a separate namespace. Resource names must be unique within an instance but can be the same across different instances.
  - You can configure the endpoint, which can be obtained from the **Instances** page in the Message Queue for Apache RocketMQ console.

```
// Recommended configuration:
properties.put(PropertyKeyConst.NAMESRV_ADDR, "xxx");
```

  - A RocketMQ client must be updated to the following latest versions for different programming languages:
    - Java: V1.8.7.1.Final
    - C and C++: V2.0.0
    - .NET: V1.1.3

## Optimization of resource application

Previously, Message Queue for Apache RocketMQ resources consisted of topics, producer IDs, and consumer IDs. Each two of the resources have a many-to-many relationship, which was difficult to comprehend. Each time you created a topic, you must associate the topic with a producer ID and a consumer ID. This process was too complex for medium- and large-sized enterprise customers.

To optimize user experience and help new users get started, the resource application process has been simplified.

The resource application process has been optimized in the following aspects:

- Topic management, which is unchanged
  - You need to apply for a topic. A topic is used to classify messages. It is the primary classifier.
- Group management
  - You do not need to apply for a producer ID. Producer IDs and consumer IDs are integrated into group IDs. In the Message Queue for Apache RocketMQ console, the Producers module has been removed. The Producers and Consumers modules have been integrated into the Groups module.
  - You do not need to associate a producer ID or consumer ID with a topic. Instead, you need only to apply for a group ID and associate it with a topic in the code.
  - Compatibility:
    - The list of producer IDs is no longer displayed. This does not affect the current services.
    - The consumer IDs that start with CID- or CID_ and that you have applied for can still be used and can be set in the PropertyKeyConst.ConsumerId or PropertyKeyConst.GROUP_ID parameter of the code.
- Sample code

> **Note**
>
> - We recommend that you update a RocketMQ client to the following latest versions for different programming languages:
>   - Java: V1.8.7.1.Final
>   - C and C++: V2.0.0
>   - .NET: V1.1.3
> - Existing producer IDs or consumer IDs can still be used and do not affect the current services. However, we recommend that you update your instance configuration to the recommended configuration.

- Recommended configuration: Integrate producer IDs and consumer IDs into group IDs.

```
// Set the PropertyKeyConst.GROUP_ID parameter. The original PropertyKeyConst.ProducerId and PropertyKeyConst.ConsumerId parameters are deprecated.
properties.put(PropertyKeyConst.GROUP_ID, "The original CID-XXX or the GID-XXX");
```

- Compatible configuration: Use a producer ID to identify a producer and a consumer ID to identify a consumer.

```
// When you create a producer, you must set the PropertyKeyConst.ProducerId parameter.
properties.put(PropertyKeyConst.ProducerId, "The original PID-XXX or the GID-XXX");
// When you create a consumer, you must set the PropertyKeyConst.ConsumerId parameter.
properties.put(PropertyKeyConst.ConsumerId, "The original CID-XXX or the GID-XXX");
```

# 1.1.3. Quick start

## 1.1.3.1. Overview

This topic describes the complete process from creating Message Queue for Apache RocketMQ resources to using Message Queue for Apache RocketMQ SDKs to send and receive messages. This helps you get started with Message Queue for Apache RocketMQ.

Quick access to Message Queue for Apache RocketMQ


1. Create resources
2. Send messages
3. Subscribe to messages

In Apsara Stack, Message Queue for Apache RocketMQ supports TCP and multiple programming languages.

Programming languages supported by Message Queue for Apache RocketMQ

| Java | C/C++ | .NET | PHP | Pyhton |
| --- | --- | --- | --- | --- |

| Java | C/C++ | .NET | PHP | Pyhton |
| --- | --- | --- | --- | --- |
| Yes | Yes | Yes | No (Open source SDKs are available) | No (Open source SDKs are available) |

If your server application uses Message Queue for Apache RocketMQ, we recommend that you use an SDK to access Message Queue for Apache RocketMQ. This method is easy-to-use and provides high availability.

As a quick start guide, this topic demonstrates how to access Message Queue for Apache RocketMQ to send and receive messages. In the example, TCP client SDK for Java is used.

### References

To send and receive messages by using TCP client SDKs for other programming languages, see the following documentation:

- C and C++: Send and receive normal messages
- .NET: Send and subscribe to normal messages

# 1.1.3.2. Log on to the Message Queue for Apache RocketMQ console

This topic describes how to log on to the Message Queue for Apache RocketMQ console.

### Prerequisites

- The URL of the Apsara Uni-manager Management Console is obtained from the deployment personnel before you log on to the Apsara Uni-manager Management Console.
- A browser is available. We recommend that you use the Google Chrome browser.

### Procedure

1. In the address bar, enter the URL of the Apsara Uni-manager Management Console. Press the Enter key.
2. Enter your username and password.

   Obtain the username and password that you can use to log on to the console from the operations administrator.

   > **Note** When you log on to the Apsara Uni-manager Management Console for the first time, you must change the password of your username. Your password must meet complexity requirements. The password must be 8 to 20 characters in length and must contain at least two of the following character types:
   >
   > - Uppercase or lowercase letters
   > - Digits
   > - Special characters, which include ! @ # $ %

3. Click **Login**.

4. If your account has multi-factor authentication (MFA) enabled, perform corresponding operations in the following scenarios:
   - It is the first time that you log on to the console after MFA is forcibly enabled by the administrator.
     a. On the Bind Virtual MFA Device page, bind an MFA device.
     b. Enter the account and password again as in Step 2 and click **Log On**.
     c. Enter a six-digit MFA verification code and click **Authenticate**.
   - You have enabled MFA and bound an MFA device.

     Enter a six-digit MFA authentication code and click **Authenticate**.

     > **Note** For more information, see the *Bind a virtual MFA device to enable MFA* topic in *Apsara Uni-manager Operations Console User Guide*.
5. In the top navigation bar, choose **Products > Middleware > Message Queue**.

# 1.1.3.3. Create resources

# 1.1.3.3.1. Resource management overview

This topic describes how to manage resources in Message Queue for Apache RocketMQ.

If a new application needs to access Message Queue for Apache RocketMQ, you must create the following Message Queue for Apache RocketMQ resources for the application:

- Instance: As a virtual machine (VM) resource of Message Queue for Apache RocketMQ, an instance stores the topics and group IDs of messages.
- Topic: In Message Queue for Apache RocketMQ, a producer sends a message to a specified topic, and a consumer subscribes to the topic to obtain and consume the message.
- Group ID: A group ID is used to identify a group of message consumers or producers.

You can add, delete, modify, and query these resources by using the Message Queue for Apache RocketMQ console or by calling the Message Queue for Apache RocketMQ API.

When you use SDKs to send and subscribe to messages, you must specify the topic and group ID that you created in the Message Queue for Apache RocketMQ console. You must also enter the AccessKey ID and AccessKey secret that you created in the Apsara Uni-manager Management Console for identity authentication.

If you have not obtained the AccessKey ID and AccessKey secret, you can obtain them in the Apsara Uni-manager Management Console. For more information, see Obtain the AccessKey ID and AccessKey secret.

# 1.1.3.3.2. Create an instance

An instance stores topics and group IDs.

## Procedure

1. Log on to the Message Queue for Apache RocketMQ console.
2. On the **Instance Details** page, click **Create Instance**.

3. On the **Create Instance** page, enter the information as prompted and click **Submit**.

# 1.1.3.3.3. Create a topic

Topic is the first-level identifier that classifies messages in Message Queue for Apache RocketMQ. For example, you can create a topic named Topic_Trade to identify transaction-specific messages.

## Prerequisites

Create an instance

## Procedure

1. Log on to the Message Queue for Apache RocketMQ console. In the left-side navigation pane, click **Topics**.
2. In the upper part of the **Topics** page, select your instance.
3. Click **Create Topic**.
4. In the **Create Topic** dialog box, enter a name for the topic in the **Topic** field.

   > **Notice** The topic name must be unique in the instance where you create the topic and must comply with the following rules:
   >
   > - The topic name cannot start with CID or GID, because CID and GID are reserved fields for group IDs.
   > - The topic name can contain only letters, digits, hyphens (-), and underscores (_).
   > - The topic name must be 3 to 64 characters in length.

5. Select a value for **Message Type**. This value defines the type of message that this topic sends and receives.

   We recommend that you create different topics to send different types of messages. For example, create Topic A for normal messages, Topic B for transactional messages, and Topic C for scheduled messages or delayed messages. For more information about message types, see Message types.

6. In the **Description** field, enter a description about the topic and click **OK**.
   The created topic appears in the topic list.

# 1.1.3.3.4. Create a group ID

After you create an instance and a topic, you must create a group ID for the message consumer or producer.

## Prerequisites

Create an instance

## Context

- A group ID is required for consumers but is optional for producers.
- The group ID must be unique in the instance where you create the group ID.
- Group IDs have an N:N relationship with topics. A consumer can subscribe to multiple topics and a topic can be subscribed to by multiple consumers. A producer can send messages to multiple topics

and a topic can receive messages from multiple producers.

- If a group ID for consumers or an existing consumer ID is created by using a Resource Access Management (RAM) user, the RAM user and its Apsara Stack tenant account have the permissions to use this group ID or consumer ID.
- If a group ID for consumers or an existing consumer ID is created by using an Apsara Stack tenant account, only the Apsara Stack tenant account has the permissions to use this group ID or consumer ID. The RAM users of this Apsara Stack tenant account do not have the permissions to use this group ID or consumer ID.
- For more information about changes from existing consumer IDs and producer IDs to group IDs, see Updates.

### Procedure

1. Log on to the Message Queue for Apache RocketMQ console. In the left-side navigation pane, click **Groups**.
2. In the upper part of the **Groups** page, select the instance that you created.
3. Click **Create Group ID**.
4. In the **Create Group ID** dialog box, enter a group ID and description. Then, click **OK**.

   > **Notice**
   >
   > The group ID must comply with the following rules:
   >
   > - The group ID must start with GID_ or GID- and can contain only letters, digits, hyphens (-), and underscores (_).
   > - The group ID must be 7 to 64 characters in length.
   > - The group ID cannot be modified after it is created.

   The created group ID appears in the group ID list.

# 1.1.3.4. Send messages

# 1.1.3.4.1. Overview

After you create a topic in the Message Queue for Apache RocketMQ console, you can send messages. Message Queue for Apache RocketMQ allows you to send messages by using the console or SDKs.

- To verify the availability of a topic, you can send messages in the console.
- In a production environment, you can use an SDK to send messages in Message Queue for Apache RocketMQ.

# 1.1.3.4.2. Send messages in the console

After you create a topic in the Message Queue for Apache RocketMQ console, you can send messages in the console to verify the availability of the topic.

### Procedure

1. Log on to the Message Queue for Apache RocketMQ console. In the left-side navigation pane, click **Topics**.

2. On the **Topics** page, find the topic that you created and click **Send Message** in the **Actions** column.
3. In the **Send Message** dialog box, enter the message content in the **Message Body** field and click **OK**.

   Parameters for sending a message

| Parameter | Description |
| --- | --- |
| **Tag** | A tag is a label that classifies messages into different types in a topic in Message Queue for Apache RocketMQ. Message Queue for Apache RocketMQ allows consumers to filter messages by using tags. This ensures that the consumers consume only messages that they are concerned with. |
| **Key** | The key of the message. It is the key attribute of the message. Keep the key unique and business-distinctive whenever possible. Message Queue for Apache RocketMQ creates an index for messages based on message keys that you specify. This way, when you query messages by message key, the messages that match the index can be hit and returned. |
| **Message Body** | The body of the message. |

   After the message is sent, a success prompt and the corresponding message ID are returned.

# 1.1.3.4.3. Send messages by using SDKs

If you use Message Queue for Apache RocketMQ in production environments, we recommend that you use SDKs to send messages. In this topic, TCP client SDK for Java is used as an example.

## Procedure

1. Use one of the following methods to add the dependency:
   - Add the dependency by using Maven:

```
<dependency>
   <groupId>com.aliyun.openservices</groupId>
   <artifactId>ons-client</artifactId>
   <version>"XXX"</version>
   // The latest version of SDK for Java.
</dependency>
```

   - Download the JAR dependency:
2. Set related parameters and run the sample code based on the following instructions:

```
import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.Producer;
import com.aliyun.openservices.ons.api.SendResult;
import com.aliyun.openservices.ons.api.ONSFactory;
import com.aliyun.openservices.ons.api.PropertyKeyConst;
```

```
import com.aliyun.openservices.ons.api.PropertyKeyConst;
import java.util.Properties;
public class ProducerTest {
    public static void main(String[] args) {
        Properties properties = new Properties();
        // The group ID that you created in the console.
        properties.put(PropertyKeyConst.GROUP_ID, "XXX");
        // The AccessKey ID used for identity authentication.
        properties.put(PropertyKeyConst.AccessKey,"XXX");
        // The AccessKey secret used for identity authentication.
        properties.put(PropertyKeyConst.SecretKey, "XXX");
        // The TCP endpoint. To obtain the endpoint, log on to the Message Queue for Apache RocketMQ console. In the left-side navigation pane, click Instance Details. In the upper part of the Instance Details page, select your instance. On the Instance Information tab, view the endpoint in the Obtain Endpoint Information section.
        properties.put(PropertyKeyConst.NAMESRV_ADDR,"XXX");
        Producer producer = ONSFactory.createProducer(properties);
        // Before you send a message, call the start() method once to start the producer.
        producer.start();
        //Send messages cyclically.
        while(true){
            Message msg = new Message( //
                // The topic that you created in the console. This is the name of the topic to which the message belongs.
                "TopicTestMQ",
                // Message Tag,
                // The message tag, which is similar to a Gmail tag. It is used to sort messages and helps the consumer filter messages on the Message Queue for Apache RocketMQ broker based on specified conditions.
                "TagA",
                // Message Body
                // The message body in the binary format. Message Queue for Apache RocketMQ does not process the message body.
                // The producer and consumer must agree on the serialization and deserialization methods.
                "Hello MQ".getBytes());
            // The message key, which is the key attribute of the message and must be globally unique. A unique key helps you query and resend a message in the console if the message fails to be received.
            // Note: Messages can be sent and received even if you do not specify the message key.
            msg.setKey("ORDERID_100");
            // Send the message. If no error occurs, the message is sent.
            // Print the message ID, which can be used to query the sending status of the message.
            SendResult sendResult = producer.send(msg);
            System.out.println("Send Message success. Message ID is: " + sendResult.getMessageId());
        }
        // Before you exit the application, shut down the producer object.
        // Note: You can choose not to shut down the producer object.
        producer.shutdown();
    }
}
```

# 1.1.3.4.4. Check whether messages are sent

After you send a message, you can check the status of the message in the console.

## Procedure

1. Log on to the Message Queue for Apache RocketMQ console. In the left-side navigation pane, click **Message Query**.
2. On the **Message Query** page, click the **By Message ID** tab.
3. In the search box, enter the topic name that corresponds to the message and the message ID returned after the message is sent, and click **Search** to query the sending status of the message.

   **Storage Time** indicates the time when the Message Queue for Apache RocketMQ broker stores the message. If the message appears in the search results, the message has been sent to the Message Queue for Apache RocketMQ broker.

   > **Notice** This step demonstrates the situation where Message Queue for Apache RocketMQ is used for the first time and the consumer has never been started. Therefore, no consumption data appears in the message status information.

## What's next

You can start the consumer and subscribe to messages. For more information, see Subscribe to messages. For more information about the message status, see Query messages and Message tracing status.

# 1.1.3.5. Subscribe to messages

After a message is sent, the consumer can subscribe to the message. You need to use the SDK for the corresponding protocol and programming language to subscribe to the message. This topic describes how to subscribe to messages by using TCP client SDK for Java.

## Procedure

1. Run the following sample code to test the message subscription feature. Set parameters based on the descriptions before you run the code.

```
import com.aliyun.openservices.ons.api.Action;
import com.aliyun.openservices.ons.api.ConsumeContext;
import com.aliyun.openservices.ons.api.Consumer;
import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.MessageListener;
import com.aliyun.openservices.ons.api.ONSFactory;
import com.aliyun.openservices.ons.api.PropertyKeyConst;
import java.util.Properties;
public class ConsumerTest {
    public static void main(String[] args) {
        Properties properties = new Properties();
        // The group ID that you created in the console.
        properties.put(PropertyKeyConst.GROUP_ID, "XXX");
        // The AccessKey ID used for identity authentication.
        properties.put(PropertyKeyConst.AccessKey, "XXX");
        // The AccessKey secret used for identity authentication.
        properties.put(PropertyKeyConst.SecretKey, "XXX");
        // The TCP endpoint. To obtain the endpoint, log on to the Message Queue for Ap
ache RocketMQ console. In the left-side navigation pane, click Instance Details. In the
upper part of the Instance Details page, select your instance. On the Instance Informat
ion tab, view the endpoint in the Obtain Endpoint Information section.
        properties.put(PropertyKeyConst.NAMESRV_ADDR,"XXX");
        Consumer consumer = ONSFactory.createConsumer(properties);
        consumer.subscribe("TopicTestMQ", "*", new MessageListener() {
            public Action consume(Message message, ConsumeContext context) {
                System.out.println("Receive: " + message);
                return Action.CommitMessage;
            }
        });
        consumer.start();
        System.out.println("Consumer Started");
    }
}
```

After you run the code, you can check whether the consumer is started in the Message Queue for Apache RocketMQ console. This operation checks whether the message subscription is successful.

2. Log on to the Message Queue for Apache RocketMQ console.
3. In the left-side navigation pane, click **Groups**.
4. Find the group ID of the consumer whose subscription you want to view, and click **Subscription** in the **Actions** column.

   If the value of **Online** is **Yes**, the consumer has been started and the subscription is successful. Otherwise, the subscription fails.

# 1.1.4. Message types

## 1.1.4.1. Normal messages

This topic describes the definition of normal messages and provides the sample code for sending and receiving normal messages.

In Message Queue for Apache RocketMQ, normal messages do not have special features. They are different from Scheduled messages and delayed messages, Ordered messages, and Transactional messages. The following pieces of sample code provide examples on how to use SDK for different programming languages to send and receive normal messages:

- Java
  - Overview
  - Send messages in multiple threads
  - Subscribe to messages
- C/C++
  - Send and receive normal messages
- .NET
  - Send and subscribe to normal messages

# 1.1.4.2. Scheduled messages and delayed messages

This topic introduces the concepts related to scheduled messages and delayed messages and describes the scenarios, usage, and usage notes of the messages in Message Queue for Apache RocketMQ.

## Concepts

- Scheduled message: A producer sends a message to the Message Queue for Apache RocketMQ broker. However, the producer wants the message to be delivered to the consumer at a specific point in time after the current time instead of being immediately delivered after the broker receives the message. In this case, the message is a scheduled message.
- Delayed message: A producer sends a message to the Message Queue for Apache RocketMQ broker. However, the producer wants the message to be delivered to the consumer after a specific period of time instead of being immediately delivered to the consumer after the broker receives the message. In this case, the message is a delayed message.

Scheduled messages and delayed messages are slightly different in code, but they can achieve the same purpose. Such a message will not be immediately delivered to a consumer after the message is sent to the Message Queue for Apache RocketMQ broker. The message will be delivered to the consumer after a period of time specified in the corresponding attribute of the message.

## Scenarios

Scheduled messages and delayed messages can be used in the following scenarios:

- A time window between message production and consumption is required. For example, when a transaction order is created on an e-commerce platform, a producer sends a delayed message to the Message Queue for Apache RocketMQ broker. A delay of 30 minutes is specified for the message to be delivered to a consumer. The message is used to ask the consumer to check whether the order is paid. If the order is unpaid, the consumer closes the order. If the order is paid, the consumer ignores the message.
- Scheduled messages are sent to trigger scheduled tasks. For example, a notification message is sent to a user at a specified point in time.

## Usage

Scheduled messages and delayed messages are slightly different in code:

- For a scheduled message, a point in time after the message sending time must be specified as the message delivery time.
- For a delayed message, a period of time after the message sending time must be specified. The message will be delivered after the specified period of time elapses.

## Usage notes

- The msg.setStartDeliverTime parameter for a scheduled message or delayed message must be set to a point in time or a period of time after the current timestamp. Unit: milliseconds. If the specified point in time or period of time is earlier than the current timestamp, the message will be immediately delivered to the consumer.
- The msg.setStartDeliverTime parameter for a scheduled message or delayed message can be set to a point in time or a period of time within 40 days. Unit: milliseconds. If the specified point in time or period of time is not within the 40 days, the message cannot be sent.
- The StartDeliverTime parameter specifies the time when the Message Queue for Apache RocketMQ broker starts to deliver the message to the consumer. If messages have been accumulated for the consumer, the scheduled message or delayed message will be queued after the accumulated messages, and will not be delivered at the specified time.
- Due to the potential time difference between the producer and the broker, time differences may also occur between the actual delivery time and the delivery time specified in the producer.
- Scheduled messages and delayed messages can be retained for at most three days on the Message Queue for Apache RocketMQ broker. For example, a message is scheduled to be delivered after five days. If it is not delivered after the fifth day, the message will be deleted on the eighth day.

## References

For more information about the sample code on how to send and receive scheduled messages and delayed messages, see the following documentation:

TCP:

- Java
  - Send and receive scheduled messages
  - Send and receive delayed messages
- C++
  - Send and receive scheduled messages
- .NET
  - Send and subscribe to scheduled messages

# 1.1.4.3. Transactional messages

This topic introduces the concepts related to transactional messages and describes the scenarios, usage, and usage notes of the messages in Message Queue for Apache RocketMQ.

## Concepts

- Message Queue for Apache RocketMQ provides a distributed transaction processing feature that is similar to X/Open XA to ensure transaction consistency by using transactional messages.

- Half message: A half message is a message that cannot be delivered temporarily. When a message is sent to the Message Queue for Apache RocketMQ broker, but the broker does not receive the second acknowledgment (ACK) for the message from the producer, the message is then marked as "temporarily undeliverable". The message in this state is called a half message.
- Message status check: The second ACK for a transactional message may be lost if network connection is unavailable or the corresponding producer application is restarted. When the Message Queue for Apache RocketMQ broker finds that a message remains as a half message for a long time, the broker will send a request to the producer to query whether the final status of the message is Commit or Rollback.

## Scenarios

Transactional messages in Message Queue for Apache RocketMQ can be used in the following scenarios:

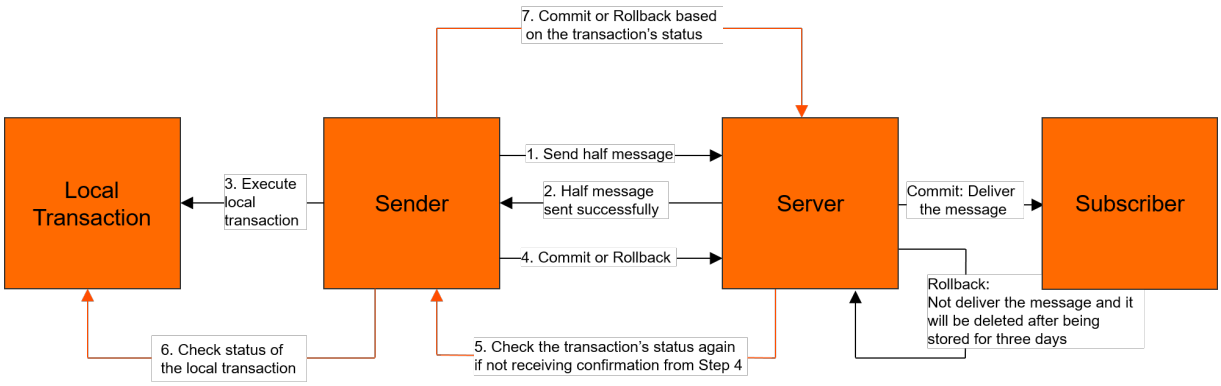
Message Queue for Apache RocketMQ provides a distributed transaction processing feature that is similar to X/Open XA to ensure transaction consistency by using transactional messages.

In the process of using the shopping cart to place an order, the user entry point is the shopping cart system, and the entry point for order placement is the transaction system. The data in the two systems must be eventually consistent. In this case, transactional messages can be used. After an order is placed, the transaction system sends a transaction order message to Message Queue for Apache RocketMQ. The shopping cart system subscribes to the transaction order message from Message Queue for Apache RocketMQ, performs corresponding business steps, and updates the shopping cart data.

## Usage

Interaction process

The following figure shows the interaction process of transactional messages in Message Queue for Apache RocketMQ.




The procedure for sending a transactional message includes the following steps:

1. A producer sends a half message to the Message Queue for Apache RocketMQ broker.
2. The Message Queue for Apache RocketMQ broker persists the message and sends an ACK to the producer to confirm that the message is sent. At this time, the message is a half message.
3. The producer starts a local transaction.
4. The producer sends a second ACK to the Message Queue for Apache RocketMQ broker to submit the execution result of the local transaction. The execution result is Commit or Rollback. If the broker receives Commit, the broker marks the half message as deliverable, and the consumer will receive the half message. If the broker receives Rollback, the broker deletes the half message, and

the consumer will not receive the half message.

The procedure for checking the status of a transactional message includes the following steps:

- If network connection is unavailable or the corresponding producer application is restarted, the Message Queue for Apache RocketMQ broker may not receive the ACK in Step 4. After a specific period of time, the Message Queue for Apache RocketMQ broker sends a request to query the status of the half message.
- After the producer receives the request, the producer checks the final status of the local transaction that corresponds to the half message.
- The producer sends another ACK to the Message Queue for Apache RocketMQ broker based on the final status of the local transaction. The broker processes the half message by following Step 4.

## Usage notes

1. Transactional messages cannot share group IDs with other types of messages. Transactional messages allow message status check, whereas other types of messages do not support this feature. The Message Queue for Apache RocketMQ broker can query the status of a transactional message on the consumer by group ID.
2. You must specify the implementation class of LocalTransactionChecker when you create the producer of transactional messages by executing ONSFactory.createTransactionProducer. This way, the status of the transactional messages can be checked when exceptions occur.
3. When a transactional message is sent to execute the local transaction, one of the following three states is returned in the execute method:
   - TransactionStatus.CommitTransaction: The transaction is committed. The consumer can consume the message.
   - TransactionStatus.RollbackTransaction: The transaction is rolled back. The message is discarded and cannot be consumed.
   - TransactionStatus.Unknow: The status of the transaction is unknown. The Message Queue for Apache RocketMQ broker is expected to send a request to the producer to query the status of the local transaction that corresponds to the message.

## Sample code

- JavaSend and subscribe to transactional messages
- C/C++Send and receive transactional messages
- .NETSend and receive transactional messages

# 1.1.4.4. Ordered messages

This topic introduces the concepts related to ordered messages and describes the scenarios and usage notes of the messages in Message Queue for Apache RocketMQ.

## Concepts

Ordered messages, also known as first-in-first-out (FIFO) messages, are a type of message provided by Message Queue for Apache RocketMQ. Ordered messages are published and consumed in FIFO order.

An ordered message involves ordered publishing and ordered consumption.

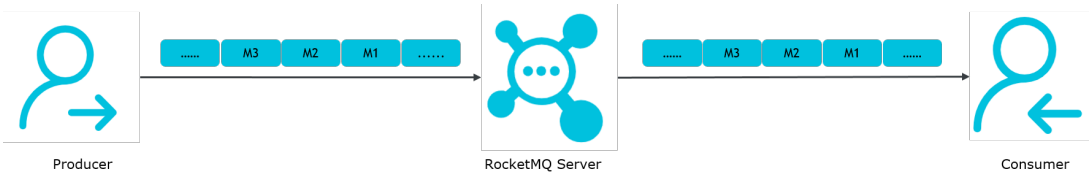
- **Ordered publishing**: A producer sends messages to a specified topic in FIFO order.
- **Ordered consumption**: A consumer receives messages of a specified topic in FIFO order. A message

that is first sent will be first received by the consumer.

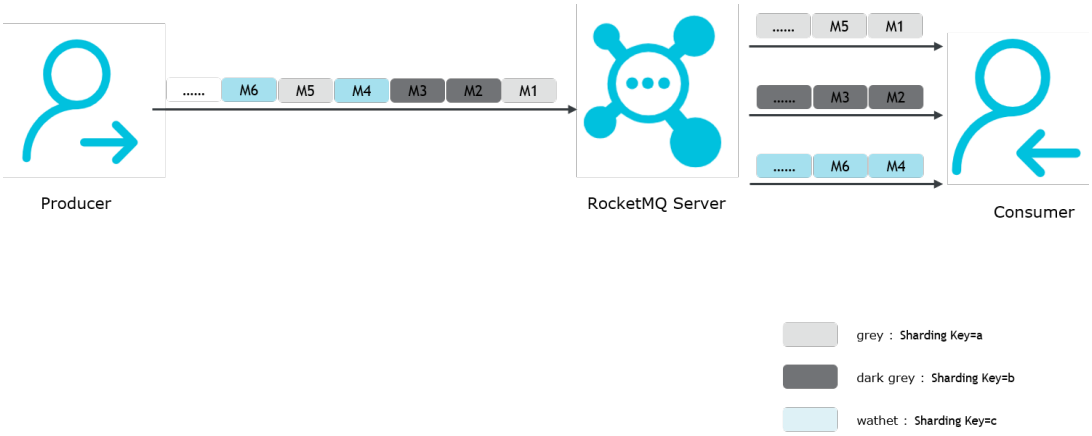
Ordered messages are classified into globally ordered messages and partitionally ordered messages.

- **Globally ordered messages**: All messages of a specified topic are published and consumed in FIFO order.
- **Partitionally ordered messages**: All messages of a specified topic are partitioned by using a partition key. The messages in each partition are published and consumed in FIFO order. A partition key is a key field that is used for ordered messages to distinguish among different partitions. It is completely different from the key of a normal message.

  A globally ordered message




  A partitionally ordered message




## Scenarios

**Globally ordered messages** can be used in the following scenarios:

Performance requirements are not demanding, and all messages are published and consumed in FIFO order.

**Partitionally ordered messages** can be used in the following scenarios:

Performance requirements are demanding, the partition key is used as the partitioning field, and messages in one partition are published and consumed in FIFO order.

Examples:

- [Example 1]

  If a verification code needs to be sent for user registration, the user ID can be used as the partition key. Messages sent by the same user will be published and subscribed to in FIFO order.

- [Example 2]

  The order ID is used as the partition key for orders on an e-commerce platform. Then, order creation messages, order payment messages, order refund messages, and logistics messages of the same order are published and subscribed to in FIFO order.

All internal e-commerce systems of Alibaba Group use partitionally ordered messages. This ensures both service order and performance.

## Comparison between globally ordered messages and partitionally ordered messages

Different types of topics are used to create different types of messages in the Message Queue for Apache RocketMQ console. The following table compares various types of topics.

Message types

| Topic type | Support for transactional messages | Support for scheduled messages | Performance |
| --- | --- | --- | --- |
| Unordered messages including normal, transactional, scheduled, and delayed messages | Yes | Yes | Highest |
| Partitionally ordered messages | No | No | High |
| Globally ordered messages | No | No | Medium |

Methods of sending messages

| Message type | Support for reliable synchronous transmission | Support for reliable asynchronous transmission | Support for one-way transmission |
| --- | --- | --- | --- |
| Unordered messages including normal, transactional, scheduled, and delayed messages | Yes | Yes | Yes |
| Partitionally ordered messages | Yes | No | No |
| Globally ordered messages | Yes | No | No |

## Usage notes

When you use ordered messages, take note of the following points:

- Ordered messages do not support broadcasting consumption.
- One group ID corresponds to one type of topic. This means that one group ID can be used to send and receive ordered messages or unordered messages.
- Ordered messages cannot be sent in asynchronous mode. Otherwise, the order cannot be ensured.
- For globally ordered messages, we recommend that you create at least two Message Queue for Apache RocketMQ instances. You can run multiple instances at the same time to ensure service

continuity. If the active instance fails, another instance can immediately take over the services. This ensures that services are uninterrupted. Only one instance is active at a time.

### Supported SDK and sample code

Use SDK for Java 1.2.7 or later.

For information about sample code, see the following documentation:

- Java: Send and receive ordered messages
- C and C++: Send and receive ordered messages
- ..NET: Send and subscribe to ordered messages

# 1.1.5. Console guide

## 1.1.5.1. Resource management

## 1.1.5.1.1. Resource management overview

This topic describes how to manage resources in Message Queue for Apache RocketMQ.

If a new application needs to access Message Queue for Apache RocketMQ, you must create the following Message Queue for Apache RocketMQ resources for the application:

- Instance: As a virtual machine (VM) resource of Message Queue for Apache RocketMQ, an instance stores the topics and group IDs of messages.
- Topic: In Message Queue for Apache RocketMQ, a producer sends a message to a specified topic, and a consumer subscribes to the topic to obtain and consume the message.
- Group ID: A group ID is used to identify a group of message consumers or producers.

You can add, delete, modify, and query these resources by using the Message Queue for Apache RocketMQ console or by calling the Message Queue for Apache RocketMQ API.

When you use SDKs to send and subscribe to messages, you must specify the topic and group ID that you created in the Message Queue for Apache RocketMQ console. You must also enter the AccessKey ID and AccessKey secret that you created in the Apsara Uni-manager Management Console for identity authentication.

If you have not obtained the AccessKey ID and AccessKey secret, you can obtain them in the Apsara Uni-manager Management Console. For more information, see Obtain the AccessKey ID and AccessKey secret.

## 1.1.5.1.2. Manage instances

An instance stores topics and group IDs. This topic describes how to create, update, view, and delete instances in the Message Queue for Apache RocketMQ console.

### Create an instance

1. Log on to the Message Queue for Apache RocketMQ console.
2. On the **Instance Details** page, click **Create Instance**.
3. On the **Create Instance** page, enter the information as prompted and click **Submit**.
4. In the **Submitted.** message, click **Back to Console**.

On the **Instance Details** page, you can view the instance that you created.

## Modify the specifications of an instance

You can upgrade or downgrade the specifications of an instance.

1. Log on to the Message Queue for Apache RocketMQ console. In the left-side navigation pane, click **Instance Details**.
2. In the upper part of the **Instance Details** page, click the name of the instance whose configurations you want to modify. Then, click **Update Specifications**.
3. On the **Update Configurations** page, set **Maximum Topics**, **Outbound Message TPS**, and **Inbound Message TPS**, and **Description**.

   > **Note** The values that you specify must be in the allowed ranges displayed on the page.

4. Click **Submit**.
5. In the **Submitted.** message, click **Back to Console**.
   On the **Instance Details** page, you can view the latest specifications of the instance.

## View details about an instance

1. Log on to the Message Queue for Apache RocketMQ console. In the left-side navigation pane, click **Instance Details**.
2. In the upper part of the **Instance Details** page, click the name of the instance that you want to view.

## Delete an instance

**Prerequisites**

- You have deleted all resources, including topics and group IDs, from the instance.
- No Message Queue for MQTT instance is bound to the instance.

1. Log on to the Message Queue for Apache RocketMQ console. In the left-side navigation pane, click **Instance Details**.
2. In the upper part of **Instance Details** page, click the name of the instance that you want to delete. Then, click **Delete Instance**.
3. In the **Note** message, read the note carefully. If you are sure to delete the instance, click **OK**.
   The message **The instance is deleted.** appears.

## References

If you need to call the Message Queue for Apache RocketMQ API to perform relevant operations, follow the instructions provided in *Message Queue for Apache RocketMQ Developer Guide*.

# 1.1.5.1.3. Manage topics

Topic is the first-level identifier that classifies messages in Message Queue for Apache RocketMQ. For example, you can create a topic named Topic_Trade to identify transaction-specific messages. This topic describes how to create, update, view, and delete topics in the Message Queue for Apache RocketMQ console.

## Prerequisites

Create an instance

## Create a topic

1. Log on to the Message Queue for Apache RocketMQ console. In the left-side navigation pane, click **Topics**.
2. In the upper part of the **Topics** page, select your instance.
3. Click **Create Topic**.
4. In the **Create Topic** dialog box, enter a name for the topic in the **Topic** field.

   > **Notice** The topic name must be unique in the instance where you create the topic and must comply with the following rules:
   >
   > - The topic name cannot start with CID or GID, because CID and GID are reserved fields for group IDs.
   > - The topic name can contain only letters, digits, hyphens (-), and underscores (_).
   > - The topic name must be 3 to 64 characters in length.

5. From the **Message Type** drop-down list, select a value. This value defines the type of message that this topic sends and receives.

   We recommend that you create different topics to send different types of messages. For example, create Topic A for normal messages, Topic B for transactional messages, and Topic C for scheduled messages or delayed messages. For more information about message types, see Message types.

6. In the **Description** field, enter a description about the topic and click **OK**.
   The created topic appears in the topic list.

## Modify the description of a topic

1. Log on to the Message Queue for Apache RocketMQ console. In the left-side navigation pane, click **Topics**.
2. In the upper part of the **Topics** page, select your instance.
3. In the topic list, find the topic whose description you want to modify and click the icon in the **Description** column.
4. In the **Edit Topic** dialog box, enter the new description and click **OK**.
   The message **The operation is successful.** appears.

## View topic information

1. Log on to the Message Queue for Apache RocketMQ console. In the left-side navigation pane, click **Topics**.
2. In the upper part of the **Topics** page, click the name of your instance. You can view all topics in the instance and the details about a specific topic. The details include the subscription, permissions, and message type.

## Delete a topic

> **Note** After a topic is deleted, producers that send messages to the topic and consumers that subscribe to the topic immediately stop services and all resources are deleted within 10 minutes. Proceed with caution.

1. Log on to the Message Queue for Apache RocketMQ console. In the left-side navigation pane, click **Topics**.
2. In the upper part of the **Topics** page, select your instance.
3. Find the topic that you want to delete, click the icon, and then select Delete.
4. In the **Caution** message, read the prompt carefully. If you are sure to delete the topic, click **OK**. The topic no longer appears in the topic list in the instance.

### References

If you need to call the Message Queue for Apache RocketMQ API to perform relevant operations, follow the instructions provided in *Message Queue for Apache RocketMQ Developer Guide*.

## 1.1.5.1.4. Manage groups

After you create an instance and a topic, you must create a group ID for the message consumer or producer. This topic describes how to create, view, and delete group IDs in the Message Queue for Apache RocketMQ console.

### Prerequisites

Create an instance

### Context

A group ID is required for consumers but is optional for producers.

### Create a group ID

1. Log on to the Message Queue for Apache RocketMQ console. In the left-side navigation pane, click **Groups**.
2. In the upper part of the **Groups** page, select your instance.
3. Click **Create Group ID**.
4. In the **Create Group ID** dialog box, enter a group ID and description. Then, click **OK**.

> **Notice**
>
> - The group ID must comply with the following rules:
>   - The group ID must start with GID_ or GID- and can contain only letters, digits, hyphens (-), and underscores (_).
>   - The group ID must be 7 to 64 characters in length.
>   - The group ID cannot be modified after it is created.
> - The group ID must be unique in the instance where you create the group ID.
> - Group IDs have an N:N relationship with topics. A consumer can subscribe to multiple topics and a topic can be subscribed to by multiple consumers. A producer can send messages to multiple topics and a topic can receive messages from multiple producers.
> - If a group ID for consumers or an existing consumer ID is created by using a Resource Access Management (RAM) user, the RAM user and its Apsara Stack tenant account have the permissions to use this group ID or consumer ID.
> - If a group ID for consumers or an existing consumer ID is created by using an Apsara Stack tenant account, only the Apsara Stack tenant account has the permissions to use this group ID or consumer ID. The RAM users of this Apsara Stack tenant account do not have the permissions to use this group ID or consumer ID.
> - For more information about changes from existing consumer IDs and producer IDs to group IDs, see Updates.

The created group ID appears in the group ID list.

## View group ID information

1. Log on to the Message Queue for Apache RocketMQ console. In the left-side navigation pane, click **Groups**.
2. In the upper part of the **Groups** page, click the name of your instance. You can view all group IDs in the instance and the details about a specific group ID. The details include the subscription, permissions, and consumer status.

## Delete a group ID

> **Note** After a group ID is deleted, the consumer instances identified by the group ID immediately stop receiving messages and all related resources are deleted within 10 minutes. Proceed with caution.

1. Log on to the Message Queue for Apache RocketMQ console. In the left-side navigation pane, click **Groups**.
2. In the upper part of the **Groups** page, select your instance.
3. Find the group ID that you want to delete, click the icon, and then select Delete.
4. In the **Caution** message, read the prompt carefully. If you are sure to delete the group ID, click **OK**. The group ID no longer appears in the group ID list in the instance.

## References

If you need to call the Message Queue for Apache RocketMQ API to perform relevant operations, follow the instructions provided in Message Queue for Apache RocketMQ Developer Guide.

# 1.1.5.2. Message query

## 1.1.5.2.1. Overview

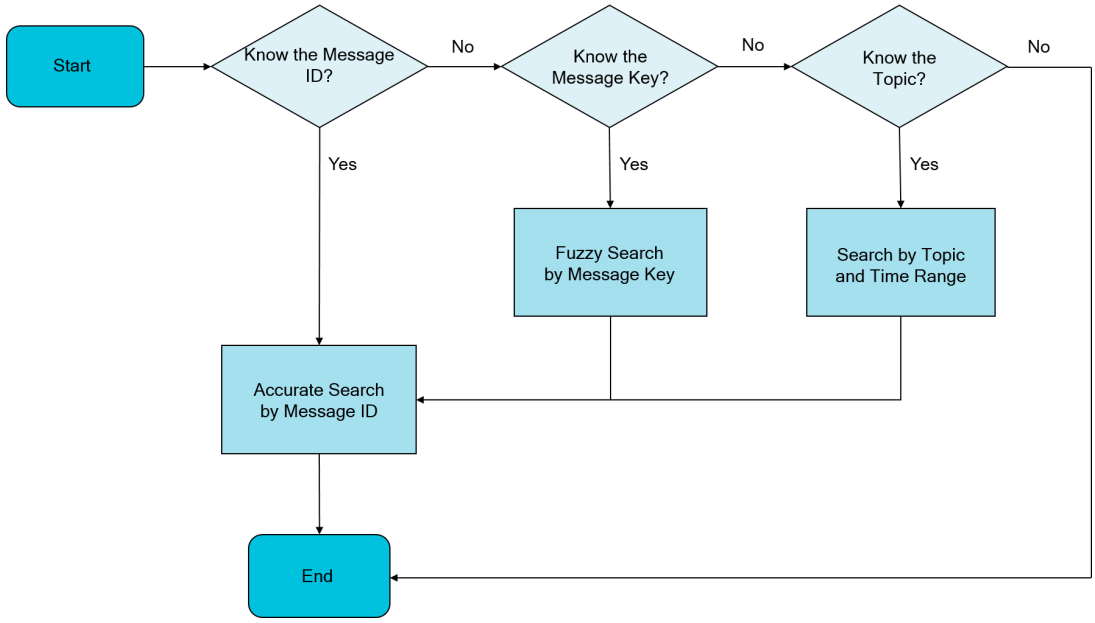
If a message is not consumed as expected, you can query the message content to troubleshoot problems. Message Queue for Apache RocketMQ allows you to query messages by message ID, by message key, and by topic.

Comparison of query methods

| Method | Condition | Type | Description |
| --- | --- | --- | --- |
| By message ID | Topic+Message ID | Exact match | You can specify a topic and a message ID to obtain a message and its attributes. |
| By message key | Topic+Message Key | Fuzzy match | You can specify a topic and a message key to query the 64 messages that are most recently sent and contain the specified message key. We recommend that you specify a unique key for each message in producers whenever possible to ensure that the number of messages with the same key does not exceed 64. Otherwise, some messages cannot be queried. |
| By topic | Topic and time range | Range match | You can specify a topic and a time range to query all messages that meet the specified conditions. This type of query returns a large number of messages. It is difficult to find a specific message that you want to query. |

We recommend that you query messages by using the following process.

Message query process



# 1.1.5.2.2. Query messages

This topic describes how to query messages in the Message Queue for Apache RocketMQ console by using three different methods.

1. Log on to the Message Queue for Apache RocketMQ console.
2. In the left-side navigation pane, click **Message Query**.
3. On the **Message Query** page, click a tab. On the tab that appears, enter the information and click **Search** to query messages.
   - **By message ID**

     If you query messages by message ID, exact match is used. You can specify a topic and a message ID to query a message by using exact match. Therefore, we recommend that you print the message ID to the log to facilitate troubleshooting after the message is sent.

     In the following sample code, SDK for Java is used to obtain a message ID:

     ```
     SendResult sendResult = producer.send(msg);
     String msgId = sendResult.getMessageId();
     ```

     To obtain the sample code for other programming languages, click **Groups** in the left-side navigation pane. On the Groups page, find the group ID of the message and click **Sample Code** in the Actions column.

   - **By message key**

     Message Queue for Apache RocketMQ creates an index for messages based on the message keys that you specify. When you enter a topic name and a message key to query messages, Message Queue for Apache RocketMQ returns the matched messages based on the index.

     > **Notice**
     >
     > If you query messages by message key, take note of the following points:
     >
     > - The query condition is the specified message key.
     > - Only the 64 messages that are most recently sent and contain the specified message key are returned. Therefore, we recommend that you specify a unique and business-distinctive key for each message.

     The following sample code provides an example on how to specify a message key:

     ```
     Message msg = new Message("Topic","*","Hello MQ".getBytes());
     /**
     * Specify the key to be indexed for each message. The key value is the key attribute of the message. We recommend that you specify a unique key for each message.
     * If you do not receive a message as expected, you can query the message in the Message Queue for Apache RocketMQ console. Messages can be sent and received even if this attribute is not specified.
     */
     msg.setKey("TestKey"+System.currentTimeMillis());
     ```

   - **By topic**

If you cannot query messages by message ID or message key, query messages by topic. You can specify a topic and time range for message sending, retrieve messages in batches, and then find the data that you need.

> **Notice**
>
> If you query messages by topic, take note of the following points:
>
> - If you specify a topic and time range to query messages, range match is used to retrieve all messages that meet the time condition within the topic. The number of retrieved messages is large. Therefore, we recommend that you narrow down the time range.
> - If you query messages by topic, a large number of messages are returned on multiple pages.

# 1.1.5.2.3. Query results

This topic describes the results returned when you query messages.

You can view the queried messages on the **Message Query** page of the Message Queue for Apache RocketMQ console. The displayed information includes the message ID, tag, key, and storage time. In addition, click the corresponding buttons in the Actions column of a message to download the message content, Query the message trace, and view the message details.

The delivery status is calculated by Message Queue for Apache RocketMQ based on the consumption progress of each group ID. For more information about the delivery status, see Message delivery status.

> **Note** The delivery status is estimated based on the consumption progress. Use the message tracing feature to query the consumption details. The message tracing feature allows you to query the complete trace of a message. For more information, see Query the message trace.

Message delivery status

| Delivery status | Possible cause |
| --- | --- |
| The message has been subscribed to and consumed at least once. | The group ID has properly consumed the message. |
| The message has been subscribed to but is filtered out by the filter expression. Check the tag of the message. | The tag of the message does not comply with the subscription of the consumer and the message is filtered out. To query the subscription of the consumer, log on to the Message Queue for Apache RocketMQ console. In the left-side navigation pane, click **Groups**. On the Groups page, find the group ID of the consumer whose subscription you want to view and click **Consumer Status** in the **Actions** column. |
| The message has been subscribed to but is not consumed. | A consumer identified by the group ID has subscribed to the message, but the message has not been consumed possibly because the consumption is slow or is blocked due to an exception. |

| Delivery status | Possible cause |
| --- | --- |
| The message has been subscribed to but the consumer that subscribes to the message and is identified by the group ID is not online. Use the message tracing feature to query the details about the message in an exact match. | A consumer identified by the group ID has subscribed to the message but the consumer is not online. Check the reason why the consumer is not online. |
| An unknown exception occurred. | Contact the customer service. |

Message Queue for Apache RocketMQ provides the consumption verification feature. You can push a specified message to a specified online consumer to check whether the consumer can consume the message based on the correct logic as expected.

> **Notice** The consumption verification feature is used only to verify whether consumers can consume messages based on the correct logic as expected. This feature does not affect the normal process of receiving messages. Therefore, information such as the consumption status of a message does not change after the consumption is verified.

# 1.1.5.3. Message tracing

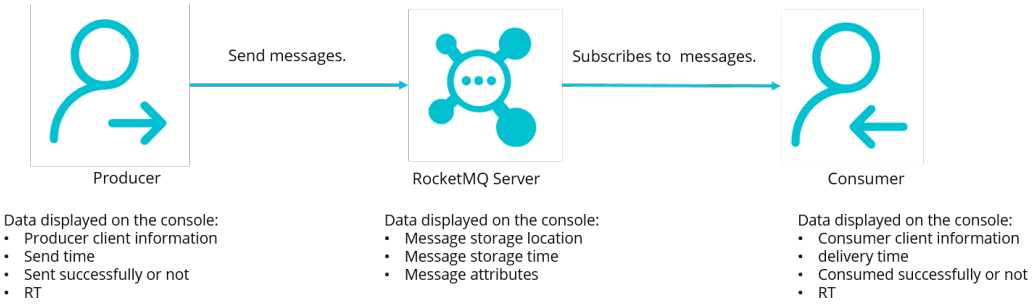
# 1.1.5.3.1. Overview

A message trace is the complete trace of a message that is sent from a producer to the Message Queue for Apache RocketMQ broker and then consumed by a consumer. The message trace includes the time, status, and other information of each node in the preceding process. The message trace provides robust data support for troubleshooting in production environments. This topic describes the scenarios, query procedure, and parameters of query results for message tracing.

## Message trace data

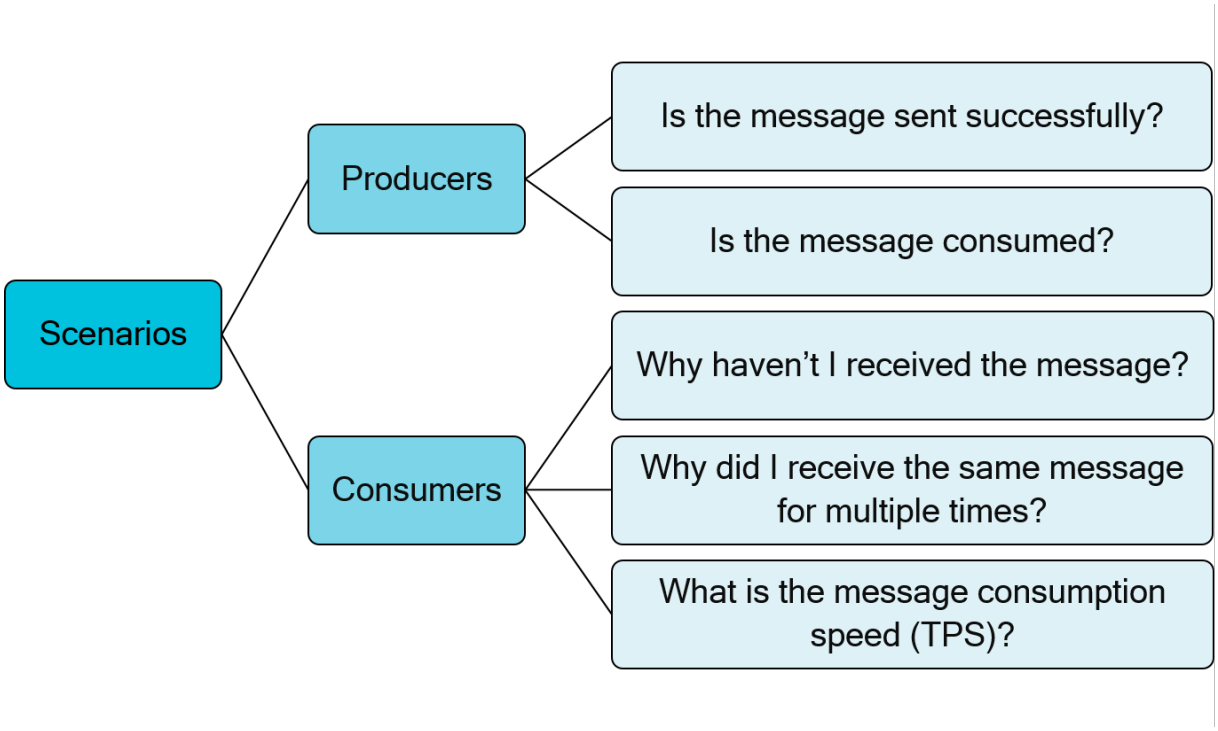
In Message Queue for Apache RocketMQ, the complete trace of a message involves three roles: producer, broker, and consumer. Each role adds relevant information to the trace when the role processes the message. The information is aggregated to indicate the status of the message. The following figure shows the relevant data.

Message trace data




## Scenarios

You can use the message tracing feature to troubleshoot problems if a message is not sent or received as expected in your production environment. You can query the message trace by message ID, message key, or topic to check whether the message is sent and received as expected within the specified time range.




## Usage notes

No extra fees are incurred when you use the message tracing feature. After a message is sent, you can query the trace of the message based on the ID or key of the message in the Message Queue for Apache RocketMQ console. You must take note of the following points.

Rules for querying message traces

| Message type | Query description |
| --- | --- |
| Normal messages | A sending trace is generated after the message is sent. If the message is not consumed, **Not Consumed** appears. After the message is consumed, the delivery and consumption information appears. |
| Ordered messages | A sending trace is generated after the message is sent. If the message is not consumed, **Not Consumed** appears. After the message is consumed, the delivery and consumption information appears. |
| Scheduled messages and delayed messages | If the current system time does not reach the specified message consumption time, the trace can be queried but the message cannot be queried. |
| Transactional messages | Before the transaction is committed, the trace can be queried but the message cannot be queried. |

## Examples

If you find that a message is not received as expected based on the log information, perform the following steps to troubleshoot the problem by using the message trace:

1. Collect the information about the message that is suspected to be abnormal. The information includes the message ID, message key, topic, and approximate sending time.
2. Log on to the Message Queue for Apache RocketMQ console, and create a query task to query the message trace based on the available information.
3. Check the query results and analyze the cause.
   - If **Not Consumed** appears in the trace, go to the **Groups** page to View the consumer status. Then, you can determine whether message accumulation is the reason why the message is not consumed.
   - If the message is consumed, find the corresponding consumer and the time when the message is consumed in the consumption information. Then, log on to the consumer to view the relevant log.

# 1.1.5.3.2. Query message traces

No extra fees are incurred when you use the message tracing feature. To use this feature, you must make sure that the version of your client SDK supports this feature. After a message is sent or received, you can query the trace of the message based on the message attributes in the Message Queue for Apache RocketMQ console.

## Prerequisites

Make sure that the version of your SDK supports the message tracing feature. You can use the following versions of SDKs:

- SDK for Java: V1.2.7 and later
- SDK for C and C++: V1.1.2 and later
- .SDK for .NET: V1.1.2 and later

## Procedure

1. Log on to the Message Queue for Apache RocketMQ console. In the left-side navigation pane, click **Message Tracing**. On the page that appears, click **Create Query Task** in the upper-right corner.
2. In the **Create Query Task** dialog box, click the By Message ID, By Message Key, or By Topic tab and enter the information as prompted. Then, click **OK**.

   > **Notice** Specify a time range as accurate as possible to narrow the query scope and speed up the query.

   Message tracing supports the following three query methods. Select a query method and specify the query criteria.

   - By message ID: You must specify the unique message ID, topic, and approximate sending time of a message. We recommend that you use this method because this method uses exact match and allows you to query message traces fast.
   - By message key: You must specify the message key, topic, and approximate sending time of a message. This method uses fuzzy match. A maximum of 1,000 traces can be displayed for a

query based on the specified message key. This method applies only to scenarios where the message ID is not recorded but a business-distinctive message key is specified.

- By topic: You must specify the topic and approximate sending time for batch query. This method uses range match and applies to scenarios where both the message ID and the message key are unavailable and the message volume is small. We do not recommend this query method, because a large volume of messages exist in a topic within the specified time range and you cannot find the message you want among these messages in this time range.

After you create a query task, you can view the query task on the **Message Tracing** page. If the value of **Task Status** is **Querying**, you cannot view the message trace.

3. In the upper-right corner, click **Refresh** until the value of Task Status becomes **Query Completed**. You can click the **+** icon to view the brief trace information, including the message attributes and consumption status.
   - If no data is found, verify whether the query information you entered is valid.
   - If the trace is queried out, brief trace information appears, including the message attributes and consumption status.
4. Click **View Traces** to check the complete trace.

   The message trace consists of three parts:

   - Producer information
   - Topic information
   - Consumer information

   You can move the pointer over a field to view the details about the field.

   If you query traces by message key or topic, multiple traces may be displayed. You can page up and down to view and compare the traces.

   For more information about the query results of message traces, see Message tracing status.

# 1.1.5.3.3. Status in message traces

This topic describes the terms and status information displayed on the Message Trace page.

Terms for message tracing

| Role | Field | Description |
| --- | --- | --- |
| Producer | Sending Time | The time when the message was sent from the producer. The time follows the ISO 8601 standard in the yyyy-MM-ddThh:mm:ssZ format. The time is displayed in UTC. |
| | Time Consumed | The period of time that the producer took to send a message by calling the Send method. Unit: milliseconds. |
| | | |

| Role | Field | Description |
| --- | --- | --- |
| Topic | Region | The region where the message is stored or the region where the consumer is located. |
| Consumer | Time Consumed | The period of time that the consumer took to execute the consumeMessage method after the message is pushed to the consumer. |
| | Delivery Time | The time when the consumer executed the consumeMessage method to start consuming the message. The time follows the ISO 8601 standard in the yyyy-MM-ddThh:mm:ssZ format. The time is displayed in UTC. |

Sending status and consumption status

| Sending status and consumption status | Field | Description |
| --- | --- | --- |
| Sending status | Sent | The message is sent and stored in the Message Queue for Apache RocketMQ broker. |
| | Sending Failed | The message fails to be sent and is not stored in the Message Queue for Apache RocketMQ broker. In this case, the broker tries to redeliver the message. |
| | Message Standing By | The message is a scheduled or delayed message and it is not the time to deliver the message. |
| | Transaction Uncommitted | The message is a transactional message and has not been committed. |
| | Message Rolled Back | The message is a transactional message and has been rolled back. |
| | All Succeeded | The message has been consumed by all the consumers to which it is delivered. |

| Sending status and consumption status | Field | Description |
| --- | --- | --- |
| Consumption status | Partially Succeeded | The message fails to be consumed in specific deliveries, or the message is consumed after it is redelivered. |
| | All Failed | The message still fails to be consumed after all delivery retries. |
| | Not Consumed | The message is not delivered to consumers. |
| | Consumption Result Unreturned | No results are returned for the message consumption method or the method is interrupted. Therefore, the consumption status is not returned to the Message Queue for Apache RocketMQ broker. |
| | Consumed | The message is consumed. |
| | Consumption Failed | A failure result is returned for the message consumption method, or the method threw an exception. |

# 1.1.5.4. View the consumer status

The Message Queue for Apache RocketMQ console allows you to check the consumer status to troubleshoot exceptions that occur during message consumption. This feature allows you to view the information about a group ID or a consumer identified by the group ID. The information includes the connection status, subscription, consumption TPS, number of accumulated messages, and thread stacks. This topic describes how to view the information.

## Context

The cause of an exception that occurs during message consumption is complicated. In most cases, the consumer status information in the console alone is insufficient to troubleshoot a problem. You must perform further troubleshooting by analyzing logs and business scenarios.

## Scenarios

You can query the consumer status for troubleshooting in the following scenarios:

- Subscription inconsistency
  - Symptom: In the **Consumer Status** panel, the value of **Consistent Subscription** column is **No** for the group ID.
  - Solution: For more information about how to handle subscription inconsistency, see Subscription inconsistency.

- Message accumulation alerts
  - Symptom: In the **Consumer Status** panel, the value in the **Accumulated Messages** column is large for the group ID.
  - Solution: For more information about how to handle message accumulation alerts, see Message accumulation.

## View the information about a group ID

1. Log on to the Message Queue for Apache RocketMQ console.
2. In the left-side navigation pane, click **Groups**.
3. On the **Groups** page, find the group ID that you want to view and click **Consumer Status** in the **Actions** column.

   The following table describes the fields in the **Consumer Status** panel.

   Description of fields in the Consumer Status panel

| Field | Description |
| --- | --- |
| Online status icon | The value is **Yes** if one consumer instance identified by the group ID is online. In this case, you can view information about all online consumer instances in the **Connection Information** section. If none of the consumer instances identified by the group ID is online, the value is **Offline** and no information is displayed in the **Connection Information** section. |
| Consistent Subscription | Indicates whether the subscriptions of all consumer instances identified by the group ID are consistent. For more information about subscription consistency, see Subscription consistency. |
| Real-time Consumption Speed | The total TPS at which messages are received by the consumer instances identified by the group ID. Unit: messages/s. |
| Real-time Accumulated Messages | The total number of messages that are not consumed by the consumer instances identified by the group ID. |
| Last Consumed At | The time when the consumer instances identified by the group ID last consumed a message. |
| Message Delay Time | The difference between the production time of the earliest unconsumed message and the current time. |

## View information about a single consumer instance identified by a specific group ID

1. If the online status of the group ID is **Yes**, you can view information about each online consumer instance identified by the group ID in the **Connection Information** section. The information

includes the client ID, host or public IP address, current process ID, and number of accumulated messages.

2. If you want to view more information about a specific consumer instance, click **Detailed Information** in the **Detailed Description** column. The information includes the number of consumer threads, consumption start time, subscription, and message consumption statistics.
3. If you want to view the stack information of the current process for a specific consumer instance, find the consumer instance and click **Stack Information** in the **Stack Information** column.

# 1.1.5.5. Reset consumer offsets

You can reset consumer offsets to skip the accumulated or undesired messages instead of consuming them, or to consume messages sent after a point in time regardless of whether the messages sent before this point in time are consumed.

## Context

When you reset consumer offsets, take note of the following points:

- You cannot reset consumer offsets in broadcasting consumption mode.
- You cannot reset consumer offsets by specifying a message ID, message key, or tag.

## Procedure

1. Log on to the Message Queue for Apache RocketMQ console. In the left-side navigation pane, click **Groups**.
2. Find the group ID whose consumer offset you want to reset, click the More icon in the **Actions** column, and then select **Reset Consumer Offset**.
3. In the **Reset Consumer Offset** dialog box, enter the corresponding topic in the **Topic** field, and then select one the following options as needed:
   - **Consumption from Latest Offset (All Accumulated Messages Cleared)**: If this option is selected, consumers identified by the group ID skip all accumulated (unconsumed) messages within the topic and restart consumption from the latest offset.

     If "reconsumeLater" is returned, the messages in the delivery retry process cannot be skipped.
   - **Consumption from a Specific Point in Time**: If this option is selected, a time picker appears. Select a point in time. Only the messages that are sent after the selected point in time will be consumed.

     The period allowed for the time picker ranges from the production time of the earliest message stored in the topic to the production time of the latest message stored in the topic. You can select a point in time only within the allowed time range.
4. Click **OK** to reset the consumer offset.

# 1.1.5.6. Dead-letter queues

Dead-letter queues are used to process messages that cannot be consumed as expected. This topic describes how to query, export, and resend dead-letter messages in dead-letter queues. This helps you manage dead-letter messages as needed and prevent missing messages.

## Background information

When a message fails to be consumed for the first time, the

Message Queue for Apache RocketMQ

broker automatically redelivers the message. If the message still cannot be consumed after the broker redelivers the message for a maximum of allowed times, the message cannot be properly consumed. Instead of immediately discarding the message,

Message Queue for Apache RocketMQ

sends it to a particular queue of the corresponding consumer.

In

Message Queue for Apache RocketMQ

, a message that cannot be properly consumed is called a *dead-letter message*, which is stored in a particular queue named *dead-letter queue*.

## Features

Dead-letter messages have the following features:

- They can no longer be consumed by consumers as expected.
- They have a valid period of three days, which is the same as that of normal messages. After the three days, dead-letter messages are automatically deleted. Therefore, process dead-letter messages within three days after they are generated.

Dead-letter queues have the following features:

- A dead-letter queue corresponds to a group ID instead of a consumer instance.
- If no dead-letter message is generated for a group ID,

  Message Queue for Apache RocketMQ

  does not create a dead-letter queue for the group ID.
- A dead-letter queue contains all the dead-letter messages of the corresponding group ID regardless of the message topic.

In the

Message Queue for Apache RocketMQ

console, you can query, export, and resend dead-letter messages.

## Methods of querying dead-letter messages

Message Queue for Apache RocketMQ

provides the following methods for you to query dead-letter messages.

| Method | Condition | Type | Description |
| --- | --- | --- | --- |

| Method | Condition | Type | Description |
| --- | --- | --- | --- |
| By group ID | Group ID and time range | Range match | You can specify a group ID and a time range to query all messages that meet the specified conditions. This type of query returns a large number of messages. It is difficult to find a specific message that you want to query. |
| By message ID | Group ID+Message ID | Exact match | You can specify a group ID and a message ID to query a message by using exact match. |

## By group ID

You can batch query all the dead-letter messages of a group ID within a time range by specifying the group ID and time range.

> **Notice** The production time of a dead-letter message refers to the time when a message is sent to the dead-letter queue after the maximum number of redelivery retries for this message is reached.

1. Log on to the Message Queue for Apache RocketMQ console.
2. In the left-side navigation pane, click **Dead-letter Queues**.
3. On the **Dead-letter Queues** page, click the **By Group ID** tab.
4. From the drop-down list of group IDs, select the group ID whose dead-letter messages you want to view.
5. Click the icon for the time picker and specify the start time and end time.
6. Click **Search**. All dead-letter messages that meet the preceding conditions appear.
7. Find the dead-letter message that you want to view and click **View Details** in the **Actions** column to view the details about the message. The details include the basic attributes, download URL of the message body, message trace, and delivery status.

## By message ID

If you query messages by message ID, exact match is used. You can precisely locate a message by specifying its group ID and message ID.

1. Log on to the Message Queue for Apache RocketMQ console.
2. In the left-side navigation pane, click **Dead-letter Queues**.
3. On the **Dead-letter Queues** page, select your instance and click the **By Message ID** tab.
4. From the drop-down list of group IDs, select the group ID whose dead-letter messages you want to view.
5. In the search box of message IDs, enter the ID of the message that you want to query.

6. Click **Search**. All dead-letter messages that meet the preceding conditions appear.
7. Find the dead-letter message that you want to view and click **View Details** in the **Actions** column to view the details about the message. The details include the basic attributes, download URL of the message body, message trace, and delivery status.

## Export dead-letter messages

If you cannot process dead-letter messages within the validity period, export the messages in the

Message Queue for Apache RocketMQ

console.

The

Message Queue for Apache RocketMQ

console allows you to export a single dead-letter message or export dead-letter messages in batches. The exported file is in the CSV format.

The following table describes the fields of an exported message.

| Field | Definition |
| --- | --- |
| topic | The topic to which the message belongs. |
| msgId | The ID of the message. |
| bornHost | The URL of the producer that produced the message. |
| bornTimestamp | The time when the message was produced. |
| storeTimestamp | The time when the message turned into a dead-letter message. |
| reconsumeTimes | The number of times that the message failed to be consumed. |
| properties | The message attributes in the JSON format. |
| body | The Base64-encoded message body. |
| bodyCRC | The cyclic redundancy check (CRC) of the message body. |

- Export a single dead-letter message

  In the

  Message Queue for Apache RocketMQ

  console, find the dead-letter message that you want to export and click **Export** in the **Actions** column.

- Export dead-letter messages in batches

  In the

Message Queue for Apache RocketMQ

console, enter the group ID to query the dead-letter messages, select the dead-letter messages that you want to export, and then click **Batch Export**.

### Resend dead-letter messages

If a message enters a dead-letter queue, the message cannot be consumed as expected for specific reasons. Therefore, you must process the message in a special way. After you troubleshoot the problems, you can resend the message to the corresponding consumer in the

Message Queue for Apache RocketMQ

console.

> **Notice** After a dead-letter message is resent to the consumer, the message will still be stored in the dead-letter queue for three days. The system automatically deletes the message after the three days.

- Resend a single dead-letter message

  In the

  Message Queue for Apache RocketMQ

  console, query one dead-letter message by message ID or query dead-letter messages by group ID. Find the dead-letter message that you want to resend and click **Resend** in the **Actions** column.

- Resend dead-letter messages in batches

  In the

  Message Queue for Apache RocketMQ

  console, query dead-letter messages by group ID, select the dead-letter messages that you want to resend, and then click **Batch Resend**.

# 1.1.5.7. Resource statistics

# 1.1.5.7.1. Overview

This topic describes how to use the resource statistics feature to query the statistics of produced messages and consumed messages.

The resource statistics feature provides the statistics of produced messages and consumed messages. This feature allows you to query the following data:

- **Statistics of produced messages**
  - Queries by topic: You can query the total number of messages that are received by a topic or the average number of messages that are received by a topic per second in a specified period of time.
  - Queries by instance: You can query the total number of messages that are received by the topics in a specified instance or the average number of messages that are received by the topics in a specified instance per second in a specified period of time.
- **Statistics of consumed messages**

- Queries by group ID: You can query the total number of messages that are sent from a topic to consumers identified by a group ID or the average number of messages that are sent from a topic to consumers identified by a group ID per second in a specified period of time.
- Queries by instance: You can query the total number of messages that are sent to all groups in a specified instance or the average number of messages that are sent to all groups in a specified instance per second in a specified period of time.

# 1.1.5.7.2. Query the statistics of produced messages

This topic describes how to query the statistics of produced messages. You can query the total number of messages that are received by a topic or all topics across the brokers in a specified instance in a specified period of time. You can also query the average number of messages that are received by a topic or all topics across the brokers in a specified instance per second in a specified period of time.

## Procedure

1. Log on to the Message Queue for Apache RocketMQ console.
2. In the left-side navigation pane, click **Resource Statistics**.
3. On the **Resource Statistics** page, click the **Message Production** tab.
4. From the **Resource Type** drop-down list, select a resource type for which you want to query the statistics of produced messages. Configure the related fields. Then, click **Search**.

   The following information describes the related fields:

   - **Resource Type**: The value can be Instance or Topic. Select Instance to query the total number of messages that are received by the topics in a specified instance or the average number of messages that are received by the topics in a specified instance per second in a specified period of time. Select Topic to query the total number of messages that are received by a specified topic or the average number of messages that are received by a specified topic per second in a specified period of time.
   - **Current Instance**: This parameter is displayed if **Resource Type** is set to Instance. This parameter is automatically set to the name and ID of the current instance.
   - **Topic**: This parameter is displayed if **Resource Type** is set to Topic. Select a topic to query the statistics of the produced messages that are sent to a specified topic in the current instance.
   - **Collection Type**: The value can be Total or TPS. Select Total to query the total number of messages that are received by the topic in each collection cycle. Select TPS to query the average number of messages that are received by the topic per second in each collection cycle.
   - **Collection Interval**: The value can be 1 Minute, 10 Minutes, 30 Minutes, or 1 Hour. This parameter specifies the interval at which data is collected. A smaller value indicates a higher data collection frequency and more detailed data.
   - **Time Range**: Message Queue for Apache RocketMQ allows you to query messages that are produced in the last three days.

   Query results are displayed in charts.

# 1.1.5.7.3. Query the statistics of consumed messages

This topic describes how to query the statistics of consumed messages. You can query the total number of messages that are sent from a topic to consumers identified by a group ID in a specified period of time. You can also query the average number of messages that are sent from a topic to consumers identified by a group ID per second in a specified period of time.

## Procedure

1. Log on to the Message Queue for Apache RocketMQ console.
2. In the left-side navigation pane, click **Resource Statistics**.
3. On the **Resource Statistics** page, click the **Message Consumption** tab.
4. From the **Resource Type** drop-down list, select a resource type for which you want to query the statistics of consumed messages. Configure the related fields. Then, click **Search**.

   The following information describes the related fields:

   - **Resource Type**: The value can be Instance or Group ID. Select Instance to query the total number of messages that are sent to the groups in a specified instance or the average number of messages that are sent to the groups in a specified instance per second in a specified period of time. Select Group ID to query the total number of messages that are sent from a topic to consumers identified by a group ID or the average number of messages that are sent from a topic to consumers identified by a group ID per second in a specified period of time.
   - **Current Instance**: This parameter is displayed if **Resource Type** is set to Instance. This parameter is automatically set to the name and ID of the current instance.
   - **Group ID**: This parameter is displayed if **Resource Type** is set to Group ID. You must select the group ID for which you want to query data.
   - **Topic**: This parameter is displayed if **Resource Type** is set to Group ID. You must select a topic from which the messages that you want to query are sent.
   - **Collection Type**: The value can be Total or TPS. Select Total to query the total number of messages that are sent to consumers identified by the group ID in each collection cycle. Select TPS to query the average number of messages that are sent to consumers identified by the group ID per second in each collection cycle.
   - **Collection Interval**: The value can be 1 Minute, 10 Minutes, 30 Minutes, or 1 Hour. This parameter specifies the interval at which data is collected. A smaller value indicates a higher data collection frequency and more detailed data.
   - **Time Range**: Message Queue for Apache RocketMQ allows you to query messages that are consumed in the last three days.

   Query results are displayed in charts.

# 1.1.5.8. Account authorization management

Message Queue for Apache RocketMQ allows you to use an Apsara Stack tenant account to grant permissions to publish and subscribe to a topic to another Apsara Stack tenant account or a Resource Access Management (RAM) user. An Apsara Stack tenant account is a level-1 department account. A RAM user is a personal account that is used to access the Apsara Stack resources.

## Grant permissions to another Apsara Stack tenant account

You can use an Apsara Stack tenant account to grant permissions to another Apsara Stack tenant account. To grant permissions to publish and subscribe to a topic, perform the following steps:

1. Log on to the Message Queue for Apache RocketMQ console. In the left-side navigation pane, click **Topics**.
2. On the **Topics** page, find the topic that you want to authorize another account to manage, click ⋯ in the **Actions** column, and then select **Authorize** from the drop-down list.
3. In the **Authorize** dialog box, set **Account Type** to **Apsara Stack Account**.
4. In the **Apsara Stack Account ID** field, enter the ID of the Apsara Stack tenant account to which you want to grant permissions.
5. From the **Authorization Type** drop-down list, select the permissions that you want to grant to the Apsara Stack tenant account. Then, click **OK**.

## Grant permissions to a RAM user

You can use an Apsara Stack tenant account to grant permissions to a RAM user that belongs to the Apsara Stack tenant account. To grant permissions to publish and subscribe to a topic, perform the following steps:

1. Log on to the Message Queue for Apache RocketMQ console. In the left-side navigation pane, click **Topics**.
2. On the **Topics** page, find the topic that you want to authorize a RAM user to manage, click ⋯ in the **Actions** column, and then select **Authorize** from the drop-down list.
3. In the **Authorize** dialog box, set **Account Type** to **RAM User**.
4. In the **RAM User Name** field, enter the name of the RAM user to which you want to grant permissions.
5. From the **Authorization Type** drop-down list, select the permissions that you want to grant to the RAM user. Then, click **OK**.

> **Note** The RAM user to which you want to grant permissions must be an account that is used to access the Apsara Stack resources and is owned by the department to which the Apsara Stack tenant account belongs.

## View authorization information

You can view the authorization records and the details of each topic in the Message Queue for Apache RocketMQ console. To view authorization information, perform the following steps:

1. Log on to the Message Queue for Apache RocketMQ console. In the left-side navigation pane, click **Topics**.
2. On the **Topics** page, find the topic that you want to view, click ⋯ in the **Actions** column, and then select **View Authorization** from the drop-down list to view the authorization records and information of the topic.

> **Note**
>
> - You do not need to authorize your account to manage the topics that you create.
> - After you log on to the Message Queue for Apache RocketMQ console by using an authorized account, you can view the topic that the account is authorized to manage. Then, you must create a group ID. You cannot use the group ID of the Apsara Stack tenant account that is used to grant permissions to your account.
> - A topic that an account is authorized to manage belongs to the Apsara Stack tenant account that is used to grant permissions. Therefore, you cannot use the authorized account to delete the topic.
> - If you grant permissions to a RAM user, you cannot use the authorized RAM user to create topics.
> - If you grant permissions to another Apsara Stack tenant account, you can use the authorized Apsara Stack tenant account to create topics. However, the created topics are not associated to the Apsara Stack tenant account that is used to grant permissions.

# 1.1.5.9. Switch between different access modes

Message Queue for Apache RocketMQ supports instance-specific management. By default, one instance can be deployed at a time. Message Queue for Apache RocketMQ supports advanced access control by using virtual private clouds (VPC) for each instance.

## Context

By default, a Message Queue for Apache RocketMQ instance supports the Any Tunnel access mode. This means that the Message Queue for Apache RocketMQ instance can be accessed in each VPC environment. You can switch the access mode in the console at any time. If the access mode of a Message Queue for Apache RocketMQ instance is switched to Single Tunnel, the instance can be accessed only in a specified VPC environment.

## Procedure

1. Log on to the Message Queue for Apache RocketMQ console. In the left-side navigation pane, click **Cluster Management**.
2. Find the instance whose access mode you want to switch and click **Switch Access Method** in the **Actions** column.
3. Select an access mode. You can select one of the following options:
   - **Single Tunnel**: If this option is selected, the instance can be accessed only in a specified VPC environment. The page displays the **vSwitch ID** field. You must specify the vSwitch ID of the VPC that you use.
   - **Any Tunnel**: If this option is selected, the instance can be accessed in each VPC environment.
4. Select an option for **Forced Switch** to indicate whether to forcibly switch the access mode. The switching between access modes may cause the transient interruption of services. When forcible switching is disabled, one access mode can be switched to another access mode only when the instance traffic is light. This means that the transactions per second (TPS) must be no more than 10. When forcible switching is enabled, one access mode can be switched to another access mode regardless of the service traffic.
5. Click **OK**.

# 1.1.6. SDK user guide

## 1.1.6.1. Demo projects

### 1.1.6.1.1. Overview

This topic helps engineers who are new to Message Queue for Apache RocketMQ to build a Message Queue for Apache RocketMQ test project. The demo project is a Java project. It contains test code for normal messages, transactional messages, and scheduled messages. The demo project also contains Spring configurations.

### 1.1.6.1.2. Prepare the environment

This topic describes how to prepare an environment for a Message Queue for Apache RocketMQ demo project.

#### Procedure

- Install an integrated development environment (IDE).

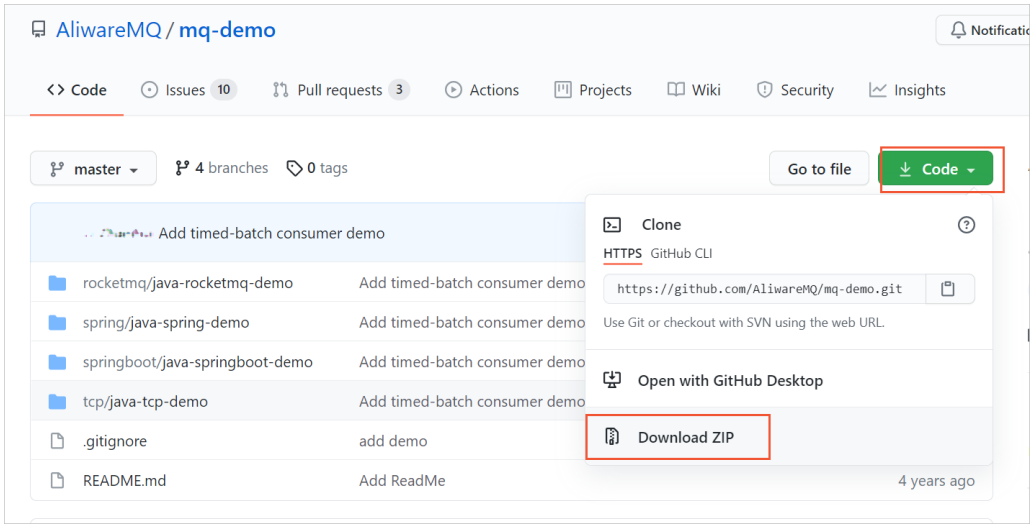
  You can use IntelliJ IDEA or Eclipse as the IDE. IntelliJ IDEA is used in this example.

  Download IntelliJ IDEA Ultimate Edition from IntelliJ IDEA. Then, follow the installation instructions to install IntelliJ IDEA Ultimate Edition.

- Download a demo project.

  Download a demo project from GitHub to your on-premises machine.

  Download a demo project

  After the downloaded package is decompressed, a folder named *mq-demo-master* appears on your on-premises machine.

>

### 1.1.6.1.3. Configure a demo project

This topic describes how to configure a demo project.

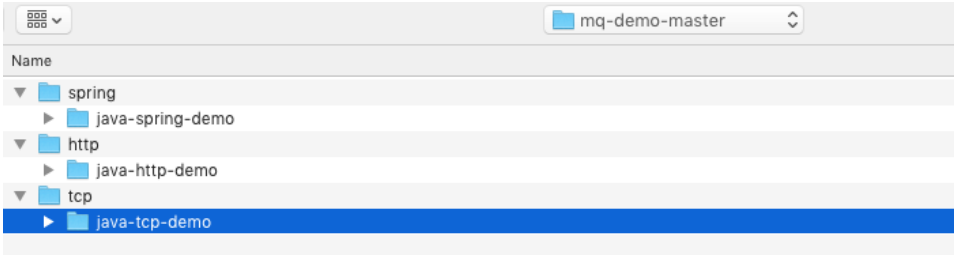
## Prerequisites

- You have prepared the environment for the demo project.
- You have installed the JDK on your on-premises machine. For more information, visit Java SE Downloads. We recommend that you use JDK 8.

## Procedure

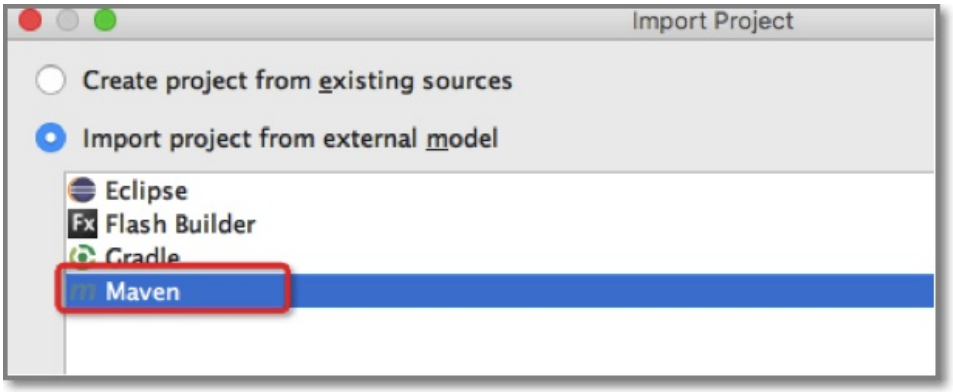
1. Import the demo project to IntelliJ IDEA.
   i. On the IntelliJ IDEA page, click **Import Project** and select the *mq-demo-master* folder.

      Select the mq-demo-master folder

   ii. Select Import project from external model.

      Select Import project from external model

   iii. Click **Next** until the project is imported. The JAR dependency needs to be loaded to the demo project. Therefore, it takes two to three minutes to import the project.
2. Creates resources.

   Create the required resources, such as topics and group IDs in the Message Queue for Apache RocketMQ console and obtain the AccessKey pair in the Apsara Uni-manager Management Console for identity authentication.

   i. For more information about how to create topics and group IDs, see Create resources.

   ii. Perform the following operations to obtain the AccessKey ID and AccessKey secret:

      In the Apsara Uni-manager Management Console, move your pointer over the profile picture and select **User Information**. On the page that appears, view the AccessKey ID and AccessKey secret in the **Apsara Stack AccessKey Pair** section.
3. Configure the demo.

   Configure the MqConfig class and the *common.xml* file.

i. The following sample code provides an example on how to configure the MqConfig class:

```
public static final String TOPIC = "The topic that you created in the Message Queue
for Apache RocketMQ console."
public static final String GROUP_ID = "The group ID that you created in the Message
Queue for Apache RocketMQ console."
public static final String ACCESS_KEY = "The AccessKey ID that you created in the A
psara Uni-manager Management Console for identity authentication."
public static final String SECRET_KEY = "The AccessKey secret that you created in t
he Apsara Uni-manager Management Console for identity authentication."
public static final String NAMESRV_ADDR = "The TCP endpoint of your Message Queue f
or Apache RocketMQ instance. You can obtain the endpoint in the Message Queue for A
pache RocketMQ console."
```

> **Note** You must use the AccessKey ID and AccessKey secret of the account that you use to create the topic.

ii. Configure the *common.xml* file.

```
 <props>
 <prop key="AccessKey">XXX</prop> <!-- Modify the values based on your resources --
>
 <prop key="SecretKey">XXX</prop>
 <prop key="GROUP_ID">XXX</prop>
 <prop key="Topic">XXX</prop>
 <prop key="NAMESRV_ADDR">XXX</prop>
</props>
```

# 1.1.6.1.4. Run the demo project

After you configure the demo project, you can start the corresponding classes to send and receive messages of different types.

## Call the main method to send and receive messages

1. Run the SimpleMQProducer class to send messages.
2. Log on to the Message Queue for Apache RocketMQ console. In the left-side navigation pane, click **Message Query**. On the Message Query page, click the **By Topic** tab. On the By Topic tab, select the topic of the message that you sent. The query result shows that the message is sent to the topic.
3. Run the SimpleMQConsumer class to receive messages. A log is printed. The log indicates that the message is received. The class needs to be initialized. This takes several seconds. Initialization seldom occurs in the production environment.

Log on to the Message Queue for Apache RocketMQ console. In the left-side navigation pane, click Instance Details. On the Instance Details page, select your instance. In the left-side navigation pane, click **Groups**. On the Groups page, find the group ID that you want to view and click **Consumer Status** in the Actions column. In the **Consumer Status** panel, the information shows that the started consumers are online and the subscriptions of the consumers are consistent.

## Use Spring to send and receive messages

1. Run the ProducerClient class to send messages.
2. Run the ConsumerClient class to receive messages.

Log on to the Message Queue for Apache RocketMQ console. In the left-side navigation pane, click Instance Details. On the Instance Details page, select your instance. In the left-side navigation pane, click **Groups**. On the Groups page, find the group ID that you want to view and click **Consumer Status** in the Actions column. In the **Consumer Status** panel, the information shows that the started consumers are online and the subscriptions of the consumers are consistent.

### Send transactional messages

Run the SimpleTransactionProducer class to send messages.

> **Note**
>
> The LocalTransactionCheckerImpl class is used to check the status of local transactions. This class is used to check whether a local transaction is committed. For more information, see Send and subscribe to transactional messages.

### Send and receive ordered messages

Run the SimpleOrderConsumer class to receive messages.

Run the SimpleOrderProducer class to send messages.

> **Note** Ordered messages are sent and consumed in first-in-first-out (FIFO) order. For more information, see Send and receive ordered messages.

### Send scheduled or delayed messages

Run the MQTimerProducer class to send messages. These messages are delivered after a delay of 3 seconds.

> **Note** You can specify an exact delay, which is up to 40 days. For more information, see Send and receive scheduled messages.

# 1.1.6.2. SDK user guide

# 1.1.6.2.1. SDK for Java

# 1.1.6.2.1.1. Usage notes

Message Queue for Apache RocketMQ provides SDK for Java for you to send and subscribe to messages. This topic describes the parameters of Java methods and how to call these methods.

Common parameters

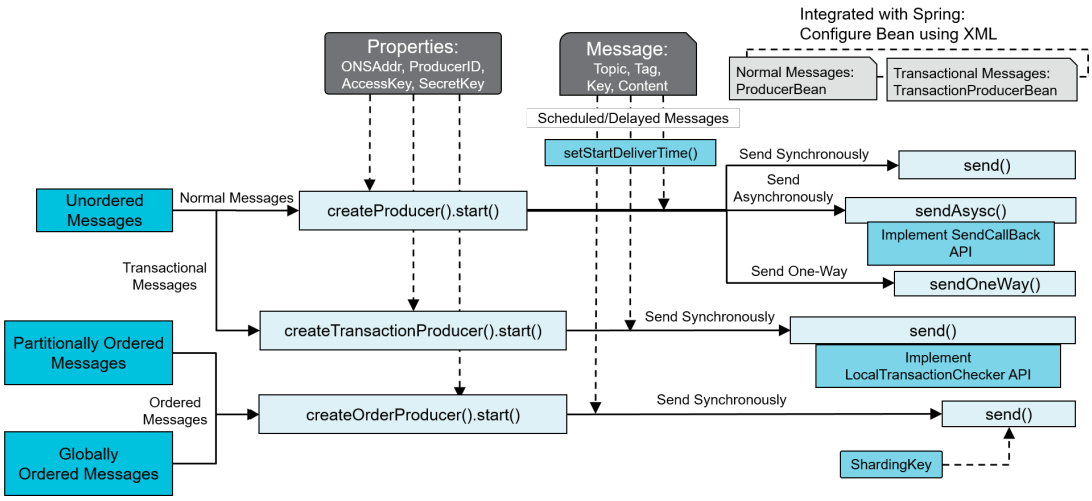
| Parameter | Description |
| --- | --- |

| Parameter | Description |
| --- | --- |
| NAMESRV_ADDR | The TCP endpoint. You can obtain the endpoint on the **Instance Details** page in the Message Queue for Apache RocketMQ console. |
| AccessKey | The AccessKey ID that you created in the Apsara Uni-manager Management Console for identity authentication. |
| SecretKey | The AccessKey secret that you created in the Apsara Uni-manager Management Console for identity authentication. |
| OnsChannel | The source of the user. Default value: ALIYUN. |

Parameters for sending messages

| Parameter | Description |
| --- | --- |
| SendMsgTimeoutMillis | The timeout period for sending messages. Unit: milliseconds. Default value: 3000. |
| CheckImmunityTimeInSeconds (for transactional messages) | The shortest time interval before the first back-check for the status of local transaction. Unit: seconds. |
| shardingKey (for ordered messages) | The partition key that is used to determine the partitions to which ordered messages are distributed. |

Methods and parameters for using SDK for Java to send messages

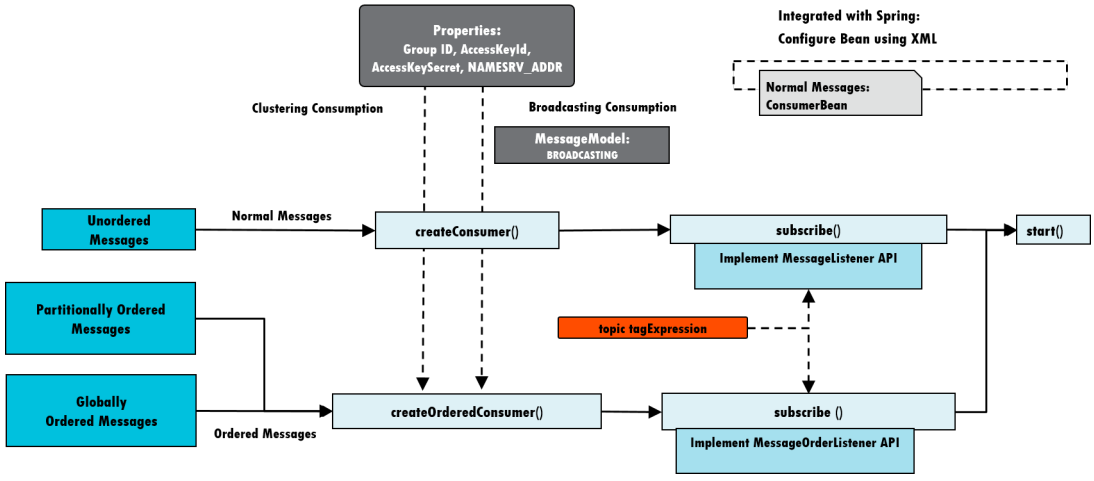



Parameters for subscribing to messages

| Parameter | Description |
| --- | --- |
| GROUP_ID | The group ID that you created in the Message Queue for Apache RocketMQ console. |
| MessageModel | The mode in which a consumer instance consumes messages. Valid values: CLUSTERING and BROADCASTING. Default value: CLUSTERING. |

| Parameter | Description |
| --- | --- |
| ConsumeThreadNums | The number of consumer threads for a consumer instance. Default value: 64. |
| MaxReconsumeTimes | The maximum number of delivery retries for a message that fails to be consumed. Default value: 16. |
| ConsumeTimeout | The maximum timeout period for consuming a message. If a message fails to be consumed within this period, the consumption fails and the message can be redelivered. A proper value must be set for each type of business. Unit: minutes. Default value: 15. |
| suspendTimeMillis (for ordered messages) | The interval between delivery retries for an ordered message that fails to be consumed. |
| maxCachedMessageAmount | The maximum number of messages cached on the on-premises client. Default value: 1000. |
| maxCachedMessageSizeInMiB | The maximum size of messages cached on the on-premises client. Valid values: 16 MB to 2 GB. Default value: 512 MB. |

Methods and parameters for using SDK for Java to subscribe to messages




## Sample code for sending and subscribing to messages

- Send and subscribe to normal messages
- Send and receive ordered messages
- Send and receive scheduled messages
- Send and receive delayed messages
- Send and subscribe to transactional messages

# 1.1.6.2.1.2. Prepare the environment

Before you run the Java code provided in this topic, prepare the environment based on the following instructions:

## Procedure

1. Introduce the dependency by using one of the following methods:
   - Introduce the dependency by using Maven:

     ```
     <dependency>
         <groupId>com.aliyun.openservices</groupId>
        <artifactId>ons-client</artifactId>
        <version>1.8.4.Final</version>
     </dependency>
     ```

   - Download the JAR dependency.
2. Go to the console to create the topics and group IDs involved in the code.

   You can customize message tags in your application. For more information about how to create a message tag, see Create resources.
3. For applications that use the TCP client SDK to access Message Queue for Apache RocketMQ, make sure that the applications are deployed on Elastic Compute Service (ECS) instances in the same region.

# 1.1.6.2.1.3. Configure logging

Client logs record exceptions that occur when the Message Queue for Apache RocketMQ clients are running. Client logs help you locate and handle these exceptions in a quick manner. This topic describes how to print the logs of a Message Queue for Apache RocketMQ client and provides the default and custom configurations.

## Print client logs

TCP client SDK for Java of Message Queue for Apache RocketMQ is programmed by using the Simple Logging Facade for Java (SLF4J).

- **Message Queue for Apache RocketMQ SDK for Java 1.7.8.Final or later**

  Message Queue for Apache RocketMQ SDK for Java 1.7.8.Final has a built-in framework for logging. You do not need to add a dependency on the corresponding logging framework for an application on the client before you print the logs of a Message Queue for Apache RocketMQ client.

  For information about the default logging configuration for a Message Queue for Apache RocketMQ client and how to modify this configuration, see Configure client logs.

- **Message Queue for Apache RocketMQ SDK for Java versions earlier than 1.7.8.Final**

  Message Queue for Apache RocketMQ SDK for Java versions earlier than 1.7.8.Final support only Log4j and Logback. These versions do not support Log4j2. For these versions, you must add a dependency on the corresponding logging framework to the *pom.xml* file or the .lib file before you print the logs of a Message Queue for Apache RocketMQ client.

  The following sample code provides examples on how to add dependencies on Log4j and Logback:

- **Method 1: Use Log4j as the logging framework**

```
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>jcl-over-slf4j</artifactId>
    <version>1.7.7</version>
</dependency>
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-log4j12</artifactId>
    <version>1.7.7</version>
</dependency>
<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.17</version>
</dependency>
```

- **Method 2: Use Logback as the logging framework**

```
<dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-core</artifactId>
    <version>1.1.2</version>
</dependency>
<dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-classic</artifactId>
<version>1.1.2</version>
</dependency>
```

> **Note**
>
> If an application uses both Log4j and Logback as logging frameworks, client logs cannot be properly printed due to logging conflicts. To properly print the logs of a Message Queue for Apache RocketMQ client, make sure that you add only one dependency on one logging framework for the application. We recommend that you run the **mvn clean dependency:tree | grep log** command to check whether your application uses only one of the logging frameworks.

## Configure logging for a Message Queue for Apache RocketMQ client

You can customize the following settings for a Message Queue for Apache RocketMQ client: **the path for storing log files** , **log level**, and **maximum number of historical log files retained**. To facilitate log transmission and viewing, the **maximum size of a single log file** retains the default value of 64 MB. This value cannot be changed.

The following table describes these parameters that you can configure.

| Parameter | Description |
| --- | --- |

| Parameter | Description |
| --- | --- |
| The path to store log files | Make sure that the application has the write permissions for this path. Otherwise, logs cannot be printed. |
| The maximum number of historical log files that are retained | You can set this parameter to a value between 1 and 100. If you enter a value that is not within this range or a value that is in an invalid format, the system retains 10 historical log files by default. |
| The log level | You can set this parameter to one of the following values: ERROR, WARN, INFO, and DEBUG. If this parameter is set to an invalid value, the system uses the default value INFO. |

- **Default configuration**

  After you start a Message Queue for Apache RocketMQ client, the client generates log files based on the following default configuration:

  - The path to store log files: `/{user.home}/logs/ons.log`, where `{user.home}` is the root directory of the account that runs the current Java process.
  - The maximum number of historical log files that are retained: 10
  - Log level: INFO
  - The maximum size of a single log file: 64 MB

- **Custom configuration**

  > **Note**
  >
  > To customize the logging configuration of a Message Queue for Apache RocketMQ client, update the SDK for Java to V1.2.5 or later.

  To customize the logging configuration of a Message Queue for Apache RocketMQ client in the SDK for Java, configure the following system parameters:

  - ons.client.logRoot: the path to store log files
  - ons.client.logFileMaxIndex: the maximum number of historical log files that are retained
  - ons.client.logLevel: the log level

  **Examples**

  Add the following system parameters to the startup script or integrated development environment (IDE) virtual machine (VM) options:

  - **Linux**

    ```
    -Dons.client.logRoot=/home/admin/logs -Dons.client.logLevel=WARN -Dons.client.logFileMaxIndex=20
    ```

- **Windows**

```
-Dons.client.logRoot=D:\logs -Dons.client.logLevel=WARN -Dons.client.logFileMaxIndex=20
```

*/home/admin/* and *D:* are only examples. Replace them with your system directories.

# 1.1.6.2.1.4. Spring integration

Overview

This topic describes how to send and subscribe to messages by using Message Queue for Apache RocketMQ in the Spring framework. This topic includes three parts: the integration of a normal message producer and Spring, the integration of a transactional message producer and Spring, and the integration of a message consumer and Spring.

The subscriptions of all consumer instances identified by the same group ID must be consistent. For more information, see Subscription consistency.

The configuration parameters supported in the Spring framework are the same as those used in TCP client SDK for Java. For more information, see How to use the Java SDK.

Integrate a producer with Spring

This topic describes how to integrate a producer with Spring.

## Procedure

1. Define information such as the producer bean in *producer.xml*.

```
<?xml version="1.0" encoding="UTF-8"?>
 <beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.spri
ngframework.org/schema/beans/spring-beans.xsd">
     <bean id="producer" class="com.aliyun.openservices.ons.api.bean.ProducerBean" init
-method="start" destroy-method="shutdown">
     <!-- The Spring framework supports all the configuration items that SDK for Java s
upports. -->
         <property name="properties" > <! -- Configurations of the producer -->
             <props>
                 <prop key="AccessKey">XXX</prop>
                 <prop key="SecretKey">XXX</prop>
                 <!-- The ons-client version is 1.8.4.Final, which must be configured.
You can obtain the TCP endpoint on the Instance Details page in the Message Queue for A
pache RocketMQ console.
                 <prop key="NAMESRV_ADDR">XXX</prop>
                 -->
             </props>
         </property>
     </bean>
 </beans>
```

2. Produce messages by using the producer that is integrated with Spring.

```
package demo;
import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.Producer;
import com.aliyun.openservices.ons.api.SendResult;
import com.aliyun.openservices.ons.api.exception.ONSClientException;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class ProduceWithSpring {
    public static void main(String[] args) {
        /**
         * The producer bean is configured in producer.xml. You can call the ApplicationContext class to obtain the bean or inject the bean to other classes, such as a specific controller.
         */
        ApplicationContext context = new ClassPathXmlApplicationContext("producer.xml");
        Producer producer = (Producer) context.getBean("producer");
        // Cyclically send messages.
        for (int i = 0; i < 100; i++) {
            Message msg = new Message( //
                    // The topic of the message.
                    "TopicTestMQ",
                    // The message tag, which is similar to a Gmail tag. The message tag is used to sort messages and helps the consumer filter messages on the Message Queue for Apache RocketMQ broker based on specified conditions.
                    "TagA",
                    // The message body in the binary format. Message Queue for Apache RocketMQ does not process the message body.
                    // The producer and consumer must agree on the serialization and deserialization methods.
                    "Hello MQ".getBytes());
            // The key of the message. The key is the business-specific attribute of the message and must be globally unique whenever possible.
            // A unique key helps you query and resend a message in the Message Queue for Apache RocketMQ console if the message fails to be received.
            // Note: Messages can be sent and received even if you do not set this parameter.
            msg.setKey("ORDERID_100");
            // Send the message. If no error occurs, the message is sent.
            try {
                SendResult sendResult = producer.send(msg);
                assert sendResult != null;
                System.out.println("send success: " + sendResult.getMessageId());
            }catch (ONSClientException e) {
                System.out.println("failed to send the message");
            }
        }
    }
}
```

## Integrate a transactional message producer with Spring

This topic describes how to integrate a producer that produces transactional messages with Spring.

## Context

For more information about transactional messages, see Send and subscribe to transactional messages.

## Procedure

1. Implement the LocalTransactionChecker class. A producer can have only one LocalTransactionChecker class.

```
package demo;
import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.transaction.LocalTransactionChecker;
import com.aliyun.openservices.ons.api.transaction.TransactionStatus;
public class DemoLocalTransactionChecker implements LocalTransactionChecker {
    public TransactionStatus check(Message msg) {
        System.out.println("Start to back-check the status of local transaction.");
        return TransactionStatus.CommitTransaction; // Returns different values for Tra
nsactionStatus based on the status check result of the local transaction.
    }
}
```

2. Define information such as the producer bean in *transactionProducer.xml*.

```
 <?xml version="1.0" encoding="UTF-8"?>
 <beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.spri
ngframework.org/schema/beans/spring-beans.xsd">
     <bean id="localTransactionChecker" class="demo.DemoLocalTransactionChecker"></bean
>
     <bean id="transactionProducer" class="com.aliyun.openservices.ons.api.bean.Transac
tionProducerBean" init-method="start" destroy-method="shutdown">
         <property name="properties" > <! -- Configurations of the transactional messag
e producer -->
             <props>
                 <prop key="AccessKey">AKDEMO</prop>
                 <prop key="SecretKey">SKDEMO</prop>
                 <prop key="GROUP_ID">GID_DEMO</prop>
                 <!-- The ons-client version is 1.8.4.Final, which must be configured.
You can obtain the TCP endpoint on the Instance Details page in the Message Queue for A
pache RocketMQ console.
                 <prop key="NAMESRV_ADDR">XXX</prop>
                 -->
             </props>
         </property>
         <property name="localTransactionChecker" ref="localTransactionChecker"></prope
rty>
     </bean>
 </beans>
```

3. Produce transactional messages by using the producer that is integrated with Spring.

```
package demo;
import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.SendResult;
import com.aliyun.openservices.ons.api.transaction.LocalTransactionExecuter;
import com.aliyun.openservices.ons.api.transaction.TransactionProducer;
import com.aliyun.openservices.ons.api.transaction.TransactionStatus;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class ProduceTransMsgWithSpring {
    public static void main(String[] args) {
        /**
         * The bean of the transactional message producer is configured in transaction
Producer.xml. You can call the ApplicationContext class to obtain the bean or inject th
e bean to other classes, such as a specific controller.
         * Send transactional messages.
         */
        ApplicationContext context = new ClassPathXmlApplicationContext("transactionPr
oducer.xml");
        TransactionProducer transactionProducer = (TransactionProducer) context.getBea
n("transactionProducer");
        Message msg = new Message("XXX", "TagA", "Hello MQ transaction===".getBytes())
;
        SendResult sendResult = transactionProducer.send(msg, new LocalTransactionExec
uter() {
            @Override
            public TransactionStatus execute(Message msg, Object arg) {
                System.out.println("A local transaction is executed.");
                return TransactionStatus.CommitTransaction; // Returns different value
s for TransactionStatus based on the execution result of the local transaction.
            }
        }, null);
    }
}
```

Integrate a consumer with Spring

This topic describes how to integrate a consumer with Spring.

## Procedure

1. Create a message listener. The following sample code provides an example:

```
package demo;
import com.aliyun.openservices.ons.api.Action;
import com.aliyun.openservices.ons.api.ConsumeContext;
import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.MessageListener;
public class DemoMessageListener implements MessageListener {
    public Action consume(Message message, ConsumeContext context) {
        System.out.println("Receive: " + message.getMsgID());
        try {
            //do something..
            return Action.CommitMessage;
        }catch (Exception e) {
            // The message failed to be consumed.
            return Action.ReconsumeLater;
        }
    }
}
```

2. Define information such as the consumer bean in *consumer.xml*.

```
 <?xml version="1.0" encoding="UTF-8"?>
 <beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.spri
ngframework.org/schema/beans/spring-beans.xsd">
     <bean id="msgListener" class="demo.DemoMessageListener"></bean> <!--Configurations
of the message listener-->
 <!-- When multiple consumers identified by the same group ID subscribe to the same top
ic, you can create multiple consumer beans. -->
     <bean id="consumer" class="com.aliyun.openservices.ons.api.bean.ConsumerBean" init
-method="start" destroy-method="shutdown">
         <property name="properties" > <!-- Configurations of the consumer -->
             <props>
                 <prop key="GROUP_ID">GID_DEMO</prop> <!-- Replace the value with the g
roup ID that you created in the console. -->
                 <prop key="AccessKey">AKDEMO</prop>
                 <prop key="SecretKey">SKDEMO</prop>
                 <!-- The ons-client version is 1.8.4.Final, which must be configured.
You can obtain the TCP endpoint on the Instance Details page in the Message Queue for A
pache RocketMQ console.
                 <prop key="NAMESRV_ADDR">XXX</prop>
                 -->
                 <!-- Set the number of consumer threads to 50.
                 <prop key="ConsumeThreadNums">50</prop>
                 -->
             </props>
         </property>
         <property name="subscriptionTable">
             <map>
                 <entry value-ref="msgListener">
                     <key>
                         <bean class="com.aliyun.openservices.ons.api.bean.Subscription
">
                             <property name="topic" value="TopicTestMQ"/>
```

```
                            <property name="expression" value="*"/><!--The expression
is the tag. You can set the value to a specific tag or *. For example, a specific tag c
an be taga||tagb||tagc.  * indicates that all tags are subscribed to. Wildcards are not
supported. -->
                        </bean>
                    </key>
                </entry>
                <!-- Add entry nodes to subscribe to more tags. -->
                <entry value-ref="msgListener">
                    <key>
                        <bean class="com.aliyun.openservices.ons.api.bean.Subscription
">
                            <property name="topic" value="TopicTestMQ-Other"/> <!--Sub
scribe to another topic. -->
                            <property name="expression" value="taga||tagb"/> <!-- Subs
cribe to multiple tags. -->
                        </bean>
                    </key>
                </entry>
            </map>
        </property>
    </bean>
</beans>
```

3. Run the consumer that is integrated with Spring.

```
package demo;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class ConsumeWithSpring {
    public static void main(String[] args) {
        /**
         * The consumer bean is configured in consumer.xml. You can call the Applicatio
nContext class to obtain the bean or inject the bean to other classes, such as a specif
ic controller.
         */
        ApplicationContext context = new ClassPathXmlApplicationContext("consumer.xml")
;
        System.out.println("Consumer Started");
    }
}
```

# 1.1.6.2.1.5. Three modes for sending messages

Overview

In Message Queue for Apache RocketMQ, messages can be sent in reliable synchronous mode, reliable asynchronous mode, and one-way mode. This topic describes the principles, scenarios, and differences of these modes, and provides sample code for your reference.

> **Note** Ordered messages can be sent only in reliable synchronous mode.

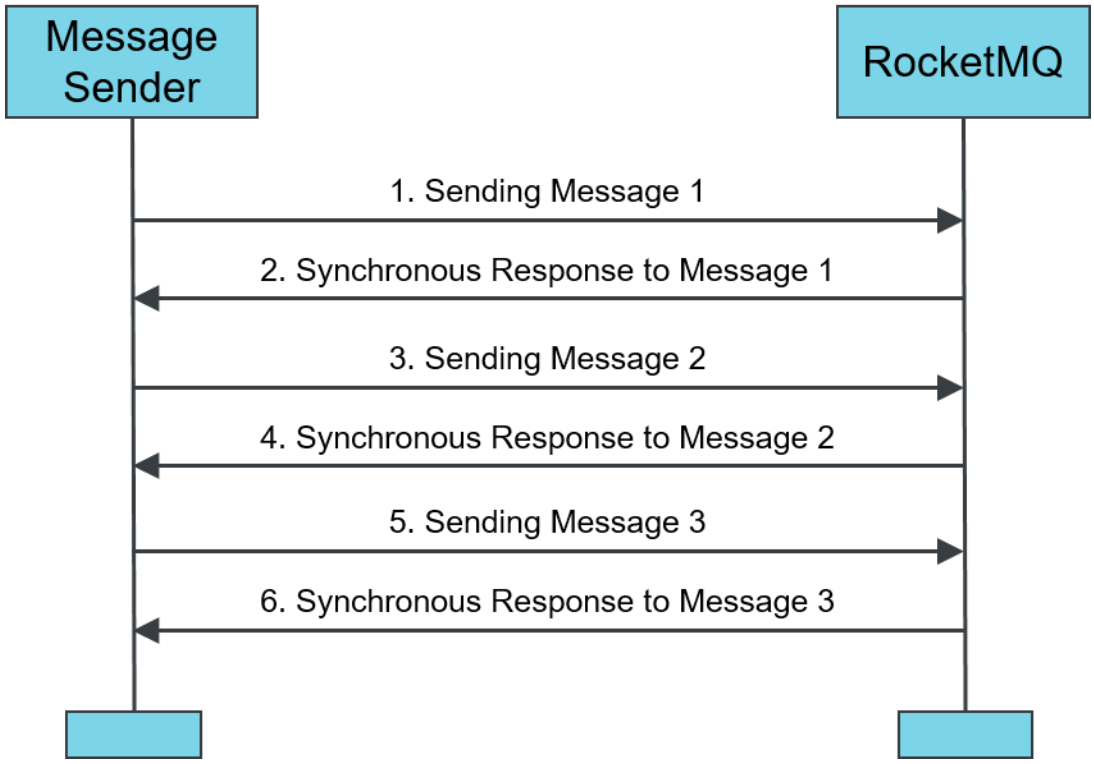
Reliable synchronous transmission

This topic describes the principle and scenarios of the reliable synchronous transmission mode.

## How it works

Synchronous transmission means that the message producer sends the next message only after it receives a response to the previous message from the broker.

Synchronous transmission




## Scenarios

This mode is applicable to various scenarios, such as important notification emails, short message service (SMS) notifications for registration results, and SMS marketing systems.

## Sample code

```
import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.Producer;
import com.aliyun.openservices.ons.api.SendResult;
import com.aliyun.openservices.ons.api.ONSFactory;
import com.aliyun.openservices.ons.api.PropertyKeyConst;
import java.util.Properties;
public class ProducerTest {
    public static void main(String[] args) {
        Properties properties = new Properties();
        // The AccessKey ID that you created in the Apsara Uni-manager Management Console f
or identity authentication.
        properties.put(PropertyKeyConst.AccessKey,"XXX");
        // The AccessKey secret that you created in the Apsara Uni-manager Management Conso
le for identity authentication.
        properties.put(PropertyKeyConst.SecretKey, "XXX");
        // The timeout interval for sending a message, in milliseconds.
```

```
        properties.setProperty(PropertyKeyConst.SendMsgTimeoutMillis, "3000");
        // The TCP endpoint. To obtain the endpoint, log on to the Message Queue for Apache
RocketMQ console. In the left-side navigation pane, click Instance Details. On the Instance
Details page, select your instance. On the Instance Information tab, view the endpoint in t
he Obtain Endpoint Information section.
        properties.put(PropertyKeyConst.NAMESRV_ADDR,
          "XXX");
        Producer producer = ONSFactory.createProducer(properties);
        // Before you use the producer to send a message, call the start() method once to s
tart the producer.
        producer.start();
        // Cyclically send messages.
        for (int i = 0; i < 10; i++){
            Message msg = new Message( //
                // The topic of the message.
                "TopicTestMQ",
                // The message tag, which is similar to a Gmail tag. The message tag is use
d to sort messages and helps the consumer filter messages on the Message Queue for Apache R
ocketMQ broker based on specified conditions.
                "TagA",
                // The message body in the binary format. Message Queue for Apache RocketMQ
does not process the message body.
                // The producer and consumer must agree on the serialization and deserializ
ation methods.
                "Hello MQ".getBytes());
            // The key of the message. The key is the business-specific attribute of the me
ssage and must be globally unique whenever possible.
            // A unique key helps you query and resend a message in the Message Queue for A
pache RocketMQ console if the message fails to be received.
            // Note: Messages can be sent and received even if you do not specify the messa
ge key.
            msg.setKey("ORDERID_" + i);
            try {
                SendResult sendResult = producer.send(msg);
                // Send the message in synchronous mode. If no error occurs, the message is
sent.
                if (sendResult != null) {
                    System.out.println(new Date() + " Send mq message success. Topic is:" +
msg.getTopic() + " msgId is: " + sendResult.getMessageId());
                }
            }
            catch (Exception e) {
                // Specify the logic to resend or persist the message if the message fails
to be sent.
                System.out.println(new Date() + " Send mq message failed. Topic is:" + msg.
getTopic());
                e.printStackTrace();
            }
        }
        // Before you exit the application, shut down the producer object.
        // Note: You can choose not to shut down the producer object.
        producer.shutdown();
    }
}
```

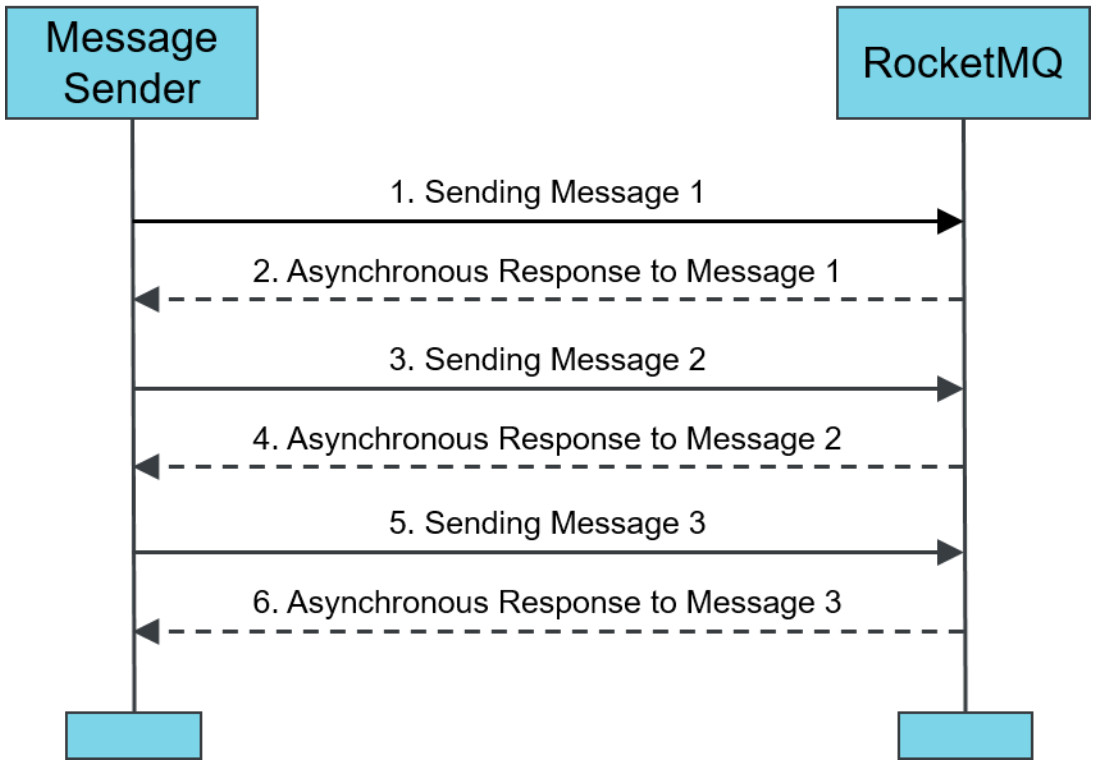
Reliable asynchronous transmission

This topic describes the principle and scenarios of the reliable asynchronous transmission mode.

## How it works

In reliable asynchronous transmission mode, a producer sends the next message without waiting for a response to the previous message from the Message Queue for Apache RocketMQ broker. This mode uses the SendCallback method to fire a callback after a message is sent. An application sends the next message before it receives a response to the previous message from the Message Queue for Apache RocketMQ broker. After the SendCallback method is called, the application receives the response to the previous message from the Message Queue for Apache RocketMQ broker and processes the response.

Asynchronous transmission




## Scenarios

This mode is used for time-consuming processes in business scenarios that are sensitive to the response time. For example, after you upload a video, a callback is fired to enable transcoding. After the video is transcoded, a callback is fired to push transcoding results.

## Sample code

```
import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.OnExceptionContext;
import com.aliyun.openservices.ons.api.Producer;
import com.aliyun.openservices.ons.api.SendCallback;
import com.aliyun.openservices.ons.api.SendResult;
import com.aliyun.openservices.ons.api.ONSFactory;
import com.aliyun.openservices.ons.api.PropertyKeyConst;
import java.util.Properties;
public static void main(String[] args) {
    Properties properties = new Properties();
```

```
        Properties properties = new Properties();
        // The AccessKey ID that you created in the Apsara Uni-manager Management Console for identity authentication.
        properties.put(PropertyKeyConst.AccessKey, "XXX");
        // The AccessKey secret that you created in the Apsara Uni-manager Management Console for identity authentication.
        properties.put(PropertyKeyConst.SecretKey, "XXX");
        // The timeout interval for sending a message, in milliseconds.
        properties.setProperty(PropertyKeyConst.SendMsgTimeoutMillis, "3000");
        // The TCP endpoint. To obtain the endpoint, log on to the Message Queue for Apache RocketMQ console. In the left-side navigation pane, click Instance Details. On the Instance Details page, select your instance. On the Instance Information tab, view the endpoint in the Obtain Endpoint Information section.
        properties.put(PropertyKeyConst.NAMESRV_ADDR,
          "XXX");
        Producer producer = ONSFactory.createProducer(properties);
        // Before you use the producer to send a message, call the start() method once to start the producer.
        producer.start();
        Message msg = new Message(
                // The topic of the message.
                "TopicTestMQ",
                // The message tag, which is similar to a Gmail tag. The message tag is used to sort messages and helps the consumer filter messages on the Message Queue for Apache RocketMQ broker based on specified conditions.
                "TagA",
                // The message body in the binary format. Message Queue for Apache RocketMQ does not process the message body. The producer and consumer must agree on the serialization and deserialization methods.
                "Hello MQ".getBytes());
        // The key of the message. The key is the business-specific attribute of the message and must be globally unique whenever possible.  // A unique key helps you query and resend a message in the Message Queue for Apache RocketMQ console if the message fails to be received.
        // Note: Messages can be sent and received even if you do not set this parameter.
        msg.setKey("ORDERID_100");
        // Send the message in asynchronous mode. The result is returned to the producer after the producer calls the callback function.
        producer.sendAsync(msg, new SendCallback() {
            @Override
            public void onSuccess(final SendResult sendResult) {
                // The message is sent to the consumer.
                System.out.println("send message success. topic=" + sendResult.getTopic() + ", msgId=" + sendResult.getMessageId());
            }
            @Override
            public void onException(OnExceptionContext context) {
                // Specify the logic to resend or persist the message if the message fails to be sent.
                System.out.println("send message failed. topic=" + context.getTopic() + ", msgId=" + context.getMessageId());
            }
        });
        // The message ID can be obtained before the callback function returns the result.
        System.out.println("send message async. topic=" + msg.getTopic() + ", msgId=" + msg
```

```
.getMsgID());
        // Before you exit the application, shut down the producer object.  Note: You can c
hoose not to shut down the producer object.
        producer.shutdown();
    }
```

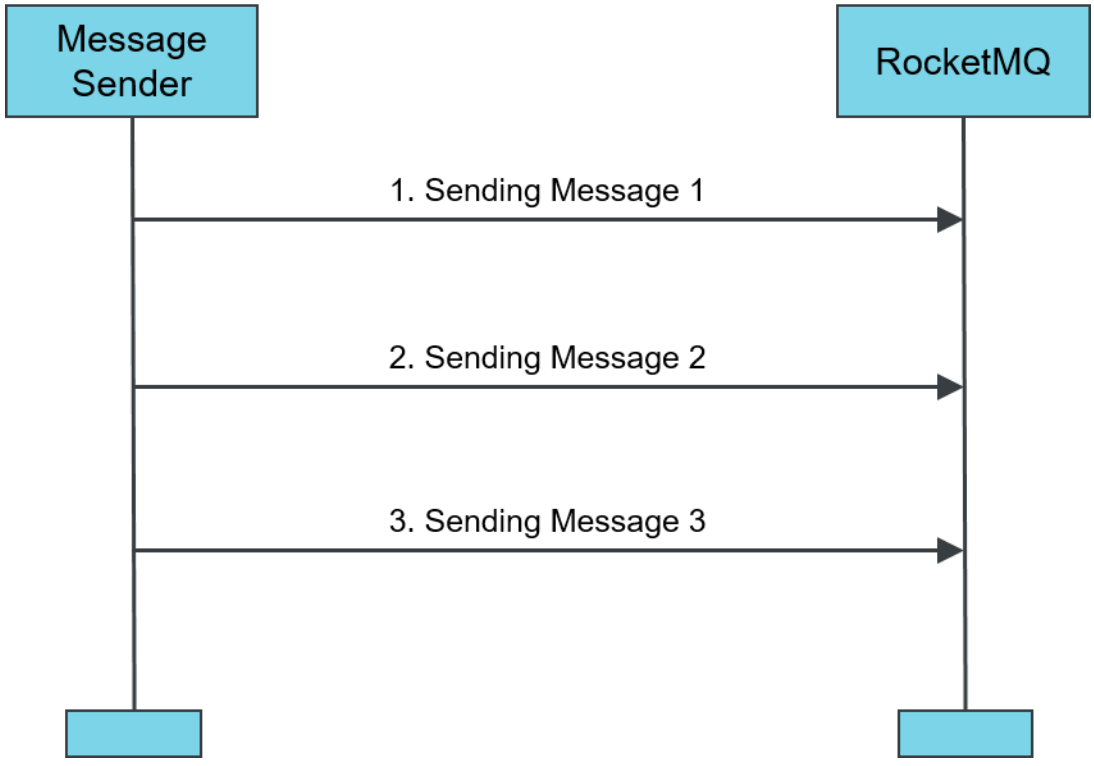
One-way transmission

This topic describes the principle and scenarios of the one-way transmission mode, and provides sample code.

## How it works

In one-way transmission mode, a producer only sends messages and does not wait for a response from the Message Queue for Apache RocketMQ broker. In addition, no callback function is triggered. In this mode, a message can be sent within microseconds.

One-way transmission




## Scenarios

This mode is applicable to scenarios where message transmission takes a short time and has no demanding reliability requirements. For example, this mode can be used for log collection.

The following table summarizes the features and major differences among the three modes.

| Transmission mode | Transactions per second (TPS) | Response | Reliability |
| --- | --- | --- | --- |
| Synchronous transmission | High | Supported | No message loss |

| Transmission mode | Transactions per second (TPS) | Response | Reliability |
| --- | --- | --- | --- |
| Asynchronous transmission | High | Supported | No message loss |
| One-way transmission | Highest | None | Possible message loss |

## Sample code

```
    import com.aliyun.openservices.ons.api.Message;
    import com.aliyun.openservices.ons.api.Producer;
    import com.aliyun.openservices.ons.api.ONSFactory;
    import com.aliyun.openservices.ons.api.PropertyKeyConst;
    import java.util.Properties;
    public static void main(String[] args) {
        Properties properties = new Properties();
        // The AccessKey ID that you created in the Apsara Uni-manager Management Console for identity authentication.
        properties.put(PropertyKeyConst.AccessKey, "XXX");
        // The AccessKey secret that you created in the Apsara Uni-manager Management Console for identity authentication.
        properties.put(PropertyKeyConst.SecretKey, "XXX");
        // The timeout interval for sending a message, in milliseconds.
        properties.setProperty(PropertyKeyConst.SendMsgTimeoutMillis, "3000");
        // The TCP endpoint. To obtain the endpoint, log on to the Message Queue for Apache RocketMQ console. In the left-side navigation pane, click Instance Details. On the Instance Details page, select your instance. On the Instance Information tab, view the endpoint in the Obtain Endpoint Information section.
        properties.put(PropertyKeyConst.NAMESRV_ADDR,
          "XXX");
        Producer producer = ONSFactory.createProducer(properties);
        // Before you use the producer to send a message, call the start() method once to start the producer.
        producer.start();
        // Cyclically send messages.
        for (int i = 0; i < 10; i++){
            Message msg = new Message(
                    // The topic of the message.
                    "TopicTestMQ",
                    // Message Tag,
                    // The message tag, which is similar to a Gmail tag. The message tag is used to sort messages and helps the consumer filter messages on the Message Queue for Apache RocketMQ broker based on specified conditions.
                    "TagA",
                    // Message Body
                    // The message body in the binary format. Message Queue for Apache RocketMQ does not process the message body. The producer and consumer must agree on the serialization and deserialization methods.
                    "Hello MQ".getBytes());
            // The key of the message. The key is the business-specific attribute of the message and must be globally unique whenever possible.
            // A unique key helps you query and resend a message in the Message Queue for A
```

```
pache RocketMQ console if the message fails to be received.
            // Note: Messages can be sent and received even if you do not specify the messa
ge key.
            msg.setKey("ORDERID_" + i);
            // In one-way transmission mode, the producer does not wait for the response fr
om the Message Queue for Apache RocketMQ broker. Therefore, data loss occurs if messages th
at fail to be delivered are not redelivered. If data loss is not acceptable, we recommend t
hat you use the reliable synchronous or asynchronous transmission mode.
            producer.sendOneway(msg);
        }
        // Before you exit the application, shut down the producer object.
        // Note: You can choose not to shut down the producer object.
        producer.shutdown();
    }
```

# 1.1.6.2.1.6. Send messages by using multiple threads

This topic describes how to send messages by using multiple threads and provides sample code.

The consumer and producer objects of Message Queue for Apache RocketMQ are thread-secure and can be shared among threads.

You can deploy multiple producer and consumer instances on one or more cloud servers. A producer or consumer instance can also run multiple threads to send or receive messages. This improves the transactions per second (TPS) for sending or receiving messages. Do not create a producer instance or consumer instance for every thread.

The following sample code provides an example on how to share a producer among threads:

```
import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.Producer;
import com.aliyun.openservices.ons.api.ONSFactory;
import com.aliyun.openservices.ons.api.PropertyKeyConst;
import com.aliyun.openservices.ons.api.SendResult;
import java.util.Properties;
public class SharedProducer {
    public static void main(String[] args) {
        // Initialize the configuration of the producer instance.
        Properties properties = new Properties();
        // The group ID that you created in the Message Queue for Apache RocketMQ console.
        properties.put(PropertyKeyConst.GROUP_ID, "XXX");
        // The AccessKey ID that you created in the Apsara Uni-manager Management Console f
or identity authentication.
        properties.put(PropertyKeyConst.AccessKey,"XXX");
        // The AccessKey secret that you created in the Apsara Uni-manager Management Conso
le for identity authentication.
        properties.put(PropertyKeyConst.SecretKey, "XXX");
        // The timeout interval for sending a message, in milliseconds.
        properties.setProperty(PropertyKeyConst.SendMsgTimeoutMillis, "3000");
        // The TCP endpoint. To obtain the endpoint, log on to the Message Queue for Apache
RocketMQ console. In the left-side navigation pane, click Instance Details. On the Instance
Details page, select your instance. On the Instance Information tab, view the endpoint in t
he Obtain Endpoint Information section.
        properties.put(PropertyKeyConst.NAMESRV_ADDR,
          "XXX");
```

```
            ^^^ );
        final Producer producer = ONSFactory.createProducer(properties);
        // Before you use the producer to send a message, call the start() method once to s
tart the producer.
        producer.start();
        // The created producer and consumer objects are thread-secure and can be shared am
ong threads. Do not create a producer instance or consumer instance for every thread.
        // Two threads share the producer object and concurrently send messages to Message
Queue for Apache RocketMQ.
        Thread thread = new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    Message msg = new Message( //
                    // The topic of the message.
                    "TopicTestMQ",
                    // The message tag, which is similar to a Gmail tag. The message tag is
used to sort messages and helps the consumer filter messages on the Message Queue for Apach
e RocketMQ broker based on specified conditions.
                    "TagA",
                    // The message body in the binary format. Message Queue for Apache Rock
etMQ does not process the message body.
                    // The producer and consumer must agree on the serialization and deseri
alization methods.
                    "Hello MQ".getBytes());
                    SendResult sendResult = producer.send(msg);
                    // Send the message in synchronous mode. If no error occurs, the messag
e is sent.
                    if (sendResult != null) {
                        System.out.println(new Date() + " Send mq message success. Topic is
:" + MqConfig.TOPIC + " msgId is: " + sendResult.getMessageId());
                    }
                } catch (Exception e) {
                    // Specify the logic to resend or persist the message if the message fa
ils to be sent.
                    System.out.println(new Date() + " Send mq message failed. Topic is:" +
MqConfig.TOPIC);
                    e.printStackTrace();
                }
            }
        });
        thread.start();
        Thread anotherThread = new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    Message msg = new Message("TopicTestMQ", "TagA", "Hello MQ".getBytes())
;
                    SendResult sendResult = producer.send(msg);
                    // Send the message in synchronous mode. If no error occurs, the messag
e is sent.
                    if (sendResult != null) {
                        System.out.println(new Date() + " Send mq message success. Topic is
:" + MqConfig.TOPIC + " msgId is: " + sendResult.getMessageId());
                    }
```

```
                } catch (Exception e) {
                    // Specify the logic to resend or persist the message if the message fa
ils to be sent.
                    System.out.println(new Date() + " Send mq message failed. Topic is:" +
MqConfig.TOPIC);
                    e.printStackTrace();
                }
            }
        });
        anotherThread.start();
        // If the producer instance is no longer used, shut it down to release resources.
        // producer.shutdown();
    }
}
```

# 1.1.6.2.1.7. Send and subscribe to ordered messages

This topic describes how to send and subscribe to ordered messages and provides sample code.

Ordered messages, also known as first-in-first-out (FIFO) messages, are a type of message provided by Message Queue for Apache RocketMQ. Such messages are published and consumed in a strict order. This topic provides the sample code for using TCP client SDK for Java to send and subscribe to ordered messages. For more information, see Ordered messages.

Use SDK for Java 1.2.7 or later to send and subscribe to ordered messages.

The methods of sending and subscribing to globally ordered messages and partitionally ordered messages are the same. The following code provides examples on how to send and subscribe to ordered messages:

## Sample code for sending ordered messages

```
package com.aliyun.openservices.ons.example.order;
import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.ONSFactory;
import com.aliyun.openservices.ons.api.PropertyKeyConst;
import com.aliyun.openservices.ons.api.SendResult;
import com.aliyun.openservices.ons.api.order.OrderProducer;
import java.util.Properties;
public class ProducerClient {
    public static void main(String[] args) {
        Properties properties = new Properties();
        // The group ID that you created in the Message Queue for Apache RocketMQ console.
        properties.put(PropertyKeyConst.GROUP_ID, "XXX");
        // The AccessKey ID that you created in the Apsara Uni-manager Management Console f
or identity authentication.
        properties.put(PropertyKeyConst.AccessKey, "XXX");
        // The AccessKey secret that you created in the Apsara Uni-manager Management Conso
le for identity authentication.
        properties.put(PropertyKeyConst.SecretKey, "XXX");
        // The TCP endpoint. To obtain the endpoint, log on to the Message Queue for Apache
RocketMQ console. In the left-side navigation pane, click Instance Details. On the Instance
Details page, select your instance. On the Instance Information tab, view the endpoint in t
he Obtain Endpoint Information section.
```

```
        properties.put(PropertyKeyConst.NAMESRV_ADDR,
          "XXX");
        OrderProducer producer = ONSFactory.createOrderProducer(properties);
        // Before you use the producer to send a message, call the start() method once to s
tart the producer.
        producer.start();
        for (int i = 0; i < 10; i++) {
            String orderId = "biz_" + i % 10;
            Message msg = new Message(//
                    // The topic of the message.
                    "Order_global_topic",
                    // The message tag, which is similar to a Gmail tag. The message tag is
used to sort messages and helps the consumer filter messages on the Message Queue for Apach
e RocketMQ broker based on specified conditions.
                    "TagA",
                    // The message body in the binary format. Message Queue for Apache Rock
etMQ does not process the message body. The producer and consumer must agree on the seriali
zation and deserialization methods.
                    "send order global msg".getBytes()
            );
            // The key of the message. The key is the business-specific attribute of the me
ssage and must be globally unique whenever possible.
            // A unique key helps you query and resend a message in the Message Queue for A
pache RocketMQ console if the message fails to be received.
            // Note: Messages can be sent and received even if you do not specify the messa
ge key.
            msg.setKey(orderId);
            // The key field that is used in ordered messages to distinguish among differen
t partitions. A partition key is different from the key of a normal message.
            // This field can be set to a non-empty string for globally ordered messages.
            String shardingKey = String.valueOf(orderId);
            try {
                SendResult sendResult = producer.send(msg, shardingKey);
                // Send the message. If no error occurs, the message is sent.
                if (sendResult != null) {
                    System.out.println(new Date() + " Send mq message success. Topic is:" +
msg.getTopic() + " msgId is: " + sendResult.getMessageId());
                }
            }
            catch (Exception e) {
                // Specify the logic to resend or persist the message if the message fails
to be sent.
                System.out.println(new Date() + " Send mq message failed. Topic is:" + msg.
getTopic());
                e.printStackTrace();
            }
        }
        // Before you exit the application, shut down the producer object.
        // Note: You can choose not to shut down the producer object.
        producer.shutdown();
    }
}
```

## Sample code for subscribing to ordered messages

```
package com.aliyun.openservices.ons.example.order;
import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.ONSFactory;
import com.aliyun.openservices.ons.api.PropertyKeyConst;
import com.aliyun.openservices.ons.api.order.ConsumeOrderContext;
import com.aliyun.openservices.ons.api.order.MessageOrderListener;
import com.aliyun.openservices.ons.api.order.OrderAction;
import com.aliyun.openservices.ons.api.order.OrderConsumer;
import java.util.Properties;
public class ConsumerClient {
    public static void main(String[] args) {
        Properties properties = new Properties();
        // The group ID that you created in the Message Queue for Apache RocketMQ console.
        properties.put(PropertyKeyConst.GROUP_ID, "XXX");
        // The AccessKey ID that you created in the Apsara Uni-manager Management Console f
or identity authentication.
        properties.put(PropertyKeyConst.AccessKey, "XXX");
        // The AccessKey secret that you created in the Apsara Uni-manager Management Conso
le for identity authentication.
        properties.put(PropertyKeyConst.SecretKey, "XXX");
        // The TCP endpoint. To obtain the endpoint, log on to the Message Queue for Apache
RocketMQ console. In the left-side navigation pane, click Instance Details. On the Instance
Details page, select your instance. On the Instance Information tab, view the endpoint in t
he Obtain Endpoint Information section.
        properties.put(PropertyKeyConst.NAMESRV_ADDR,
          "XXX");
          // The time to wait to redeliver the ordered message when the message fails to be
consumed. Valid values: 10 to 1800. Unit: milliseconds.
        properties.put(PropertyKeyConst.SuspendTimeMillis, "100");
        // The maximum number of delivery retries when the message fails to be consumed.
        properties.put(PropertyKeyConst.MaxReconsumeTimes, "20");
        // Before you use the consumer to subscribe to a message, call the start() method o
nce to start the consumer.
        OrderConsumer consumer = ONSFactory.createOrderedConsumer(properties);
        consumer.subscribe(
                // The topic of the message.
                "Jodie_Order_Topic",
                // Subscribe to messages with specified tags in the specified topic.
                // 1. * indicates that the consumer subscribes to all messages in the speci
fied topic.
                // 2. TagA || TagB || TagC indicates that the consumer subscribes to messag
es with TagA, TagB, or TagC.
                "*",
                new MessageOrderListener() {
                    /**
                     * 1. OrderAction.Suspend is returned if a message fails to be consumed
or an exception occurs during message processing.<br>
                     * 2. OrderAction.Success is returned if a message is processed.
                     */
                    @Override
                    public OrderAction consume(Message message, ConsumeOrderContext context
) {
```

```
                        System.out.println(message);
                        return OrderAction.Success;
                    }
                });
        consumer.start();
    }
}
```

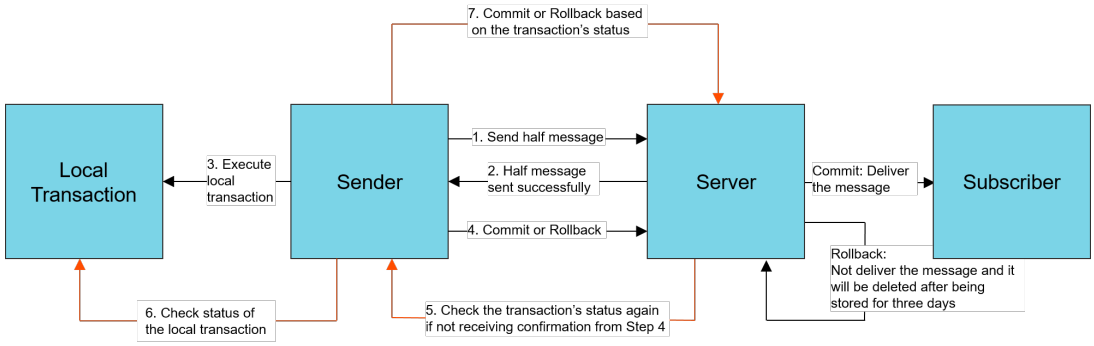
# 1.1.6.2.1.8. Send and subscribe to transactional messages

This topic describes the interaction process and the back-check mechanism of transactional messages. This topic also shows you how to send and subscribe to transactional messages, and provides sample code.

## Interaction process

Interaction process of transactional messages shows the interaction process of transactional messages in Message Queue for Apache RocketMQ.

Interaction process of transactional messages




## Send transactional messages

Perform the following steps to send a transactional message:

1. Send a half message and execute a local transaction. The following code provides an example:

```
package com.alibaba.webx.TryHsf.app1;
import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.PropertyKeyConst;
import com.aliyun.openservices.ons.api.SendResult;
import com.aliyun.openservices.ons.api.transaction.LocalTransactionExecuter;
import com.aliyun.openservices.ons.api.transaction.TransactionProducer;
import com.aliyun.openservices.ons.api.transaction.TransactionStatus;
import java.util.Properties;
import java.util.concurrent.TimeUnit;
public class TransactionProducerClient {
 private final static Logger log = ClientLogger.getLog(); // Configure logging to facil
itate troubleshooting.
 public static void main(String[] args) throws InterruptedException {
     final BusinessService businessService = new BusinessService(); // Your on-premises
business
```

```
business.
     Properties properties = new Properties();
     // The group ID that you created in the Message Queue for Apache RocketMQ console.
Note: Transactional messages cannot share group IDs with other types of messages.
     properties.put(PropertyKeyConst.GROUP_ID, "XXX");
     // The AccessKey ID that you created in the Apsara Uni-manager Management Console
for identity authentication.
     properties.put(PropertyKeyConst.AccessKey, "XXX");
     // The AccessKey secret that you created in the Apsara Uni-manager Management Cons
ole for identity authentication.
     properties.put(PropertyKeyConst.SecretKey, "XXX");
     // The TCP endpoint. To obtain the endpoint, log on to the Message Queue for Apach
e RocketMQ console. In the left-side navigation pane, click Instance Details. On the In
stance Details page, select your instance. On the Instance Information tab, view the en
dpoint in the Obtain Endpoint Information section.
     properties.put(PropertyKeyConst.NAMESRV_ADDR,
       "XXX");
     TransactionProducer producer = ONSFactory.createTransactionProducer(properties,
             new LocalTransactionCheckerImpl());
     producer.start();
     Message msg = new Message("Topic", "TagA", "Hello MQ transaction===".getBytes());
     try {
             SendResult sendResult = producer.send(msg, new LocalTransactionExecuter()
{
                 @Override
                 public TransactionStatus execute(Message msg, Object arg) {
                     // The ID of the message. Two messages may have the same message b
ody but cannot have the same ID. The current message ID cannot be queried in the consol
e.
                     String msgId = msg.getMsgID();
                     // Calculate the message body by using CRC32 or other algorithms,
such as MD5.
                     long crc32Id = HashUtil.crc32Code(msg.getBody());
                     // The message ID and CRC32 ID are used to prevent duplicate messa
ges.
                     // You do not need to specify the message ID or CRC32 ID if your b
usiness itself achieves idempotence. Otherwise, specify the message ID or CRC32 ID to e
nsure idempotence.
                     // To prevent duplicate messages, calculate the message body by us
ing the CRC32 or MD5 algorithm.
                     Object businessServiceArgs = new Object();
                     TransactionStatus transactionStatus = TransactionStatus.Unknow;
                     try {
                         boolean isCommit =
                             businessService.execbusinessService(businessServiceArgs);
                         if (isCommit) {
                             // Commit the message if the local transaction succeeds.
                             transactionStatus = TransactionStatus.CommitTransaction;
                         } else {
                             // Roll back the message if the local transaction fails.
                             transactionStatus = TransactionStatus.RollbackTransaction;
                         }
                     } catch (Exception e) {
                         log.error("Message Id:{}", msgId, e);
                     }
```

```
                    System.out.println(msg.getMsgID());
                    log.warn("Message Id:{}transactionStatus:{}", msgId, transactionSt
atus.name());
                    return transactionStatus;
                }
            }, null);
        }
        catch (Exception e) {
            // Specify the logic to resend or persist the message if the message fails
to be sent.
            System.out.println(new Date() + " Send mq message failed. Topic is:" + msg
.getTopic());
            e.printStackTrace();
        }
    // Use the demo example to prevent the process from exiting. This is not required
in actual use.
    TimeUnit.MILLISECONDS.sleep(Integer.MAX_VALUE);
 }
}
```

2. Commit the status of the transactional message.

   After the local transaction is executed, the Message Queue for Apache RocketMQ broker must be notified of the transaction status of the current message no matter whether the execution is successful or fails. The Message Queue for Apache RocketMQ broker can be notified in one of the following ways:

   - Commit the status after the local transaction is executed.
   - Wait until the Message Queue for Apache RocketMQ broker sends a request to check the transaction status of the message.

   A transaction can be in one of the following states:

   - TransactionStatus.CommitTransaction: The transaction is committed. The consumer can consume the message.
   - TransactionStatus.RollbackTransaction: The transaction is rolled back. The message is discarded and cannot be consumed.
   - TransactionStatus.Unknow: The status of the transaction is unknown. The Message Queue for Apache RocketMQ broker is expected to send a request again to the producer to query the status of the local transaction that corresponds to the message.

```
public class LocalTransactionCheckerImpl implements LocalTransactionChecker {
   private final static Logger log = ClientLogger.getLog();
   final  BusinessService businessService = new BusinessService();
   @Override
   public TransactionStatus check(Message msg) {
       // The ID of the message. Two messages may have the same message body but cannot
have the same ID. The current message is a half message. Therefore, its message ID cann
ot be queried in the console.
       String msgId = msg.getMsgID();
       // Calculate the message body by using CRC32 or other algorithms, such as MD5.
       long crc32Id = HashUtil.crc32Code(msg.getBody());
       // The message ID and CRC32 ID are used to prevent duplicate messages.
       // You do not need to specify the message ID or CRC32 ID if your business itself
achieves idempotence. Otherwise, specify the message ID or CRC32 ID to ensure idempoten
ce.
       // To prevent duplicate messages, calculate the message body by using the CRC32
or MD5 algorithm.
       // The parameter object of your business. Specify the object based on your busin
ess.
       Object businessServiceArgs = new Object();
       TransactionStatus transactionStatus = TransactionStatus.Unknow;
       try {
           boolean isCommit = businessService.checkbusinessService(businessServiceArgs)
;
           if (isCommit) {
               // Commit the message if the local transaction succeeds.
               transactionStatus = TransactionStatus.CommitTransaction;
           } else {
               // Roll back the message if the local transaction fails.
               transactionStatus = TransactionStatus.RollbackTransaction;
           }
       } catch (Exception e) {
           log.error("Message Id:{}", msgId, e);
       }
       log.warn("Message Id:{}transactionStatus:{}", msgId, transactionStatus.name());
       return transactionStatus;
   }
 }
```

### Utility class

```
import java.util.zip.CRC32;
public class HashUtil {
    public static long crc32Code(byte[] bytes) {
        CRC32 crc32 = new CRC32();
        crc32.update(bytes);
        return crc32.getValue();
    }
}
```

### Back-check mechanism for transaction status

- Why must the back-check mechanism for transaction status be implemented when transactional

messages are sent?

If the half message is sent in Step 1 but TransactionStatus.Unknow is returned for the local transaction, or no status is committed for the local transaction because the application exits, the status of the half message is unknown to the Message Queue for Apache RocketMQ broker. Therefore, the Message Queue for Apache RocketMQ broker periodically requests the producer to check and report the status of the half message.

- What does the business logic do when the check method is called back?

  The check method for transactional messages in Message Queue for Apache RocketMQ must contain the logic of transaction consistency check. After a transactional message is sent, Message Queue for Apache RocketMQ must call the LocalTransactionChecker method to respond to the request of the Message Queue for Apache RocketMQ broker for the status of the local transaction. Therefore, the check method for transactional messages must contain the following check items:

  i. Check the status of the local transaction that corresponds to the half message. The status is committed or rollback.

  ii. Commit the status of the local transaction that corresponds to the half message to the Message Queue for Apache RocketMQ broker.

### Subscribe to transactional messages

The method for subscribing to transactional messages is the same as that for subscribing to normal messages. For more information, see Subscribe to messages.

## 1.1.6.2.1.9. Send and subscribe to delayed messages

This topic describes how to send and subscribe to delayed messages and provides sample code.

Delayed messages are delivered to a consumer after a specified period of time from when they are sent to the Message Queue for Apache RocketMQ broker. For example, the specified period of time can be 3 seconds. Delayed messages are used in scenarios where a time window between message production and consumption is required or tasks need to be triggered after a delay. Delayed messages are used in a similar way to delay queues.

For more information about the concepts and usage notes of delayed messages, see Scheduled messages and delayed messages.

### Send delayed messages

The following sample code provides an example on how to send delayed messages:

```
import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.ONSFactory;
import com.aliyun.openservices.ons.api.Producer;
import com.aliyun.openservices.ons.api.PropertyKeyConst;
import com.aliyun.openservices.ons.api.SendResult;
import java.util.Properties;
public class ProducerDelayTest {
    public static void main(String[] args) {
        Properties properties = new Properties();
        // The AccessKey ID that you created in the Apsara Uni-manager Management Console f
or identity authentication.
        properties.put(PropertyKeyConst.AccessKey, "XXX");
        // The AccessKey secret that you created in the Apsara Uni-manager Management Conso
```

```
ie for identity authentication.
        properties.put(PropertyKeyConst.SecretKey, "XXX");
        // The TCP endpoint. To obtain the endpoint, log on to the Message Queue for Apache
RocketMQ console. In the left-side navigation pane, click Instance Details. On the Instance
Details page, select your instance. On the Instance Information tab, view the endpoint in t
he Obtain Endpoint Information section.
        properties.put(PropertyKeyConst.NAMESRV_ADDR,
          "XXX");
        Producer producer = ONSFactory.createProducer(properties);
        // Before you use the producer to send a message, call the start() method once to s
tart the producer.
        producer.start();
        Message msg = new Message( //
                // The topic that you created in the Message Queue for Apache RocketMQ cons
ole.
                "Topic",
                // The message tag, which is similar to a Gmail tag. The message tag is use
d to sort messages and helps the consumer filter messages on the Message Queue for Apache R
ocketMQ broker based on specified conditions.
                "tag",
                // The message body in the binary format. Message Queue for Apache RocketMQ
does not process the message body. The producer and consumer must agree on the serializatio
n and deserialization methods.
                "Hello MQ".getBytes());
        // The key of the message. The key is the business-specific attribute of the messag
e and must be globally unique whenever possible.
        // A unique key helps you query and resend a message in the Message Queue for Apach
e RocketMQ console if the message fails to be received.
        // Note: Messages can be sent and received even if you do not specify the message k
ey.
        msg.setKey("ORDERID_100");
        try {
            // The specified period of time, in milliseconds. After the specified period of
time elapses, the Message Queue for Apache RocketMQ broker delivers the message to the cons
umer. For example, you can set this parameter to 3 and the Message Queue for Apache RocketM
Q broker delivers the message to the consumer after 3 seconds. The value must be later than
the current time.
            long delayTime = System.currentTimeMillis() + 3000;
            // The time when the Message Queue for Apache RocketMQ broker starts to deliver
the message.
            msg.setStartDeliverTime(delayTime);
            SendResult sendResult = producer.send(msg);
            // Send the message in synchronous mode. If no error occurs, the message is sen
t.
            if (sendResult != null) {
            System.out.println(new Date() + " Send mq message success. Topic is:" + msg.get
Topic() + " msgId is: " + sendResult.getMessageId());
            }
            } catch (Exception e) {
            // Specify the logic to resend or persist the message if the message fails to b
e sent.
            System.out.println(new Date() + " Send mq message failed. Topic is:" + msg.getT
opic());
            e.printStackTrace();
        }
```

```
        }
        // Before you exit the application, shut down the producer object.<br>
        // Note: You can choose not to shut down the producer object.
        producer.shutdown();
    }
}
```

## Subscribe to delayed messages

The method for subscribing to delayed messages is the same as that for subscribing to normal messages. For more information, see Subscribe to messages.

# 1.1.6.2.1.10. Send and subscribe to scheduled messages

This topic describes the scenarios for sending and subscribing to scheduled messages and provides sample code.

Scheduled messages are consumed after a specified timestamp. Such messages are used in scenarios where a time window between message production and consumption is required or tasks need to be triggered at a scheduled time.

For more information about the concepts and usage notes of scheduled messages, see Scheduled messages and delayed messages.

## Send scheduled messages

The following sample code provides an example on how to send scheduled messages:

```
import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.ONSFactory;
import com.aliyun.openservices.ons.api.Producer;
import com.aliyun.openservices.ons.api.PropertyKeyConst;
import com.aliyun.openservices.ons.api.SendResult;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Properties;
public class ProducerDelayTest {
    public static void main(String[] args) {
        Properties properties = new Properties();
        // The AccessKey ID that you created in the Apsara Uni-manager Management Console f
or identity authentication.
        properties.put(PropertyKeyConst.AccessKey, "XXX");
        // The AccessKey secret that you created in the Apsara Uni-manager Management Conso
le for identity authentication.
        properties.put(PropertyKeyConst.SecretKey, "XXX");
        // The TCP endpoint. To obtain the endpoint, log on to the Message Queue for Apache
RocketMQ console. In the left-side navigation pane, click Instance Details. On the Instance
Details page, select your instance. On the Instance Information tab, view the endpoint in t
he Obtain Endpoint Information section.
        properties.put(PropertyKeyConst.NAMESRV_ADDR,
          "XXX");
        Producer producer = ONSFactory.createProducer(properties);
        // Before you use the producer to send a message, call the start() method once to s
tart the producer.
        producer.start();
```

```
        Message msg = new Message( //
                // The topic of the message.
                "Topic",
                // The message tag, which is similar to a Gmail tag. The message tag is used to sort messages and helps the consumer filter messages on the Message Queue for Apache RocketMQ broker based on specified conditions.
                "tag",
                // The message body in the binary format. Message Queue for Apache RocketMQ does not process the message body. The producer and consumer must agree on the serialization and deserialization methods.
                "Hello MQ".getBytes());
        // The key of the message. The key is the business-specific attribute of the message and must be globally unique whenever possible.
        // A unique key helps you query and resend a message in the Message Queue for Apache RocketMQ console if the message fails to be received.
        // Note: Messages can be sent and received even if you do not specify the message key.
        msg.setKey("ORDERID_100");
        try {
            // The time when the Message Queue for Apache RocketMQ broker delivers the message to the consumer, in milliseconds. For example, you can set this parameter to 2016-03-07 16:21:00 and the broker delivers the message at 16:21:00 on March 7, 2016. The value must be later than the current time. If the scheduled time is earlier than the current time, the message is immediately delivered to the consumer.
            long timeStamp = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").parse("2016-03-07 16:21:00").getTime();
            msg.setStartDeliverTime(timeStamp);
            // Send the message. If no error occurs, the message is sent.
            SendResult sendResult = producer.send(msg);
            System.out.println("Message Id:" + sendResult.getMessageId());
        }
        catch (Exception e) {
            // Specify the logic to resend or persist the message if the message fails to be sent.
            System.out.println(new Date() + " Send mq message failed. Topic is:" + msg.getTopic());
            e.printStackTrace();
        }
        // Before you exit the application, shut down the producer object.
        // Note: You can choose not to shut down the producer object.
        producer.shutdown();
    }
}
```

## Subscribe to scheduled messages

The method for subscribing to scheduled messages is the same as that for subscribing to normal messages. For more information, see Subscribe to messages.

# 1.1.6.2.1.11. Subscribe to messages

This topic describes message subscription modes and provides sample code.

> **Note** The subscriptions of all consumer instances identified by the same group ID must be consistent. For more information, see Subscription consistency.

## Subscription modes

Message Queue for Apache RocketMQ supports the following message subscription modes:

- **Clustering subscription:** In this mode, all the consumer instances identified by the same group ID evenly share messages. Assume that a topic contains nine messages and a group ID identifies three consumer instances. In clustering consumption mode, each instance consumes three messages.

  ```
  // Configure clustering subscription, which is the default mode.
  properties.put(PropertyKeyConst.MessageModel, PropertyValueConst.CLUSTERING);
  ```

- **Broadcasting subscription:** In this mode, each consumer instance identified by a group ID consumes each message once. Assume that a topic contains nine messages and a group ID identifies three consumer instances. In broadcasting consumption mode, each instance consumes nine messages.

  ```
  // Configure broadcasting subscription.
  properties.put(PropertyKeyConst.MessageModel, PropertyValueConst.BROADCASTING);
  ```

## Sample code

```
import com.aliyun.openservices.ons.api.Action;
import com.aliyun.openservices.ons.api.ConsumeContext;
import com.aliyun.openservices.ons.api.Consumer;
import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.MessageListener;
import com.aliyun.openservices.ons.api.ONSFactory;
import com.aliyun.openservices.ons.api.PropertyKeyConst;
import java.util.Properties;
public class ConsumerTest {
    public static void main(String[] args) {
        Properties properties = new Properties();
        // The group ID that you created in the Message Queue for Apache RocketMQ console.
        properties.put(PropertyKeyConst.GROUP_ID, "XXX");
        // The AccessKey ID that you created in the Apsara Uni-manager Management Console f
or identity authentication.
        properties.put(PropertyKeyConst.AccessKey, "XXX");
        // The AccessKey secret that you created in the Apsara Uni-manager Management Conso
le for identity authentication.
        properties.put(PropertyKeyConst.SecretKey, "XXX");
        // The TCP endpoint. To obtain the endpoint, log on to the Message Queue for Apache
RocketMQ console. In the left-side navigation pane, click Instance Details. On the Instance
Details page, select your instance. On the Instance Information tab, view the endpoint in t
he Obtain Endpoint Information section.
        properties.put(PropertyKeyConst.NAMESRV_ADDR,
          "XXX");
          // Clustering subscription, which is the default mode.
          // properties.put(PropertyKeyConst.MessageModel, PropertyValueConst.CLUSTERING);
          // Broadcasting subscription.
          // properties.put(PropertyKeyConst.MessageModel, PropertyValueConst.BROADCASTING)
;
        Consumer consumer = ONSFactory.createConsumer(properties);
        consumer.subscribe("TopicTestMQ", "TagA||TagB", new MessageListener() { // Subscrib
e to multiple tags.
            public Action consume(Message message, ConsumeContext context) {
                System.out.println("Receive: " + message);
                return Action.CommitMessage;
            }
        });
        // Subscribe to another topic.
        consumer.subscribe("TopicTestMQ-Other", "*", new MessageListener() { // Subscribe t
o all tags.
            public Action consume(Message message, ConsumeContext context) {
                System.out.println("Receive: " + message);
                return Action.CommitMessage;
            }
        });
        consumer.start();
        System.out.println("Consumer Started");
    }
}
```

> **Note** In broadcasting consumption mode, you cannot query message accumulation information in the Message Queue for Apache RocketMQ console. You can create multiple group IDs to achieve the effect of broadcasting consumption. For more information, see Clustering consumption and broadcasting consumption.

# 1.1.6.2.2. SDK for C or C++

# 1.1.6.2.2.1. Prepare the SDK for C or C++ environment

Overview

Make sure that the following prerequisites are met before you use SDK for C++ to access Message Queue for Apache RocketMQ:

> **Note**
> - You have created the topics and group IDs involved in the code in the Message Queue for Apache RocketMQ console. You can customize message tags in your application. For more information about how to create a message tag, see Create resources.
> - Applications that use Message Queue for Apache RocketMQ are deployed on Elastic Compute Service (ECS) instances.

Download SDK for C++

This topic describes the preparations, instructions, and usage notes for using SDK for C++ to access Message Queue for Apache RocketMQ so that you can use SDK for C++ to send and subscribe to messages.

## Procedure

1. Download SDK for C++ used in Linux.
2. Decompress the downloaded package.

   After the package is decompressed, the following directory structure appears:

   - *demos/*

     Contains examples on how to send and consume normal messages and ordered messages and how to send messages in one-way transmission mode. This directory also contains the *CMakeList.txt* file that is used to compile and manage *demos*.

   - *include/*

     Contains header files that are required by your own programs.

   - *lib/*

     Contains dynamic libraries based on x86_64. The libraries include the libonsclient4cpp.so interface library and the librocketmq_client_core.so core library.

   - *changelog*

     Contains bug fixes and new features in the new releases.

Use SDK for C++ in Linux

This topic describes how to use SDK for C++ in Linux.

Starting June 28, 2019, the new SDK version provides only dynamic library solutions. The library file of Message Queue for Apache RocketMQ is stored in the *lib/* directory. You must link *librocketmq_client_core.so* with *libonsclient4cpp.so* when you generate executable files. *demos* has introduced the features of C++ 11 and uses CMake for management. Therefore, you must install CMake 3.0 or later and g++ 4.8 or later in advance.

## Dynamic solution

GCC 5.x or later has introduced Dual ABI. Therefore, you must add the -D_GLIBCXX_USE_CXX11_ABI=0 option when you compile the preceding links.

The following sample code provides an example on how to use *demos*:

```
cd aliyun-mq-linux-cpp-sdk // The path to which the downloaded SDK package is decompressed.
cd demos // Go to the demos directory and modify the demos file by entering information suc
h as the topic and key that you created in the Message Queue for Apache RocketMQ console.
cmake  .   // Check the dependencies and generate a compilation script.
make // Compile the code.
cd bin // Run the program in the directory where the generated executable files are located
.
```

# 1.1.6.2.2.2. Send and subscribe to normal messages

This topic provides the sample code for sending and subscribing to normal messages.

## Send normal messages

The following sample code provides an example on how to send normal messages:

```
Failed to resolve content from t1841033.dita#concept_2047098/codeblock_023_jzy_dbr
```

## Subscribe to normal messages

For more information about how to subscribe to normal messages and about relevant sample code, see Subscribe to messages.

# 1.1.6.2.2.3. Send and subscribe to ordered messages

This topic provides the sample code for sending and subscribing to ordered messages.

## Send ordered messages

The following sample code provides an example on how to send ordered messages:

```
Failed to resolve content from t1841034.dita#concept_2047099/codeblock_ps2_wtp_4vc
```

## Subscribe to ordered messages

The following sample code provides an example on how to subscribe to ordered messages:

```
Failed to resolve content from t1841034.dita#concept_2047099/codeblock_b62_xrh_2xw
```

# 1.1.6.2.2.4. Send and subscribe to scheduled messages

This topic provides the sample code for sending and subscribing to scheduled messages.

Scheduled messages are consumed by consumers after a specified period of time. Such messages are used in scenarios where a time window between message production and consumption is required or tasks need to be triggered at a scheduled time. Scheduled messages are used in a similar way to delay queues.

## Send scheduled messages

The following sample code provides an example on how to send scheduled messages:

```
#include "ONSFactory.h"
#include "ONSClientException.h"
using namespace ons;
int main()
{
    // Create a producer instance and configure the information required to send messages.
    ONSFactoryProperty factoryInfo = new ONSFactoryProperty();
    factoryInfo.setFactoryProperty(ONSFactoryProperty::ProducerId, "XXX");//The group ID yo
u created in the
Message Queue for Apache RocketMQ console.
    factoryInfo.setFactoryProperty(ONSFactoryProperty::NAMESRV_ADDR, "XXX"); //The TCP endp
oint. Go to the Instances page in the
Message Queue for Apache RocketMQ console, and view the endpoint in the Endpoint Informatio
n section.
    factoryInfo.setFactoryProperty(ONSFactoryProperty::PublishTopics,"XXX" );//The topic yo
u created in the
Message Queue for Apache RocketMQ console.
    factoryInfo.setFactoryProperty(ONSFactoryProperty::MsgContent, "xxx");//The message con
tent.
    factoryInfo.setFactoryProperty(ONSFactoryProperty::AccessKey, "xxx");//The AccessKey ID
you created in the
Message Queue for Apache RocketMQ console for identity authentication.
    factoryInfo.setFactoryProperty(ONSFactoryProperty::SecretKey, "xxx" );//The AccessKey s
ecret you created in the
Message Queue for Apache RocketMQ console for identity authentication.
    // Create a producer instance.
    Producer *pProducer = ONSFactory::getInstance()->createProducer(factoryInfo);
    // Before sending a message, call the start method once to start the producer.
    pProducer->start();
    Message msg(
            // The message topic.
            factoryInfo.getPublishTopics(),
            // The message tag, which is similar to a Gmail tag. It is used to sort message
s, enabling the consumer to filter messages on the Message Queue for Apache RocketMQ broker
based on the specified criteria.
            "TagA",
            // The message body, which cannot be empty.
Message Queue for Apache RocketMQ does not process the message body. The producer and consu
mer must negotiate consistent serialization and deserialization methods.
            factoryInfo.getMessageContent()
    );
```

```
    // The message key, which must be globally unique.
    // A unique identifier enables you to query a message and resend it in the
Message Queue for Apache RocketMQ console if you fail to receive the message.
    // Note: Messages can still be sent and received even if this attribute is not set.
    msg.setKey("ORDERID_100");
    // The delivery time, in ms. After the time is specified, a message can be consumed onl
y after this time. In this example, a message can be consumed 3 seconds later.
    long deliverTime = Current system time (ms) + 3000;
    msg.setStartDeliverTime(deliverTime);
    // The message sending result, which is successful if no exception occurs.
    try
    {
        SendResultONS sendResult = pProducer->send(msg);
    }
    catch(ONSClientException & e)
    {
        // Customize exception handling details.
    }
    // Destroy the producer object before exiting the application. Otherwise, memory leakag
e may occur.
    pProducer->shutdown();
    return 0;
}
```

### Subscribe to scheduled messages

For more information about how to subscribe to scheduled messages and about relevant sample code, see Subscribe to messages.

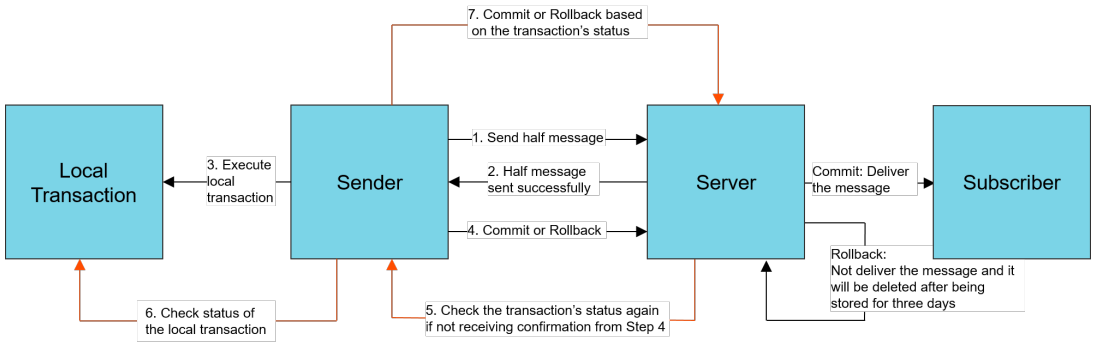
# 1.1.6.2.2.5. Send and subscribe to transactional messages

This topic describes the interaction process and the back-check mechanism of transactional messages. This topic also shows you how to send and subscribe to transactional messages, and provides sample code.

### Interaction process

Transactional message interaction flowchart shows the interaction process of transactional messages in Message Queue for Apache RocketMQ.

Interaction process of transactional messages




## Send transactional messages

Perform the following steps to send a transactional message:

1. Send a half message and execute a local transaction. The following code provides examples on how to send and subscribe to transactional messages:

   ```
   Failed to resolve content from t1841036.dita#concept_2047101/codeblock_sjf_ek1_fei
   ```

2. Commit the status of the transactional message.

   After the local transaction is executed, the Message Queue for Apache RocketMQ broker must be notified of the transaction status of the current message no matter whether the execution is successful or fails. The Message Queue for Apache RocketMQ broker can be notified in one of the following ways:

   - Commit the status after the local transaction is executed.
   - Wait until the Message Queue for Apache RocketMQ broker sends a request to check the transaction status of the message.

A transaction can be in one of the following states:

- TransactionStatus.CommitTransaction: The transaction is committed. The consumer can consume the message.
- TransactionStatus.RollbackTransaction: The transaction is rolled back. The message is discarded and cannot be consumed.
- TransactionStatus.Unknow: The status of the transaction is unknown. The Message Queue for Apache RocketMQ broker is expected to send a request again to the producer to query the status of the local transaction that corresponds to the message.

```
class MyLocalTransactionChecker : LocalTransactionChecker
{
    MyLocalTransactionChecker()
    {
    }
    ~MyLocalTransactionChecker()
    {
    }
    virtual TransactionStatus check(Message &value)
    {
        // The ID of the message. The current message ID cannot be queried in the console. Two messages may have the same message body but cannot have the same ID.
        string msgId = value.getMsgID();
        // Calculate the message body by using CRC32 or other algorithms, such as MD5.
        // The message ID and CRC32 ID are used to prevent duplicate messages.
        // You do not need to specify the message ID or CRC32 ID if your business itself achieves idempotence. Otherwise, specify the message ID or CRC32 ID to ensure idempotence.
        // To prevent duplicate messages, calculate the message body by using the CRC32 or MD5 algorithm.
        TransactionStatus transactionStatus = Unknow;
        try {
            boolean isCommit = Execution result of the local transaction;
            if (isCommit) {
                // Commit the message if the local transaction succeeds.
                transactionStatus = CommitTransaction;
            } else {
                // Roll back the message if the local transaction fails.
                transactionStatus = RollbackTransaction;
            }
        } catch(...) {
            //exception error
        }
        return transactionStatus;
    }
}
```

### Back-check mechanism for transaction status

- Why must the back-check mechanism for transaction status be implemented when transactional messages are sent?

  If the half message is sent in Step 1 but TransactionStatus.Unknow is returned for the local transaction, or no status is committed for the local transaction because the application exits, the status of the half message is unknown to the Message Queue for Apache RocketMQ broker. Therefore, the Message Queue for Apache RocketMQ broker periodically requests the producer to check and report the status of the half message.

- What does the business logic do when the check method is called back?

  The check method for transactional messages in Message Queue for Apache RocketMQ must contain the logic of transaction consistency check. After a transactional message is sent, Message Queue for Apache RocketMQ must call the LocalTransactionChecker method to respond to the request of the Message Queue for Apache RocketMQ broker for the status of the local transaction. Therefore, the check method for transactional messages must contain the following check items:

i. Check the status of the local transaction that corresponds to the half message. The status is committed or rollback.

ii. Commit the status of the local transaction that corresponds to the half message to the Message Queue for Apache RocketMQ broker.

- How do different states of the local transaction affect the half message?
  - TransactionStatus.CommitTransaction: The transaction is committed. The consumer can consume the message.
  - TransactionStatus.RollbackTransaction: The transaction is rolled back. The message is discarded and cannot be consumed.
  - TransactionStatus.Unknow: The status of the transaction is unknown. The Message Queue for Apache RocketMQ broker is expected to send a request again to the producer to query the status of the local transaction that corresponds to the message.

For more information about the code, see the implementation of MyLocalTransactionChecker.

## Subscribe to transactional messages

For more information about how to subscribe to transactional messages and about relevant sample code, see Subscribe to messages.

# 1.1.6.2.2.6. Subscribe to messages

This topic describes how to subscribe to messages by using SDK for C or C++ provided by Message Queue for Apache RocketMQ.

> **Note** The subscriptions of all consumer instances identified by the same group ID must be consistent. For more information, see Subscription consistency.

## Subscription modes

Message Queue for Apache RocketMQ supports the following message subscription modes:

- **Clustering subscription:**

  This mode is used to implement clustering consumption. In clustering consumption mode, all the consumer instances identified by the same group ID evenly share messages. Assume that a topic contains nine messages and a group ID identifies three consumer instances. In clustering consumption mode, each instance consumes three messages.

```
 // Configure clustering subscription, which is the default mode.
 factoryInfo.setFactoryProperty(ONSFactoryProperty:: MessageModel, ONSFactoryProperty::CL
USTERING);
```

- **Broadcasting subscription:**

  This mode is used to implement broadcasting consumption. In broadcasting consumption mode, each consumer instance identified by a group ID consumes a message once. Assume that a topic contains nine messages and a group ID identifies three consumer instances. In broadcasting consumption mode, each instance consumes nine messages.

```
 // Configure broadcasting subscription.
 factoryInfo.setFactoryProperty(ONSFactoryProperty:: MessageModel, ONSFactoryProperty::BROADCASTING);
```

## Sample code

```
#include "rocketmq/ONSFactory.h"
#include <iostream>
#include <thread>
#include <mutex>
using namespace ons;
std::mutex console_mtx;
class ExampleMessageListener : public MessageListener {
public:
    Action consume(Message& message, ConsumeContext& context) {
        // The system processes the message. After the processing result is accepted by the consumer, the system returns CommitMessage to the producer.
        // If a message consumption failure occurs or the consumer wants to consume the message again, the system returns ReconsumeLater to the producer. Then, the message is delivered to the consumer again after a specific period.
        std::lock_guard<std::mutex> lk(console_mtx);
        std::cout << "Received a message. Topic: " << message.getTopic() << ", MsgId: "
        << message.getMsgID() << std::endl;
        return CommitMessage;
    }
};
int main(int argc, char* argv[]) {
    std::cout << "=======Before consuming messages=======" << std::endl;
    ONSFactoryProperty factoryInfo;
    // Specify the group ID that you created in the Message Queue for Apache RocketMQ console. For the service versions that involve instances, producer IDs and consumer IDs are replaced with group IDs. This configuration ensures the compatibility with earlier versions.
    factoryInfo.setFactoryProperty(ONSFactoryProperty::ConsumerId, "GID_XXX");
    // Specify the AccessKey ID of your Alibaba Cloud account.
    factoryInfo.setFactoryProperty(ONSFactoryProperty::AccessKey, "Your Access Key");
    // Specify the AccessKey secret of your Alibaba Cloud account.
    factoryInfo.setFactoryProperty(ONSFactoryProperty::SecretKey, "Your Secret Key");
    // Specify the TCP endpoint of your Message Queue for Apache RocketMQ instance. You can view the endpoint in the Message Queue for Apache RocketMQ console.
    factoryInfo.setFactoryProperty(ONSFactoryProperty::NAMESRV_ADDR,
                                   "http://xxxxxxxxxxxxxxx.aliyuncs.com:80");
    PushConsumer *consumer = ONSFactory::getInstance()->createPushConsumer(factoryInfo);
    // Specify a topic that you created in the Message Queue for Apache RocketMQ console.
    const char* topic_1 = "topic-1";
    // Subscribe to the messages attached with the tag-1 tag in topic-1.
    const char* tag_1 = "tag-1";
    const char* topic_2 = "topic-2";
    // Subscribe to all messages in topic-2.
    const char* tag_2 = "*";
    // Use a custom listener function to process the received messages and return the processing results
```

```
using results.
    ExampleMessageListener * message_listener = new ExampleMessageListener();
    consumer->subscribe(topic_1, tag_1, message_listener);
    consumer->subscribe(topic_2, tag_2, message_listener);
    // The preparation is complete. You must invoke the startup function to start the consu
mer.
    consumer->start();
    // Keep the thread running and do not shut down the consumer.
    std::this_thread::sleep_for(std::chrono::milliseconds(60 * 1000));
    consumer->shutdown();
    delete message_listener;
    std::cout << "=======After consuming messages======" << std::endl;
    return 0;
}
```

# 1.1.6.2.3. SDK for .NET

# 1.1.6.2.3.1. .Prepare the SDK for .NET environment

Overview

Before you use SDK for . NET to access Message Queue for Apache RocketMQ and send and subscribe to messages, make sure that the following prerequisites are met:

> **Note**
>
> - You have created the topics and group IDs involved in the code in the Message Queue for Apache RocketMQ console. You can customize message tags in your application. For more information about how to create a message tag, see Create resources.
> - Applications that use Message Queue for Apache RocketMQ are deployed on Elastic Compute Service (ECS) instances.

Download SDK for .NET

Message Queue for Apache RocketMQ SDK for .NET is a managed wrapper based on Apache RocketMQ Client CPP.Message Queue for Apache RocketMQ SDK for .NET is independent of Windows .NET public library.Multithreading and parallel processing in C++ are used to ensure the efficiency and stability of Message Queue for Apache RocketMQ SDK for .NET.

## Context

If Visual Studio is used to develop .NET applications and class libraries, the default target platform is Any CPU. This means that x86 or x64 is automatically selected based on the CPU type at runtime. This capability is provided because the assembly compiled by using .NET is based on the intermediate language (IL). At runtime, the just in-time compiler (JIT) in the common language runtime (CLR) of .NET converts the IL code into the x86 or x64 machine code. The DLL generated by the C or C++ compiler is the machine code. Therefore, a target platform is selected during compilation. The C or C++ project is compiled as an x64 64-bit DLL by configuring compilation options. Therefore, the 64-bit DLL in release mode compiled by using Visual Studio 2015 is provided. The 64-bit DLL in release mode is also available to other Visual Studio versions.

> **Note** C++ DLL files require the installation package of the Virtual C++ 2015 runtime environment. If the Visual Studio 2015 runtime environment is not installed, run the vc_redist.x64.exe program provided in the SDK.

## Procedure

1. Download the SDK package.

   We recommend that both new users and existing users that are not concerned with upgrade costs download the latest SDK. Download the latest version of SDK for .NET that are used in Windows

2. Decompress the downloaded package.

   After the package is decompressed, the following directory structure appears:

   - *demo/*

     Contains examples on how to send normal messages, send messages in one-way mode, send ordered messages, consume normal messages, and consume ordered messages.

   - *lib/*

     Contains files related to the underlying C++ DLL and the installation package of the Virtual C++ 2015 runtime environment. If Visual Studio 2015 is not installed, copy and run the vc_redist.x64.exe program, as shown in the following information:

     ```
     64/
         NSClient4CPP.lib
         ONSClient4CPP.dll
         ONSClient4CPP.pdb
     vc_redist.x64.exe
     ```

   - *interface/*

     Encapsulates P/Invoke code. The code must be included in the user project code.

   - *SDK_GUIDE.pdf*

     Contains the documentation and frequently asked questions (FAQ) about how to prepare the SDK environment.

   - *changelog*

     Contains bug fixes and new features in the new releases.

.Configure SDK for .NET

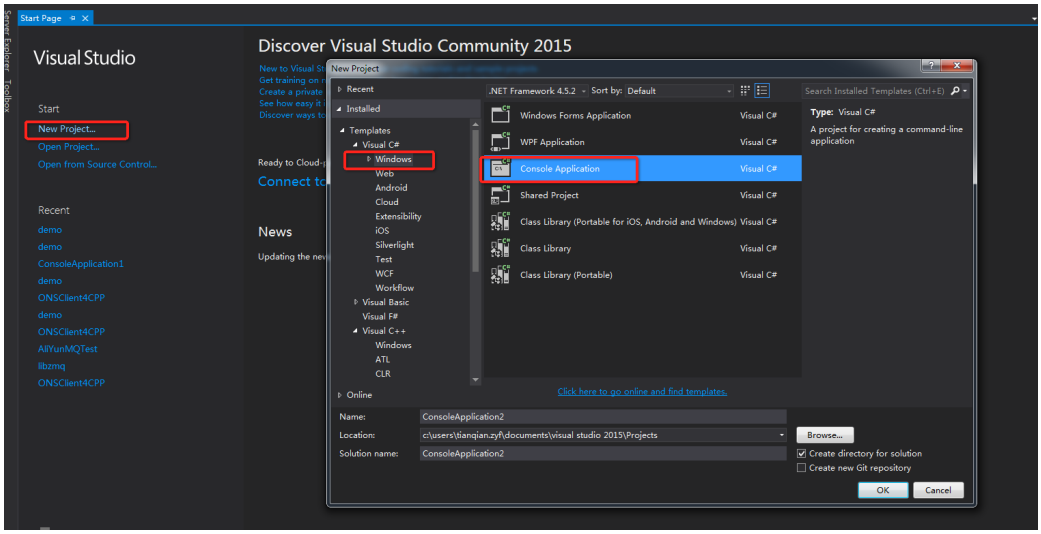
This topic shows you how to use SDK for .NET in Windows.

## Procedure

Use SDK for .NET in Visual Studio 2015NET SDK

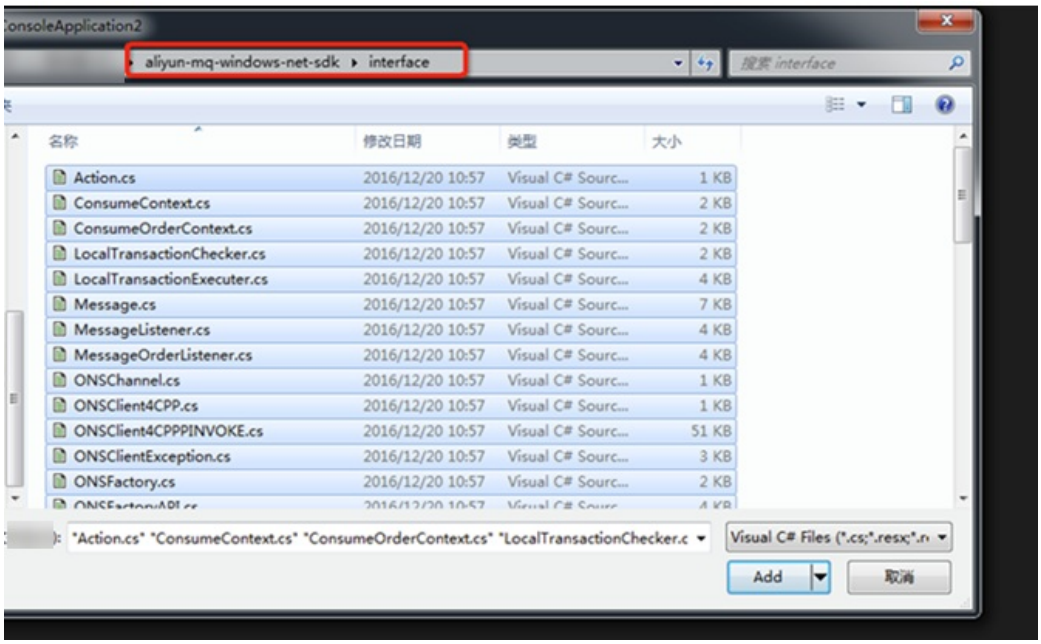
1. Use Visual Studio 2015 to create your project.

   .NET SDK-1



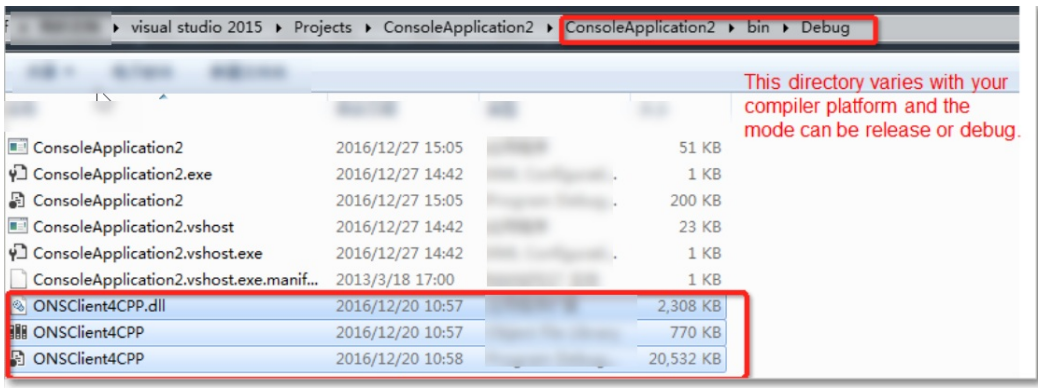

2. Right-click the project and choose **Add** > **Add Existing Item** to add all files in the **interface** directory of the downloaded SDK package.

   .NET SDK-2



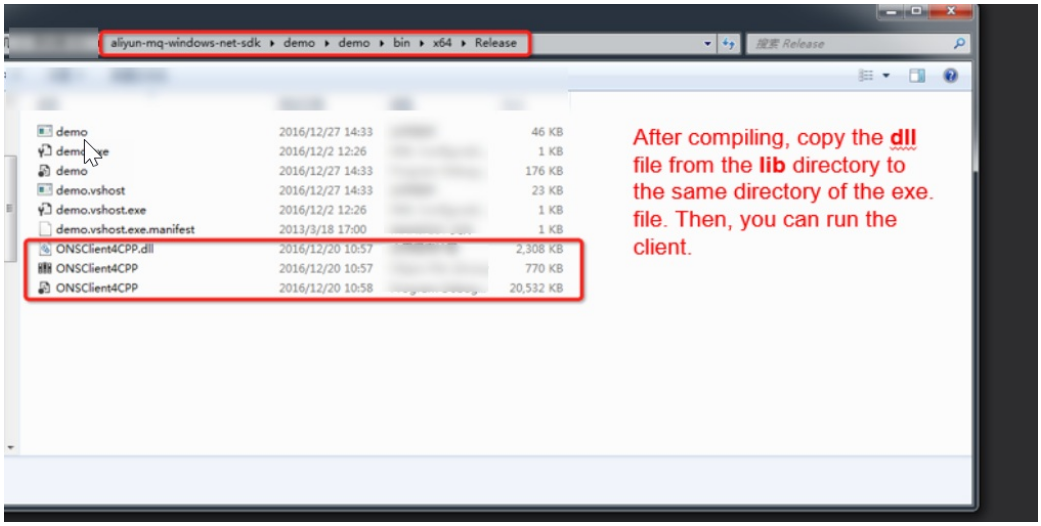

3. Right-click the project and choose **Properties** > **Configuration Manager**. Set **Active solution configuration** to **Release** and set **Active solution platform** to **x64**.
4. Write and compile the test program, save the DLL file of the SDK to the directory of the executable file or to the system directory, and then run the program.

   .NET SDK-3




**Note** The SDK provides a preconfigured demo project. You can directly open the project and compile it. When you run the project, copy the related DLL file to the directory of the executable file, as shown in the following figure.

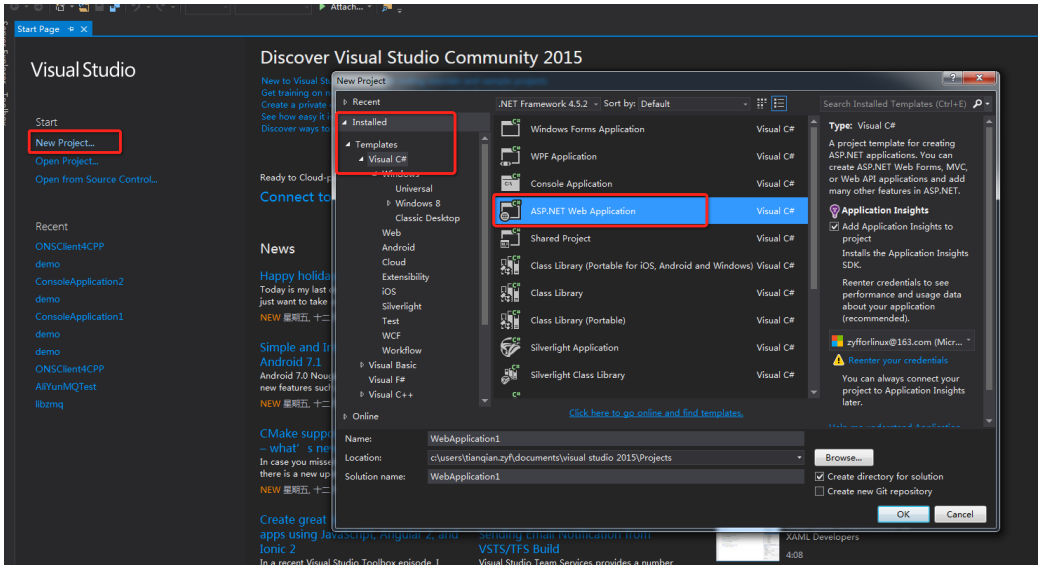
.NET SDK-4




**Configure ASP.NET in Visual Studio 2015 to use Message Queue for Apache RocketMQ SDK**

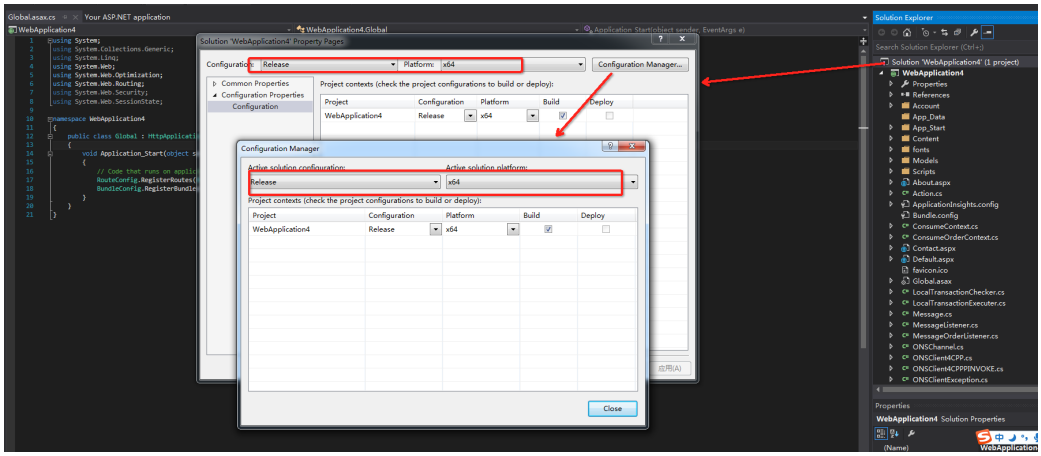
5. Create a **Web Forms** project for ASP.NET by using Visual Studio 2015.

.NET SDK-5




6. Right-click the project and choose **Properties** > **Configuration Manager**. Set **Active solution configuration** to **Release** and set **Active solution platform** to **x64**.

   .NET SDK-6




7. Right-click the project and choose **Add** > **Add Existing Item** to add all files in the interface directory of the downloaded SDK package.

   For more information about how to configure a common .NET project, see Step 2.

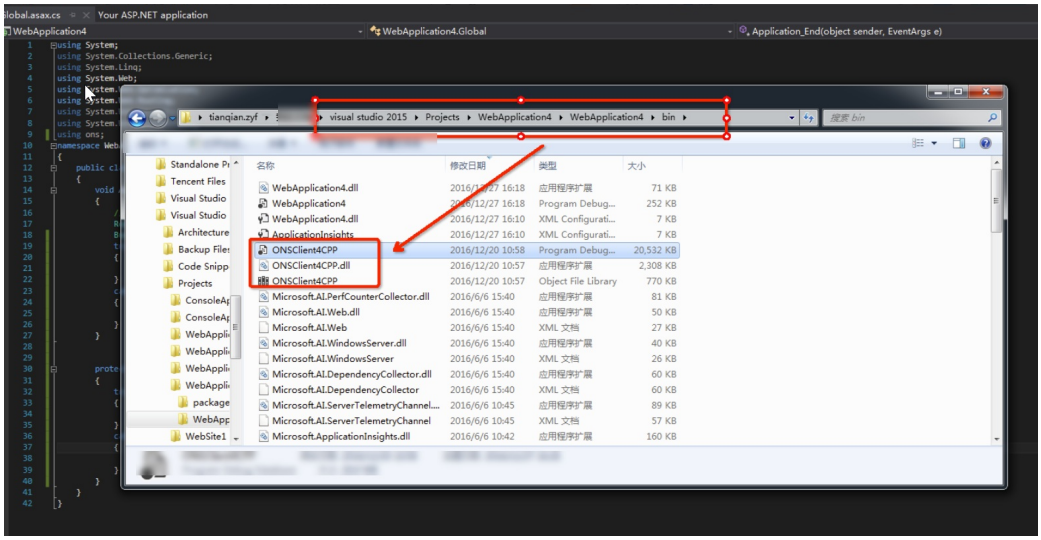
8. Add the code for starting and stopping the SDK to the **Global.asax.cs** file.

> **Note** We recommend that you encapsulate the SDK code as a singleton class so that the code cannot be recycled by the garbage collector due to scope problems. The example directory of the SDK contains the Example.cs file for implementing a simple singleton class. To use Example.cs, you must include it in your own project.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Optimization;
using System.Web.Routing;
using System.Web.Security;
using System.Web.SessionState;
using ons;    // The namespace where the SDK is located.
using test;    // The namespace where the class with the roughly encapsulated SDK is lo
cated. See the Example.cs file in the example directory of the SDK.
namespace WebApplication4
{
    public class Global : HttpApplication
    {
        void Application_Start(object sender, EventArgs e)
        {
            // Code that runs on application startup
            RouteConfig.RegisterRoutes(RouteTable.Routes);
            BundleConfig.RegisterBundles(BundleTable.Bundles);
            try
            {
                // The code for starting the SDK. The following code is the code after
the SDK is roughly encapsulated.
                OnscSharp.CreateProducer();
                OnscSharp.StartProducer();
            }
            catch (Exception ex)
            {
                    // Specify the logic for handling errors.
            }
        }
        protected void Application_End(object sender, EventArgs e)
        {
            try
            {
               // The code for stopping the SDK.
              OnscSharp.ShutdownProducer();
            }
            catch (Exception ex)
            {
              // Specify the logic for handling errors.
            }
        }
    }
}
```

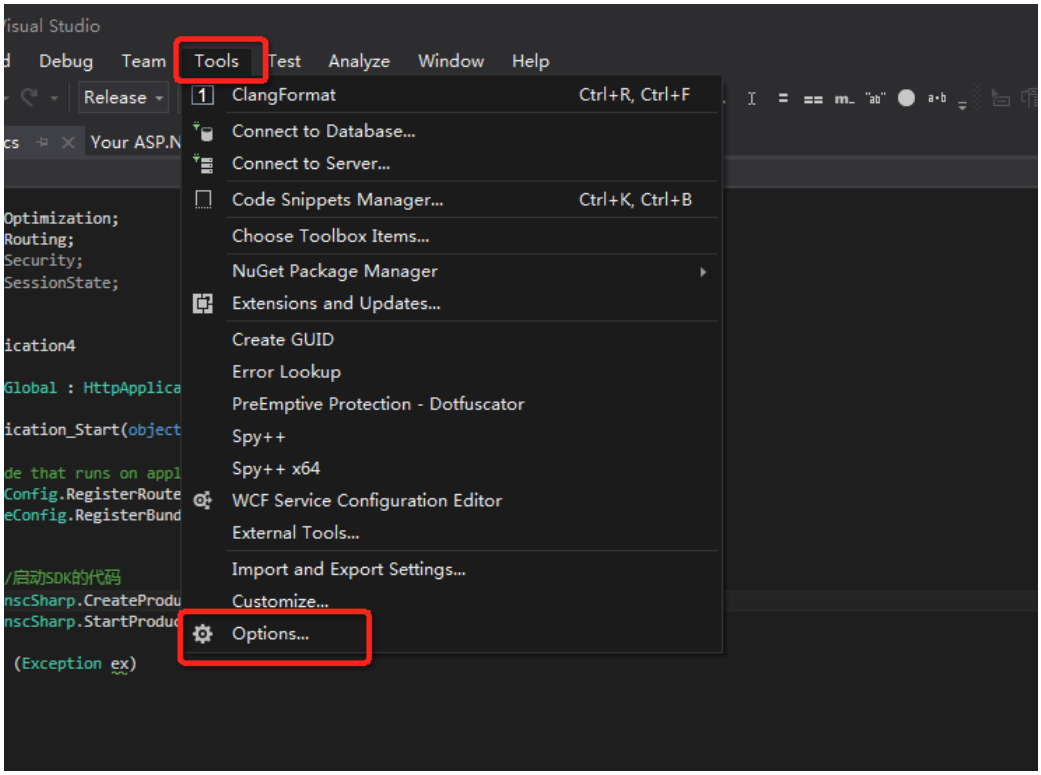
9. Write and compile the test program.
10. Save the DLL file of the SDK to the directory of the executable file or to the system directory and run the program.

    .NET SDK-7

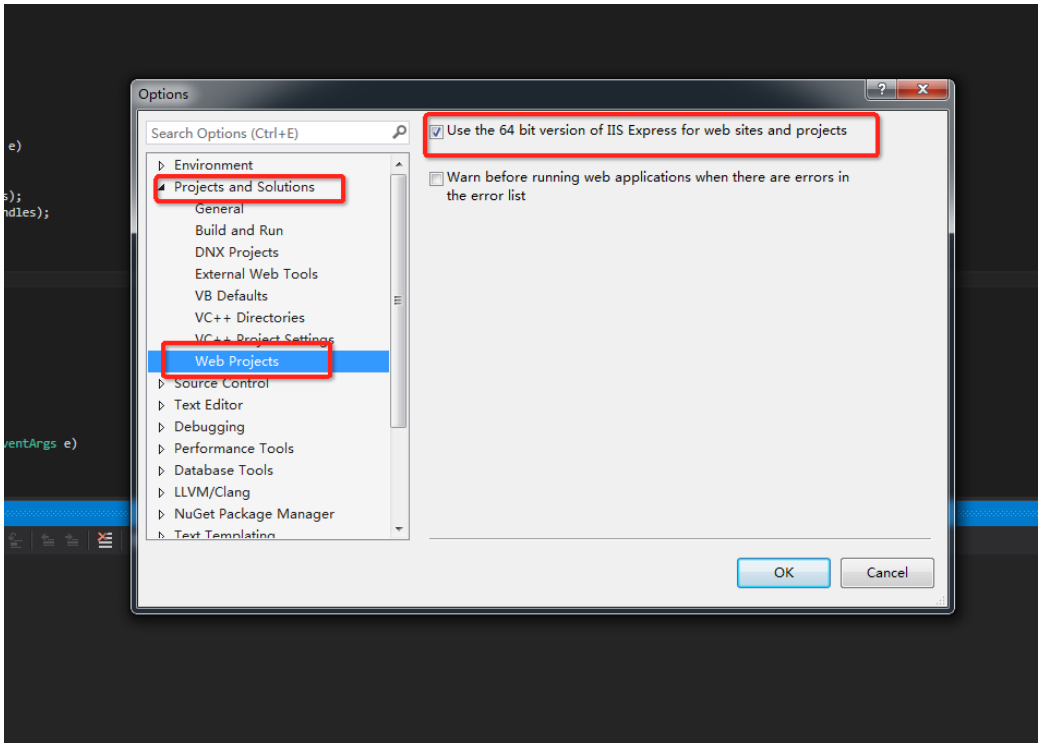


11. Choose **Tools** > **Options** > **Projects and Solutions** > **Web Projects**. Then, select the **Use the 64 bit version of IIS Express for websites and projects** check box.

.NET SDK-8




.NET SDK-9




# 1.1.6.2.3.2. Send and subscribe to normal messages

This topic provides the sample code for sending and subscribing to normal messages.

## Send normal messages

The following sample code provides an example on how to send normal messages: Set related parameters based on the instructions.

```
using System;
using ons;
public class ProducerExampleForEx
{
    public ProducerExampleForEx()
    {
    }
    static void Main(string[] args) {
        // Configure your account based on the resources that you created in the console.
        ONSFactoryProperty factoryInfo = new ONSFactoryProperty();
        // The AccessKey ID that you created in the Apsara Uni-manager Management Console f
or identity authentication.
        factoryInfo.setFactoryProperty(ONSFactoryProperty.AccessKey, "Your access key");
        // The AccessKey secret that you created in the Apsara Uni-manager Management Conso
le for identity authentication.
        factoryInfo.setFactoryProperty(ONSFactoryProperty.SecretKey, "Your access secret");
        // The group ID that you created in the Message Queue for Apache RocketMQ console.
        factoryInfo.setFactoryProperty(ONSFactoryProperty.ProducerId, "GID_example");
        // The topic that you created in the Message Queue for Apache RocketMQ console.
        factoryInfo.setFactoryProperty(ONSFactoryProperty.PublishTopics, "T_example_topic_n
ame");
        // The TCP endpoint. To obtain the endpoint, log on to the Message Queue for Apache
```

```
RocketMQ console. In the left-side navigation pane, click Instance Details. On the Instance
Details page, select your instance. On the Instance Information tab, view the endpoint in t
he Obtain Endpoint Information section.
        factoryInfo.setFactoryProperty(ONSFactoryProperty.NAMESRV_ADDR, "NameSrv_Addr");
        // Specify the log path.
        factoryInfo.setFactoryProperty(ONSFactoryProperty.LogPath, "C://log");
        // Create a producer instance.
        // Note: Producer instances are thread-secure and can be used to send messages of d
ifferent topics. Each of your threads
        // needs only one producer instance.
        Producer producer = ONSFactory.getInstance().createProducer(factoryInfo);
        // Start the producer instance.
        producer.start();
        // Create a message object.
        Message msg = new Message(factoryInfo.getPublishTopics(), "tagA", "Example message
body");
        msg.setKey(Guid.NewGuid().ToString());
        for (int i = 0; i < 32; i++) {
            try
            {
                SendResultONS sendResult = producer.send(msg);
                Console.WriteLine("send success {0}", sendResult.getMessageId());
            }
            catch (Exception ex)
            {
                Console.WriteLine("send failure{0}", ex.ToString());
            }
        }
        // Shut down the producer instance when your thread is about to exit.
        producer.shutdown();
    }
}
```

### Subscribe to normal messages

For more information about how to subscribe to normal messages and about relevant sample code, see Subscribe to messages.

# 1.1.6.2.3.3. Send and subscribe to ordered messages

This topic describes how to send and subscribe to ordered messages and provides sample code.

### Send ordered messages

The following sample code provides an example on how to send ordered messages:

```
using System;
using ons;
public class OrderProducerExampleForEx
{
    public OrderProducerExampleForEx()
    {
    }
    static void Main(string[] args) {
```

```
        // Configure your account based on the resources that you created in the Apsara Uni
-manager Management Console.
        ONSFactoryProperty factoryInfo = new ONSFactoryProperty();
        // The AccessKey ID that you created in the Apsara Uni-manager Management Console f
or identity authentication.
        factoryInfo.setFactoryProperty(ONSFactoryProperty.AccessKey, "Your access key");
        // The AccessKey secret that you created in the Apsara Uni-manager Management Conso
le for identity authentication.
        factoryInfo.setFactoryProperty(ONSFactoryProperty.SecretKey, "Your access secret");
        // The group ID that you created in the Message Queue for Apache RocketMQ console.
        factoryInfo.setFactoryProperty(ONSFactoryProperty.ProducerId, "GID_example");
        // The topic that you created in the Message Queue for Apache RocketMQ console.
        factoryInfo.setFactoryProperty(ONSFactoryProperty.PublishTopics, "T_example_topic_n
ame");
        // The TCP endpoint. To obtain the endpoint, log on to the Message Queue for Apache
RocketMQ console. In the left-side navigation pane, click Instance Details. On the Instance
Details page, select your instance. On the Instance Information tab, view the endpoint in t
he Obtain Endpoint Information section.
        factoryInfo.setFactoryProperty(ONSFactoryProperty.NAMESRV_ADDR, "NameSrv_Addr");
        // Specify the log path.
        factoryInfo.setFactoryProperty(ONSFactoryProperty.LogPath, "C://log");
        // Create a producer instance.
        // Note: Producer instances are thread-safe and can be used to send messages of dif
ferent topics. Each thread
        // requires only one producer instance.
        OrderProducer producer = ONSFactory.getInstance().createOrderProducer(factoryInfo);
        // Start the producer instance.
        producer.start();
        // Create a message.
        Message msg = new Message(factoryInfo.getPublishTopics(), "tagA", "Example message
body");
        string shardingKey = "App-Test";
        for (int i = 0; i < 32; i++) {
            try
            {
                SendResultONS sendResult = producer.send(msg, shardingKey);
                Console.WriteLine("send success {0}", sendResult.getMessageId());
            }
            catch (Exception ex)
            {
                Console.WriteLine("send failure{0}", ex.ToString());
            }
        }
        // Disable the producer instance when your thread is about to exit.
        producer.shutdown();
    }
}
```

## Subscribe to ordered messages

The following sample code provides an example on how to subscribe to ordered messages:

```
using System;
using System.Text;
```

```
using System.Threading;
using ons;
namespace demo
{
    public class MyMsgOrderListener : MessageOrderListener
    {
        public MyMsgOrderListener()
        {
        }
        ~MyMsgOrderListener()
        {
        }
        public override ons.OrderAction consume(Message value, ConsumeOrderContext context)
        {
            Byte[] text = Encoding.Default.GetBytes(value.getBody());
            Console.WriteLine(Encoding.UTF8.GetString(text));
            return ons.OrderAction.Success;
        }
    }
    class OrderConsumerExampleForEx
    {
        static void Main(string[] args)
        {
            // Configure your account based on the resources that you created in the Apsara Uni-manager Management Console.
            ONSFactoryProperty factoryInfo = new ONSFactoryProperty();
            // The AccessKey ID that you created in the Apsara Uni-manager Management Console for identity authentication.
            factoryInfo.setFactoryProperty(ONSFactoryProperty.AccessKey, "Your access key");
            // The AccessKey secret that you created in the Apsara Uni-manager Management Console for identity authentication.
            factoryInfo.setFactoryProperty(ONSFactoryProperty.SecretKey, "Your access secret");
            // The group ID that you created in the Message Queue for Apache RocketMQ console.
            factoryInfo.setFactoryProperty(ONSFactoryProperty.ConsumerId, "GID_example");
            // The topic that you created in the Message Queue for Apache RocketMQ console.
            factoryInfo.setFactoryProperty(ONSFactoryProperty.PublishTopics, "T_example_topic_name");
            // The TCP endpoint. To obtain the endpoint, log on to the Message Queue for Apache RocketMQ console. In the left-side navigation pane, click Instance Details. On the Instance Details page, select your instance. On the Instance Information tab, view the endpoint in the Obtain Endpoint Information section.
            factoryInfo.setFactoryProperty(ONSFactoryProperty.NAMESRV_ADDR, "NameSrv_Addr");
            // Specify the log path.
            factoryInfo.setFactoryProperty(ONSFactoryProperty.LogPath, "C://log");
            // Create a consumer instance.
            OrderConsumer consumer = ONSFactory.getInstance().createOrderConsumer(factoryInfo);
            // Subscribe to topics.
            consumer.subscribe(factoryInfo.getPublishTopics(), "*",new MyMsgOrderListener());
```

```
            // Start the consumer instance.
            consumer.start();
            // Put the main thread to sleep for a period of time.
            Thread.Sleep(30000);
            // Disable the consumer instance when the instance is no longer used.
            consumer.shutdown();
        }
    }
}
```

# 1.1.6.2.3.4. Send and subscribe to scheduled messages

Scheduled messages are consumed by consumers after a specified period of time. Such messages are used in scenarios where a time window between message production and consumption is required or tasks need to be triggered at a scheduled time. Scheduled messages are used in a similar way to delay queues.

## Send scheduled messages

The following sample code provides an example on how to send scheduled messages:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Runtime.InteropServices;
using ons;
namespace ons
{
    class onscsharp
    {
        static void Main(string[] args)
        {
            // Set the parameters that are required to create a producer. These parameters
ensure that you can use the producer.
            ONSFactoryProperty factoryInfo = new ONSFactoryProperty();
            factoryInfo.setFactoryProperty(factoryInfo.ProducerId, "XXX ");// The group ID
that you created in the Message Queue for Apache RocketMQ console.
            factoryInfo.setFactoryProperty(factoryInfo.NAMESRV_ADDR, "XXX"); // The TCP end
point. To obtain the endpoint, log on to the Message Queue for Apache RocketMQ console. In
the left-side navigation pane, click Instance Details. On the Instance Details page, select
your instance. On the Instance Information tab, view the endpoint in the Obtain Endpoint In
formation section.
            factoryInfo.setFactoryProperty(factoryInfo.PublishTopics, "XXX");// The topic t
hat you created in the Message Queue for Apache RocketMQ console.
            factoryInfo.setFactoryProperty(factoryInfo.MsgContent, "XXX");// The message co
ntent.
            factoryInfo.setFactoryProperty(factoryInfo.AccessKey, "XXX");// The AccessKey I
D that you created in the Apsara Uni-manager Management Console for identity authentication
.
            factoryInfo.setFactoryProperty(factoryInfo.SecretKey,"XXX");// The AccessKey se
cret that you created in the Apsara Uni-manager Management Console for identity authenticat
ion.
```

```
            // Create a producer.
            ONSFactory onsfactory = new ONSFactory();
            Producer pProducer = onsfactory.getInstance().createProducer(factoryInfo);
            // Before you use the producer to send a message, call the start() method once
to start the producer.
            pProducer.start();
            Message msg = new Message(
                //Message Topic
                factoryInfo.getPublishTopics(),
                //Message Tag
                "TagA",
                //Message Body
                factoryInfo.getMessageContent()
            );
            // The key of the message. The key is the business-specific attribute of the me
ssage and must be globally unique whenever possible.
            // A unique key helps you query and resend a message in the Message Queue for A
pache RocketMQ console if the message fails to be received.
            // Note: Messages can be sent and received even if you do not specify the messa
ge key.
            msg.setKey("ORDERID_100");
            // The period of time after which the Message Queue for Apache RocketMQ broker
delivers the message to the consumer. Unit: milliseconds. The message can be consumed only
after the specified period of time. In this example, the message can be consumed 3 seconds
later.
            long deliverTime = Current system time (ms) + 3000;
            msg.setStartDeliverTime(deliverTime);
            // Send the message. If no error occurs, the message is sent.
            try
            {
                SendResultONS sendResult = pProducer.send(msg);
            }
            catch(ONSClientException e)
            {
                // Specify the logic for handling failures.
            }
            // Before you exit the application, shut down the producer object. Otherwise, m
emory leaks may occur.
            pProducer.shutdown();
        }
    }
}
```

### Subscribe to scheduled messages

For more information about how to subscribe to scheduled messages and about relevant sample code, see Subscribe to messages.

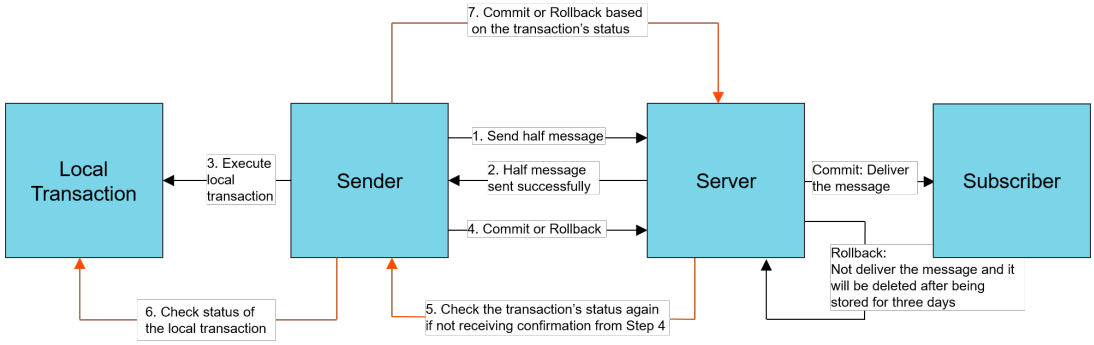
# 1.1.6.2.3.5. Send and subscribe to transactional messages

This topic describes the interaction process and the back-check mechanism of transactional messages. This topic also shows you how to send and subscribe to transactional messages, and provides sample code.

## Interaction process

Transactional message interaction shows the interaction process of transactional messages in Message Queue for Apache RocketMQ.

Interaction process of transactional messages




## Send transactional messages

Perform the following steps to send a transactional message:

1. Send a half message and execute a local transaction. The following code provides examples on how to send and subscribe to transactional messages:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Runtime.InteropServices;
using ons;
namespace ons
{
public class MyLocalTransactionExecuter : LocalTransactionExecuter
{
    public MyLocalTransactionExecuter()
    {
    }
    ~MyLocalTransactionExecuter()
    {
    }
    public override TransactionStatus execute(Message value)
    {
            Console.WriteLine("execute topic: {0}, tag:{1}, key:{2}, msgId:{3},msgbody
:{4}, userProperty:{5}",
            value.getTopic(), value.getTag(), value.getKey(), value.getMsgID(), value.
getBody(), value.getUserProperty("VincentNoUser"));
            // The ID of the message. Two messages may have the same message body but
cannot have the same ID. The current message ID cannot be queried in the console.
            string msgId = value.getMsgID();
            // Calculate the message body by using CRC32 or other algorithms, such as
MD5.
```

```
            // The message ID and CRC32 ID are used to prevent duplicate messages.
            // To prevent duplicate messages, calculate the message body by using the
CRC32 or MD5 algorithm.
            TransactionStatus transactionStatus = TransactionStatus.Unknow;
            try {
                boolean isCommit = Execution result of the local transaction;
                if (isCommit) {
                    // Commit the message if the local transaction succeeds.
                    transactionStatus = TransactionStatus.CommitTransaction;
                } else {
                    // Roll back the message if the local transaction fails.
                    transactionStatus = TransactionStatus.RollbackTransaction;
                }
            } catch (Exception e) {
                //exception handle
            }
            return transactionStatus;
    }
 }
 class onscsharp
 {
     static void Main(string[] args)
     {
         ONSFactoryProperty factoryInfo = new ONSFactoryProperty();
         factoryInfo.setFactoryProperty(factoryInfo.NAMESRV_ADDR, "XXX");//The TCP endp
oint. To obtain the endpoint, log on to the Message Queue for Apache RocketMQ console.
In the left-side navigation pane, click Instance Details. On the Instance Details page,
select your instance. On the Instance Information tab, view the endpoint in the Obtain
Endpoint Information section.
         factoryInfo.setFactoryProperty(factoryInfo.ProducerId, "");// The group ID tha
t you created in the Message Queue for Apache RocketMQ console.
         factoryInfo.setFactoryProperty(factoryInfo.PublishTopics, "");// The topic tha
t you created in the Message Queue for Apache RocketMQ console.
         factoryInfo.setFactoryProperty(factoryInfo.MsgContent, "");//message body
         factoryInfo.setFactoryProperty(factoryInfo.AccessKey, "");// The AccessKey ID
that you created in the Apsara Uni-manager Management Console for identity authenticati
on.
         factoryInfo.setFactoryProperty(factoryInfo.SecretKey, "");// The AccessKey sec
ret that you created in the Apsara Uni-manager Management Console for identity authenti
cation.
         //create transaction producer
         ONSFactory onsfactory = new ONSFactory();
         LocalTransactionChecker myChecker = new MyLocalTransactionChecker();
         TransactionProducer pProducer = onsfactory.getInstance().createTransactionProd
ucer(factoryInfo,ref myChecker);
         // Before you use the producer to send a message, call the start() method once
to start the producer. After the producer is started, messages can be concurrently sent
in multiple threads.
         pProducer.start();
             Message msg = new Message(
             //Message Topic
             factoryInfo.getPublishTopics(),
             //Message Tag
             "TagA",
             //Message Body
```

```
                //Message Body
                factoryInfo.getMessageContent()
            );
            // The key of the message. The key is the business-specific attribute of the m
essage and must be globally unique whenever possible.
            // A unique key helps you query and resend a message in the Message Queue for
Apache RocketMQ console if the message fails to be received.
            // Note: Messages can be sent and received even if you do not specify the mess
age key.
            msg.setKey("ORDERID_100");
            // Send the message. If no error occurs, the message is sent.
            try
            {
                LocalTransactionExecuter myExecuter = new MyLocalTransactionExecuter();
                SendResultONS sendResult = pProducer.send(msg, ref myExecuter);
            }
            catch(ONSClientException e)
            {
                Console.WriteLine("\nexception of sendmsg:{0}",e.what() );
            }
            // Before you exit the application, shut down the producer object. Otherwise,
memory leaks may occur.
            // The producer cannot be started again after the producer object is shut down
.
            pProducer.shutdown();
        }
    }
    }
```

2. Commit the status of the transactional message.

   After the local transaction is executed, the Message Queue for Apache RocketMQ broker must be notified of the transaction status of the current message no matter whether the execution is successful or fails. The Message Queue for Apache RocketMQ broker can be notified in one of the following ways:

   - Commit the status after the local transaction is executed.
   - Wait until the Message Queue for Apache RocketMQ broker sends a request to check the transaction status of the message.

   A transaction can be in one of the following states:

   - TransactionStatus.CommitTransaction: The transaction is committed. The consumer can consume the message.
   - TransactionStatus.RollbackTransaction: The transaction is rolled back. The message is discarded and cannot be consumed.
   - TransactionStatus.Unknow: The status of the transaction is unknown. The Message Queue for Apache RocketMQ broker is expected to send a request again to the producer to query the status of the local transaction that corresponds to the message.

```
    public class MyLocalTransactionChecker : LocalTransactionChecker
    {
        public MyLocalTransactionChecker()
        {
        }
        ~MyLocalTransactionChecker()
        {
        }
        public override TransactionStatus check(Message value)
        {
                Console.WriteLine("check topic: {0}, tag:{1}, key:{2}, msgId:{3},msgbod
y:{4}, userProperty:{5}",
                value.getTopic(), value.getTag(), value.getKey(), value.getMsgID(), val
ue.getBody(), value.getUserProperty("VincentNoUser"));
                // The ID of the message. Two messages may have the same message body b
ut cannot have the same ID. The current message ID cannot be queried in the console.
                string msgId = value.getMsgID();
                // Calculate the message body by using CRC32 or other algorithms, such
as MD5.
                // The message ID and CRC32 ID are used to prevent duplicate messages.
                // You do not need to specify the message ID or CRC32 ID if your busine
ss itself achieves idempotence. Otherwise, specify the message ID or CRC32 ID to ensure
idempotence.
                // To prevent duplicate messages, calculate the message body by using t
he CRC32 or MD5 algorithm.
                TransactionStatus transactionStatus = TransactionStatus.Unknow;
                try {
                    boolean isCommit = Execution result of the local transaction;
                    if (isCommit) {
                        // Commit the message if the local transaction succeeds.
                        transactionStatus = TransactionStatus.CommitTransaction;
                    } else {
                        // Roll back the message if the local transaction fails.
                        transactionStatus = TransactionStatus.RollbackTransaction;
                    }
                } catch (Exception e) {
                    //exception handle
                }
                return transactionStatus;
        }
        }
```

### Back-check mechanism for transaction status

- Why must the back-check mechanism for transaction status be implemented when transactional messages are sent?

  If the half message is sent in Step 1 but TransactionStatus.Unknow is returned for the local transaction, or no status is committed for the local transaction because the application exits, the status of the half message is unknown to the Message Queue for Apache RocketMQ broker. Therefore, the Message Queue for Apache RocketMQ broker periodically requests the producer to check and report the status of the half message.

- What does the business logic do when the check method is called back?

The check method for transactional messages in Message Queue for Apache RocketMQ must contain the logic of transaction consistency check. After a transactional message is sent, Message Queue for Apache RocketMQ must call the LocalTransactionChecker method to respond to the request of the Message Queue for Apache RocketMQ broker for the status of the local transaction. Therefore, the check method for transactional messages must contain the following check items:

i. Check the status of the local transaction that corresponds to the half message. The status is committed or rollback.

ii. Commit the status of the local transaction that corresponds to the half message to the Message Queue for Apache RocketMQ broker.

- How do different states of the local transaction affect the half message?
  - TransactionStatus.CommitTransaction: The transaction is committed. The consumer can consume the message.
  - TransactionStatus.RollbackTransaction: The transaction is rolled back. The message is discarded and cannot be consumed.
  - TransactionStatus.Unknow: The status of the transaction is unknown. The Message Queue for Apache RocketMQ broker is expected to send a request again to the producer to query the status of the local transaction that corresponds to the message.

For more information about the code, see the implementation of MyLocalTransactionChecker.

## Subscribe to transactional messages

For more information about how to subscribe to transactional messages and about relevant sample code, see Subscribe to messages.

# 1.1.6.2.3.6. Subscribe to messages

This topic describes how to use Message Queue for Apache RocketMQ SDK for .NET to subscribe to messages.

> **Note**
> - The subscriptions of all consumer instances identified by the same group ID must be consistent. For more information, see Subscription consistency.

## Subscription modes

Message Queue for Apache RocketMQ supports the following message subscription modes:

- **Clustering subscription**: In this mode, all the consumer instances identified by the same group ID evenly share messages. Assume that a topic contains nine messages and a group ID identifies three consumer instances. In clustering consumption mode, each instance consumes three messages.

```
 // Configure clustering subscription, which is the default mode.
 factoryInfo.setFactoryProperty(ONSFactoryProperty.MessageModel, ONSFactoryProperty.CLUST
ERING);
```

- **Broadcasting subscription**: In this mode, each consumer instance identified by a group ID consumes a message once. Assume that a topic contains nine messages and a group ID identifies three consumer instances. In broadcasting consumption mode, each instance consumes nine messages.

```
 // Configure broadcasting subscription.
 factoryInfo.setFactoryProperty(ONSFactoryProperty.MessageModel, ONSFactoryProperty.BROADCASTING);
```

## Sample code

```
using System;
using System.Threading;
using System.Text;
using ons;
// The callback function to be executed when a message is pulled from the Message Queue for Apache RocketMQ broker.
public class MyMsgListener : MessageListener
{
    public MyMsgListener()
    {
    }
    ~MyMsgListener()
    {
    }
    public override ons.Action consume(Message value, ConsumeContext context)
    {
        Byte[] text = Encoding.Default.GetBytes(value.getBody());
        Console.WriteLine(Encoding.UTF8.GetString(text));
        return ons.Action.CommitMessage;
    }
}
public class ConsumerExampleForEx
{
    public ConsumerExampleForEx()
    {
    }
    static void Main(string[] args) {
        // Configure your account based on the resources that you created in the console.
        ONSFactoryProperty factoryInfo = new ONSFactoryProperty();
        // The AccessKey ID that you created in the Apsara Uni-manager Management Console for identity authentication.
        factoryInfo.setFactoryProperty(ONSFactoryProperty.AccessKey, "Your access key");
        // The AccessKey secret that you created in the Apsara Uni-manager Management Console for identity authentication.
        factoryInfo.setFactoryProperty(ONSFactoryProperty.SecretKey, "Your access secret");
        // The group ID that you created in the Message Queue for Apache RocketMQ console.
        factoryInfo.setFactoryProperty(ONSFactoryProperty.ConsumerId, "GID_example");
        // The topic that you created in the Message Queue for Apache RocketMQ console.
        factoryInfo.setFactoryProperty(ONSFactoryProperty.PublishTopics, "T_example_topic_name");
        // The TCP endpoint. To obtain the endpoint, log on to the Message Queue for Apache RocketMQ console. In the left-side navigation pane, click Instance Details. On the Instance Details page, select your instance. On the Instance Information tab, view the endpoint in the Obtain Endpoint Information section.
        factoryInfo.setFactoryProperty(ONSFactoryProperty.NAMESRV_ADDR, "NameSrv_Addr");
        // Specify the log path.
        factoryInfo.setFactoryProperty(ONSFactoryProperty.LogPath, "C://log");
        // The clustering consumption mode
```

```
        // The clustering consumption mode.
        // factoryInfo.setFactoryProperty(ONSFactoryProperty:: MessageModel, ONSFactoryProperty.CLUSTERING);
        // The broadcasting consumption mode.
        // factoryInfo.setFactoryProperty(ONSFactoryProperty:: MessageModel, ONSFactoryProperty.BROADCASTING);
        // Create a consumer instance.
        PushConsumer consumer = ONSFactory.getInstance().createPushConsumer(factoryInfo);
        // Subscribe to topics.
        consumer.subscribe(factoryInfo.getPublishTopics(), "*", new MyMsgListener());
        // Start the consumer instance.
        consumer.start();
        // This value is an example in the demo. In your production environment, you must set a proper value to make sure that the process does not unexpectedly exit.
        Thread.Sleep(300000);
        // Shut down the consumer instance when the process is about to exit.
        consumer.shutdown();
    }
}
```

# 1.1.6.3. Client parameters

This topic describes the parameters that are configured for Message Queue for Apache RocketMQ clients.

Client parameters

| Parameter | Client | Default value | Recommended value | Description | Client version |
| --- | --- | --- | --- | --- | --- |
| AccessKey | Producer or consumer | Configured by the user | Configured by the user | The AccessKey ID that is used to authenticate the user. | >=1.2.7.Final |
| SecretKey | Producer or consumer | Configured by the user | Configured by the user | The AccessKey secret that is used to authenticate the user. | >=1.2.7.Final |
| NAMESRV_ADDR | Producer or consumer | Generated after deployment | Generated after deployment | The endpoint that is used to connect to Message Queue for Apache RocketMQ. | >=1.2.7.Final |

| Parameter | Client | Default value | Recommended value | Description | Client version |
| --- | --- | --- | --- | --- | --- |
| MsgTraceSwitch | Producer or consumer | true | true | Specifies whether to enable the message tracing feature of Message Queue for Apache RocketMQ. | >=1.7.0.Final |
| GROUP_ID | Producer or consumer | Created in the console | Created in the console | The ID of the group to which the producer or consumer client belongs. Group IDs are compatible with producer IDs (PIDs) or consumer IDs (CIDs) in earlier versions. | >=1.7.8.Final |
| ProducerId | Producer | Created in the console | Created in the console | The ID of the group to which the producer client belongs. This parameter takes effect only on transactional messages. If a producer client fails, the Message Queue for Apache RocketMQ broker initiates requests to check the status of transactional messages on other producer clients in the same group. | >=1.2.7.Final |

| Parameter | Client | Default value | Recommended value | Description | Client version |
| --- | --- | --- | --- | --- | --- |
| SendMsgTimeoutMillis | Producer | 5000 | Default | The timeout period for sending a message. If the Message Queue for Apache RocketMQ broker does not return an acknowledgment to the producer client within the specified period of time, the producer client determines that the message failed to send. | >=1.2.7.Final |
| ConsumerId | Consumer | Created in the console | Created in the console | The ID of the group to which the consumer client belongs. | >=1.2.7.Final |

| Parameter | Client | Default value | Recommended value | Description | Client version |
| --- | --- | --- | --- | --- | --- |
| MessageModel | Consumer | CLUSTERING | Default | The consumption mode. Valid values: CLUSTERING and BROADCASTING. CLUSTERING specifies that each subscribed message is received only by one consumer client. BROADCASTING specifies that each subscribed message is received by all consumer clients. | >=1.2.7.Final |
| ConsumeThreadNums | Consumer | Dynamically adjusted from 20 to 64 | Adjusted based on business requirements | The number of consumer threads. In most cases, this parameter is set to a larger value if a longer time is required to consume a single message. | >=1.2.7.Final |
| MaxReconsumeTimes | Consumer | 16 | Default | The maximum number of delivery retries that can be performed when a message fails to be consumed. | >=1.2.7.Final |

| Parameter | Client | Default value | Recommended value | Description | Client version |
| --- | --- | --- | --- | --- | --- |
| ConsumeTimeout | Consumer | 15 | Default | The timeout period for consumption of each message. If the time to consume a message exceeds the specified timeout period, the message fails to be consumed and is redelivered after a retry interval. Configure an appropriate value for each type of application. Unit: minute. | >=1.2.7.Final |

| Parameter | Client | Default value | Recommended value | Description | Client version |
| --- | --- | --- | --- | --- | --- |
| PostSubscriptionWhenPull | Consumer | false | Adjusted based on the consumption mode | Specifies whether to carry the latest subscription together with each request. If MessageModel is set to BROADCASTING, this parameter must be set to true to prevent messages from failing to be received due to subscription inconsistency. If MessageModel is set to CLUSTERING, this parameter must be set to false because subscription consistency is required for clustering consumption. | >=1.2.7.Final |

| Parameter | Client | Default value | Recommended value | Description | Client version |
| --- | --- | --- | --- | --- | --- |
| ConsumeMessageBatchMaxSize | Consumer | 1 | Adjusted based on business requirements | The maximum number of messages that can be consumed in each batch. The actual number of messages that are consumed in a batch can be smaller than the value of this parameter. The value must be an integer from 1 to 32. The default value is 1. | >=1.6.0.Final |
| MaxCachedMessageAmount | Consumer | 5000 | Adjusted based on the memory of consumer clients | The maximum number of messages that a consumer client can cache. A large value can cause an out of memory (OOM) issue on the client. The value must be an integer from 100 to 50000. The default value is 5000. | >=1.7.0.Final |

| Parameter | Client | Default value | Recommended value | Description | Client version |
| --- | --- | --- | --- | --- | --- |
| MaxCachedMessageSizeInMiB | Consumer | 512 | Adjusted based on the memory of consumer clients | The maximum size of messages that a consumer client can cache. A large value can cause an out of memory (OOM) issue on the client. The value ranges from 16 to 2048. The default value is 512. Unit: MB. | >=1.7.0.Final |

# 1.1.6.4. Client error codes

This topic describes error codes related to sending and subscribing to messages and their references.

## Error codes related to sending and subscribing to messages

| HTTP status code | Status flag | Description | Cause and recommended solution | Broker version |
| --- | --- | --- | --- | --- |
| 13 | MESSAGE_ILLEGAL | This error is returned when the message verification fails. | Check whether the message body is empty.<br>Check whether the length of the message property exceeds 32,767 bytes.<br>Check whether the total size of the message exceeds 4 MB. | >=4.0.1 |

| HTTP status code | Status flag | Description | Cause and recommended solution | Broker version |
| --- | --- | --- | --- | --- |
| 17 | TOPIC_NOT_EXIST | This error is returned when the specified message topic does not exist. | 1. Create a topic in the Message Queue for Apache RocketMQ console.<br>2.Restart your application.<br>For more information, see the "Nonexistent topic" section in the Nonexistent resources topic. | >=4.0.1 |
| 26 | SUBSCRIPTION_GROUP_NOT_EXIST | This error is returned if the specified group ID does not exist. | 1.Create a group ID in the Message Queue for Apache RocketMQ console.<br>2.Restart your application.<br>For more information, see the "Nonexistent group ID" section in the Nonexistent resources topic. | >=4.0.1 |
| 24 | SUBSCRIPTION_NOT_EXIST | This error is returned when the subscription does not exist. | 1.Check whether the consumers identified by the group ID have been started.<br>2.Check whether subscription inconsistency occurs between consumers identified by the group ID. | >=4.0.1 |

| HTTP status code | Status flag | Description | Cause and recommended solution | Broker version |
| --- | --- | --- | --- | --- |
| 23 | SUBSCRIPTION_PARSE_FAILED | This error is returned when the system failed to parse the subscription expression. | Check the corresponding topic subscription expression and tag. | >=4.0.1 |
| 25 | SUBSCRIPTION_NOT_LATEST | This error is returned if subscription inconsistency occurs. | If this status continues for a moment, it is automatically restored.<br>For more information, see Subscription inconsistency. | >=4.0.1 |
| 14 | SERVICE_NOT_AVAILABLE | This error is returned when messages cannot be sent. | The requested Message Queue for Apache RocketMQ broker is discontinued, the broker is abnormal and does not support write operations, or the broker is a standby broker.<br>For more information, see the "The message failed to be sent." section in the Usage-related exceptions topic. | >=4.0.1 |

| HTTP status code | Status flag | Description | Cause and recommended solution | Broker version |
| --- | --- | --- | --- | --- |
| 16 | NO_PERMISSION (message sending) | This error is returned when the request is invalid. | The requested Message Queue for Apache RocketMQ broker disallows write operations.<br>The topic on the requested Message Queue for Apache RocketMQ broker disallows write operations.<br>The requested Message Queue for Apache RocketMQ broker disallows transactional messages. | >=4.0.1 |
| 16 | NO_PERMISSION (message subscription) | This error is returned when the request is invalid. | The requested Message Queue for Apache RocketMQ broker disallows read operations.<br>The current consumer group does not have the read permissions.<br>The pulled topic disallows read operations.<br>The current consumer group disallows message broadcasting. | >=4.0.1 |

| HTTP status code | Status flag | Description | Cause and recommended solution | Broker version |
| --- | --- | --- | --- | --- |
| 1 | SYSTEM_ERROR | This error is returned when a system exception occurs. | This is a temporary timeout that results from the restart of the Message Queue for Apache RocketMQ broker or heavy load on the broker.<br>For more information, see the "The message failed to be sent." section in the Usage-related exceptions topic. | >=4.0.1 |
| 1 | SYSTEM_ERROR (permission verification) | This error is returned when the permission verification fails. | Check whether the user is granted the permissions to publish messages to and subscribe messages from the topic. | >=4.0.1 |
| 2 | SYSTEM_BUSY | This error is returned when the system is busy and the request is denied. | This is a temporary timeout that results from the restart of the Message Queue for Apache RocketMQ broker or heavy load on the broker.<br>For more information, see the "The message failed to be sent." section in the Usage-related exceptions topic. | >=4.0.1 |

# 1.1.7. Best practices

# 1.1.7.1. Clustering consumption and broadcasting consumption

This topic describes the terms, scenarios, and usage notes of clustering consumption and broadcasting consumption in Message Queue for Apache RocketMQ.

## Terms

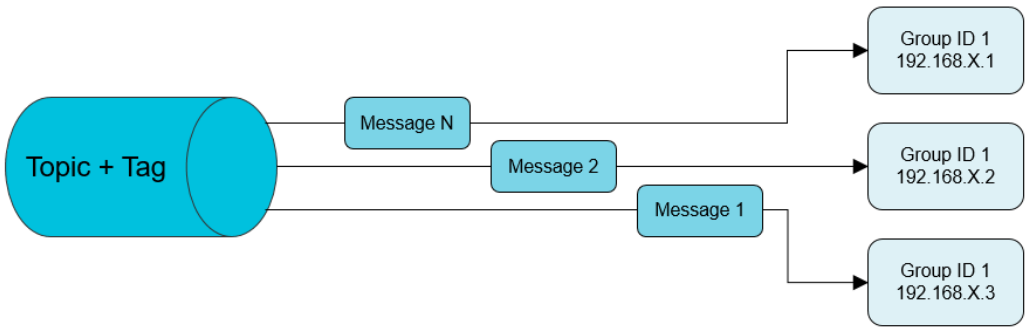
Message Queue for Apache RocketMQ is a messaging system that is based on the publish-subscribe model. In Message Queue for Apache RocketMQ, a consumer subscribes to a topic to obtain and consume messages. In most cases, consumer applications use distributed systems. Multiple machines are deployed in one cluster. Therefore, Message Queue for Apache RocketMQ defines the following terms:

- **Cluster:** Consumers identified by the same group ID belong to the same cluster. These consumers must have the same consumption logic that also involves tags. These consumers can be considered logically as one consumption node.
- **Clustering consumption:** In this mode, a message needs to be processed only by a consumer in the cluster.
- **Broadcasting consumption:** In this mode, Message Queue for Apache RocketMQ broadcasts each message to all consumers registered in the cluster to ensure that the message is consumed by each consumer at least once.

## Scenarios

- **Clustering consumption mode:**

  Clustering consumption mode

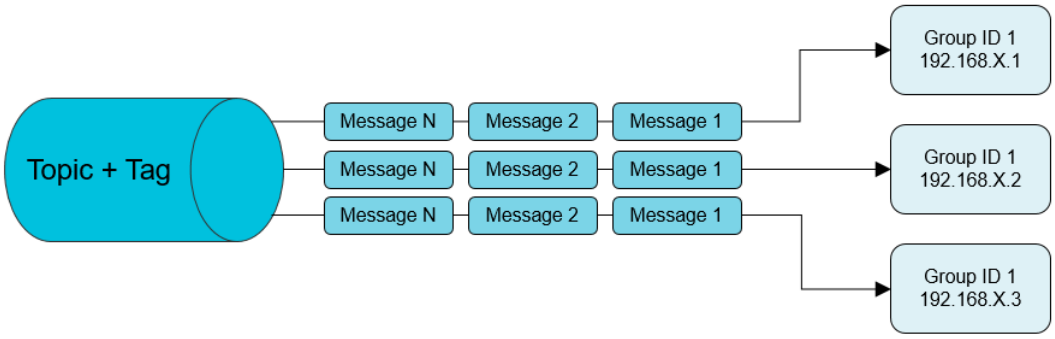



  **Scenarios and usage notes:**

  - Consumers are deployed in a cluster and each message needs to be processed only once.
  - The consumption progress is recorded on the Message Queue for Apache RocketMQ broker. Therefore, the reliability is high.
  - In clustering consumption mode, each message is delivered to only one consumer in the cluster for processing. If a message needs to be processed by each consumer in the cluster, use the broadcasting consumption mode.
  - In clustering consumption mode, it is not guaranteed that a failed message can be routed to the same consumer each time the message is redelivered. Therefore, no definitive assumptions can be made during message processing.

- **Broadcasting consumption mode:**

Broadcasting consumption mode



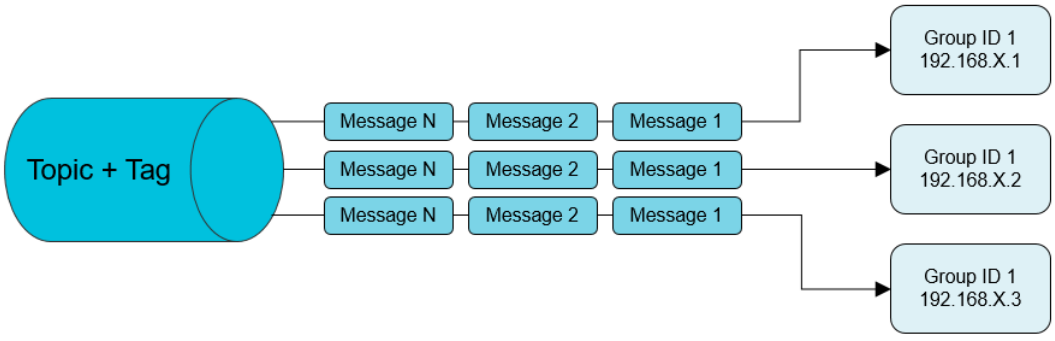

Scenarios and usage notes:

- Ordered messages are not supported in broadcasting consumption mode.
- Consumer offsets cannot be reset in broadcasting consumption mode.
- Each message needs to be processed by multiple consumers that are subject to the same logic.
- The consumption progress is recorded on the consumer. Duplicate messages are more likely to occur in broadcasting consumption mode than in clustering consumption mode.
- In broadcasting consumption mode, Message Queue for Apache RocketMQ ensures that each message is consumed at least once by each consumer, but does not resend a message that fails to be consumed. Therefore, you need to pay attention to consumption failures.
- In broadcasting consumption mode, a consumer starts consumption from the latest message each time the consumer is restarted. The consumer automatically skips the messages that are sent to the Message Queue for Apache RocketMQ broker when the consumer is stopped. Therefore, use this mode with caution.
- In broadcasting consumption mode, each message is repeatedly processed by many consumers. Therefore, we recommend that you use the clustering consumption mode whenever possible.
- Only Java clients support the broadcasting consumption mode.
- In broadcasting consumption mode, the Message Queue for Apache RocketMQ broker does not record the consumption progress. In this mode, you cannot query accumulated messages, configure message accumulation alerts, or query subscriptions in the Message Queue for Apache RocketMQ console.

- **Use the clustering consumption mode to simulate the broadcasting consumption mode**

  If the broadcasting consumption mode is required for your business, you can create multiple group IDs to subscribe to the same topic.

  Use the clustering consumption mode to simulate the broadcasting consumption mode




  Scenarios and usage notes:

- Each message needs to be processed by multiple consumers, and the logic of the consumers can be the same or different.
- The consumption progress is recorded on the Message Queue for Apache RocketMQ broker. Therefore, the reliability is higher than that in broadcasting consumption mode.
- For each group ID, one or more consumer instances can be deployed. When multiple consumer instances are deployed, these instances compose a cluster and work together to consume messages. Assume that three consumer instances C1, C2, and C3 are deployed for Group ID 1. These instances share the messages sent from the Message Queue for Apache RocketMQ broker to Group ID 1. In addition, these instances must subscribe to the same topics and same tags.

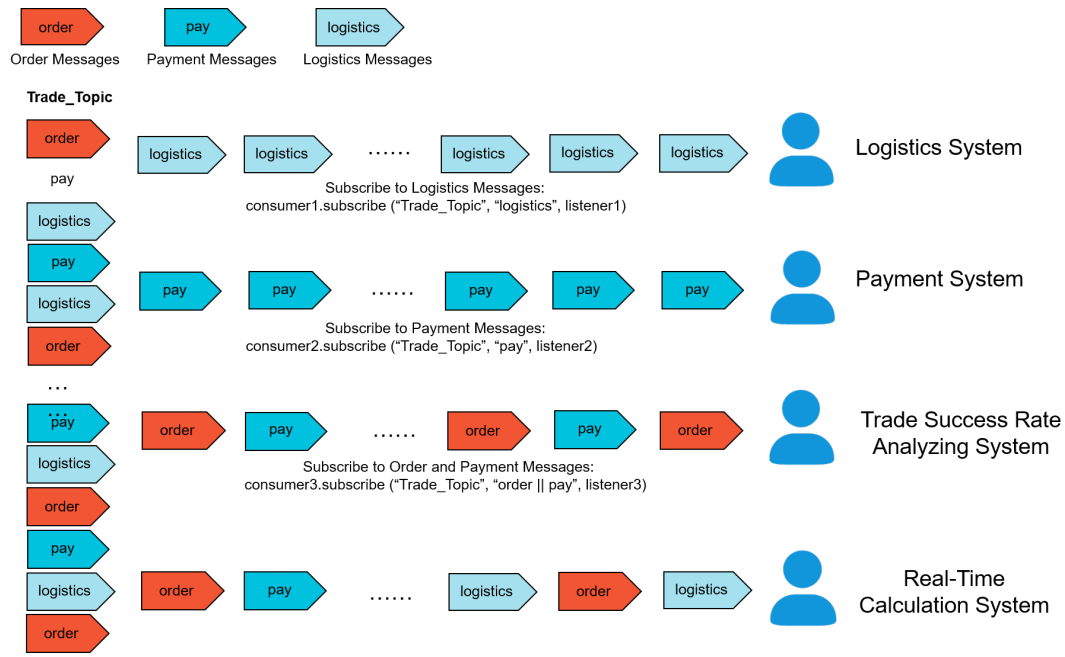
# 1.1.7.2. Message filtering

This topic describes how Message Queue for Apache RocketMQ consumers filter messages on the Message Queue for Apache RocketMQ broker based on tags.

A tag is used to classify messages in a topic into different types. Message Queue for Apache RocketMQ allows consumers to filter messages by using tags. This ensures that the consumers consume only messages that they are concerned with.

The following figure shows an example in the e-commerce transaction scenario. In the process from placing an order to receiving the product by the customer, a series of messages including order messages, payment messages, and logistics messages are generated. These messages are sent to the Trade_Topic topic and subscribed to by different systems, such as the payment system, analysis system for transaction success rate, and real-time computing system. Among these systems, the logistics system receives only logistics messages and the real-time computing system receives all transaction-related messages, including the order messages, payment messages, and logistics messages.

Filter messages



> **Note** To classify messages, you can create multiple topics, or create multiple tags in the same topic. In most cases, messages in one topic are irrelevant to those in another topic. Tags are used to distinguish between relevant messages in the same topic. For example, you can create different tags in the same topic to distinguish between a set and its subsets or distinguish between processes in sequence.

## Examples

- Send messages

  Specify a tag for each message before the message is sent.

```
Message msg = new Message("MQ_TOPIC","TagA","Hello MQ".getBytes());
```

- Subscribe to messages
  - Consumption method 1

    If a consumer needs to subscribe to messages of all types in a topic, use an asterisk (*) to represent all tags.

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

  - Consumption method 2

    If a consumer needs to subscribe to messages of a specific type in a topic, specify the corresponding tag.

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

  - Consumption method 3

    If a consumer needs to subscribe to messages of multiple types in a topic, separate the corresponding tags with two vertical bars (||).

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

- Consumption method 4 (error example)

  If a consumer subscribes to messages with specific tags in a topic multiple times, the tags in the last subscription prevail:

```
// In the following error code, the consumer can receive only messages with TagB in MQ_
TOPIC and cannot receive messages with TagA.
consumer.subscribe("MQ_TOPIC", "TagA", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        System.out.println(message.getMsgID());
        return Action.CommitMessage;
    }
});
consumer.subscribe("MQ_TOPIC", "TagB", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        System.out.println(message.getMsgID());
        return Action.CommitMessage;
    }
});
```

# 1.1.7.3. Subscription consistency

In Message Queue for Apache RocketMQ, a group ID represents a consumer instance group. For most distributed applications, multiple consumer instances are attached to the same group ID. Subscription consistency means that the processing logic of all consumer instances identified by the same group ID must be identical. If the subscriptions of the consumer instances are inconsistent, errors occur in the message consumption logic and messages may be lost.

Subscriptions in Message Queue for Apache RocketMQ involve topics and tags. Therefore, all consumer instances identified by the same group ID must be consistent in the following two aspects to ensure subscription consistency:

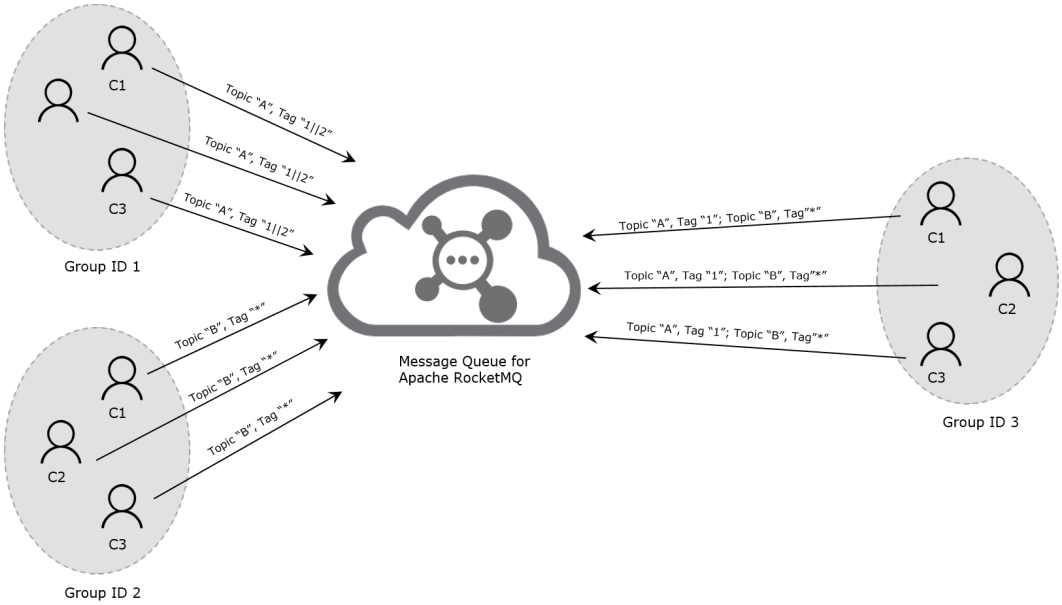
- The topics to which the consumer instances subscribe must be the same.
- The tags of the topics to which the consumer instances subscribe must be the same.

## Examples of subscriptions

- Consistent subscriptions

  Multiple group IDs subscribe to multiple topics, and the subscriptions of different consumer instances identified by the same group ID are consistent, as shown in the following figure.

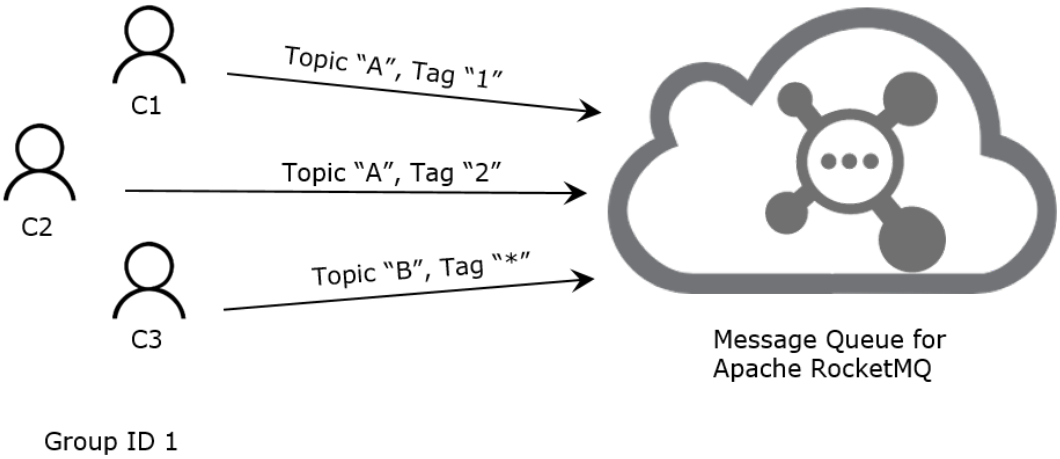
  Consistent subscriptions




- Inconsistent subscriptions

  One group ID subscribes to multiple topics, but the subscriptions of different consumer instances identified by the group ID are inconsistent, as shown in the following figure.

  Inconsistent subscriptions




## Sample code of subscriptions

Sample code of inconsistent subscriptions

- Example 1

  In the following example, two consumer instances identified by the same group ID subscribe to different topics.

  Consumer instance 1-1:

```
Properties properties = new Properties();
properties.put(PropertyKeyConst.GROUP_ID, "GID_jodie_test_1");
Consumer consumer = ONSFactory.createConsumer(properties);
consumer.subscribe("jodie_test_A", "*", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        System.out.println(message.getMsgID());
        return Action.CommitMessage;
    }
});
```

Consumer instance 1-2:

```
Properties properties = new Properties();
properties.put(PropertyKeyConst.GROUP_ID, "GID_jodie_test_1");
Consumer consumer = ONSFactory.createConsumer(properties);
consumer.subscribe("jodie_test_B ", "*", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        System.out.println(message.getMsgID());
        return Action.CommitMessage;
    }
});
```

- Example 2

  In the following example, two consumer instances identified by the same group ID subscribe to the same topic but subscribe to different numbers of tags of the topic. Consumer instance 2-1 has subscribed to Tag A, whereas consumer instance 2-2 has not specified a tag.

  Consumer instance 2-1:

```
Properties properties = new Properties();
properties.put(PropertyKeyConst.GROUP_ID, "GID_jodie_test_2");
Consumer consumer = ONSFactory.createConsumer(properties);
consumer.subscribe("jodie_test_A", "TagA", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        System.out.println(message.getMsgID());
        return Action.CommitMessage;
    }
});
```

  Consumer instance 2-2:

```
Properties properties = new Properties();
properties.put(PropertyKeyConst.GROUP_ID, "GID_jodie_test_2");
Consumer consumer = ONSFactory.createConsumer(properties);
consumer.subscribe("jodie_test_A", "*", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        System.out.println(message.getMsgID());
        return Action.CommitMessage;
    }
});
```

- Example 3

  In this example, the subscriptions are inconsistent due to the following reasons:

- Two consumer instances identified by the same group ID subscribe to different numbers of topics.
- Both the consumer instances subscribe to one same topic but subscribe to different tags of the topic.

Consumer instance 3-1:

```
Properties properties = new Properties();
properties.put(PropertyKeyConst.GROUP_ID, "GID_jodie_test_3");
Consumer consumer = ONSFactory.createConsumer(properties);
consumer.subscribe("jodie_test_A", "TagA", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        System.out.println(message.getMsgID());
        return Action.CommitMessage;
    }
});
consumer.subscribe("jodie_test_B", "TagB", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        System.out.println(message.getMsgID());
        return Action.CommitMessage;
    }
});
```

Consumer instance 3-2:

```
Properties properties = new Properties();
properties.put(PropertyKeyConst.GROUP_ID, "GID_jodie_test_3");
Consumer consumer = ONSFactory.createConsumer(properties);
consumer.subscribe("jodie_test_A", "TagB", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        System.out.println(message.getMsgID());
        return Action.CommitMessage;
    }
});
```

# 1.1.7.4. Message delivery retry

## 1.1.7.4.1. Overview

This topic describes how the Message Queue for Apache RocketMQ broker redelivers an ordered message or unordered message after the ordered message or unordered message fails to be consumed.

**Note** Delivery retries of messages take effect only in clustering consumption mode. If a consumer fails to consume a message in broadcasting consumption mode, the Message Queue for Apache RocketMQ broker does not attempt to redeliver the message to the consumer. In this case, the consumer skips the message and continues to consume new messages.

## 1.1.7.4.2. Delivery retries for ordered messages

This topic describes how to redeliver an ordered message after the message fails to be sent.

If a consumer failed to consume an ordered message, the Message Queue for Apache RocketMQ broker automatically keeps redelivering the message at an interval of 1 second. This may block message consumption. Therefore, when you use ordered messages, make sure that your application can promptly detect and handle consumption failures to prevent blocking.

# 1.1.7.4.3. Delivery retries for unordered messages

This topic describes how to redeliver an unordered message after the message fails to be sent.

Unordered messages include normal, scheduled, delayed, and transactional messages. You can specify a response code to trigger message delivery retry if a consumer fails to consume such messages.

Delivery retries of unordered messages take effect only in clustering consumption mode. If a consumer fails to consume an unordered message in broadcasting consumption mode, the Message Queue for Apache RocketMQ broker does not attempt to redeliver the message to the consumer. In this case, the consumer skips the message and continues to consume new messages.

## Number of message delivery retries

By default, Message Queue for Apache RocketMQ allows a maximum of 16 delivery retries for each message. The following table lists the intervals between delivery retries.

Intervals between delivery retries of an unordered message

| Nth delivery retry | Interval | Nth delivery retry | Interval |
|---|---|---|---|
| 1 | 10 seconds | 9 | 7 minutes |
| 2 | 30 seconds | 10 | 8 minutes |
| 3 | 1 minute | 11 | 9 minutes |
| 4 | 2 minutes | 12 | 10 minutes |
| 5 | 3 minutes | 13 | 20 minutes |
| 6 | 4 minutes | 14 | 30 minutes |
| 7 | 5 minutes | 15 | 1 hour |
| 8 | 6 minutes | 16 | 2 hours |

If a message fails to be consumed after 16 delivery retries, the message will not be delivered. If a message fails to be consumed, the Message Queue for Apache RocketMQ broker attempts to redeliver the message for up to 16 times within the next 4 hours and 46 minutes until the message is consumed. If the message is not consumed after the 16 deliveries, the message is no longer delivered.

> **Note**
>
> The message ID of a redelivered message does not change regardless of the number of delivery retries for the message.

## Configure message delivery retries

### Implement message delivery retries after a message consumption failure

If you want to implement message delivery retries in clustering consumption mode, you must configure message delivery retries in the implementation of the MessageListener class by using one of the following methods:

- Return Action.ReconsumeLater. We recommend that you use this method.
- Return null.
- Throw an exception.

#### Sample code

```
public class MessageListenerImpl implements MessageListener {
    @Override
    public Action consume(Message message, ConsumeContext context) {
        //Method 3: The message processing logic throws an exception. The message will be r
edelivered.
        doConsumeMessage(message);
        //Method 1: Return Action.ReconsumeLater. The message will be redelivered.
        return Action.ReconsumeLater;
        //Method 2: Return null. The message will be redelivered.
        return null;
        //Method 3: Throw an exception. The message will be redelivered.
        throw new RuntimeException("Consumer Message exceotion");
    }
}
```

### Implement no message delivery retries after a message consumption failure

If you do not want to implement delivery retries for a message in clustering consumption mode, exceptions thrown by the consumption logic must be caught, and Action.CommitMessage must be returned. After Action.CommitMessage is returned, the message will not be redelivered.

#### Sample code

```
public class MessageListenerImpl implements MessageListener {
    @Override
    public Action consume(Message message, ConsumeContext context) {
        try {
            doConsumeMessage(message);
        } catch (Throwable e) {
            // Exceptions thrown by the consumption logic are caught, and Action.CommitMess
age is returned.
            return Action.CommitMessage;
        }
        // If the message is properly processed, Action.CommitMessage is directly returned.
        return Action.CommitMessage;
    }
}
```

### Customize the maximum number of message delivery retries

> **Note** To customize the logging configuration of your Message Queue for Apache RocketMQ client, you must update TCP client SDK for Java to version 1.2.2 or later.

Message Queue for Apache RocketMQ allows you to specify the maximum number of message delivery retries when a consumer is started. The interval between delivery retries must comply with the following policies:

- If the maximum number of message delivery retries is 16 or less, the interval between delivery retires is as described in the preceding table.
- If the maximum number of message delivery retries is greater than 16, the interval between the 17th and later delivery retries is 2 hours.

The following sample code provides an example on how to specify the maximum number of message delivery retries:

```
Properties properties = new Properties();
// Set the maximum number of message delivery retries for the corresponding group ID to 20.
properties.put(PropertyKeyConst.MaxReconsumeTimes,"20");
Consumer consumer =ONSFactory.createConsumer(properties);
```

> **Note**
>
> - The specified maximum number of message delivery retries applies to all consumer instances identified by the same group ID.
> - If you specify the maximum number of message delivery retries for one of two consumer instances identified by the same group ID, the specified maximum number applies to both the consumer instances.
> - The configuration of the last started consumer instance overwrites the configurations of previously started consumer instances.

### Obtain the number of message delivery retries

After a consumer receives a message, the consumer can obtain the number of delivery retries for the message. The following sample code provides an example:

```
public class MessageListenerImpl implements MessageListener {
    @Override
    public Action consume(Message message, ConsumeContext context) {
        // Obtain the number of message delivery retries.
        System.out.println(message.getReconsumeTimes());
        return Action.CommitMessage;
    }
}
```

## 1.1.7.5. Consumption idempotence

After a Message Queue for Apache RocketMQ consumer receives messages, the consumer needs to perform idempotent processing on these messages based on the unique business-specific keys of the messages.

## Necessity for consumption idempotence

In Internet applications, duplicate messages may occur in Message Queue for Apache RocketMQ especially if Internet connection is unstable. Duplicate messages may occur in the following two scenarios:

- A producer repeatedly sends a message to the Message Queue for RocketMQ broker.

  If a network disconnection occurs or the producer breaks down after a message is sent to and persisted in the Message Queue for Apache RocketMQ broker, the broker fails to respond to the producer. If the producer realizes that the message failed to be sent and resends the message, the consumer subsequently receives two messages that have the same content and message ID.

- The Message Queue for Apache RocketMQ broker repeatedly delivers a message to a consumer.

  A message is delivered to a consumer and is processed by the consumer. However, a network disconnection occurs when the consumer sends a response to the Message Queue for Apache RocketMQ broker. To ensure that the message is consumed at least once, the Message Queue for Apache RocketMQ broker redelivers the previously processed message after the network connection recovers. The consumer subsequently receives two messages that have the same content and message ID.

- Duplicate messages are generated when rebalancing is triggered in scenarios such as network jitter, broker restart, and consumer application restart.

  Traffic is rebalanced if the Message Queue for Apache RocketMQ broker or consumer client is restarted or scaled. In this case, a consumer may receive duplicate messages.

## Solution

Message IDs may be duplicate. Therefore, we recommend that you do not implement idempotent processing based on message IDs. The best practice is to use unique business-specific keys for idempotent processing. The following sample code provides an example on how to specify a unique business-specific key for a message:

```
Message message = new Message();
message.setKey("ORDERID_100");
SendResult sendResult = producer.send(message);
```

The following sample code provides an example on how a consumer performs idempotent processing after it receives a message:

```
consumer.subscribe("ons_test", "*", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        String key = message.getKey()
        // Use the unique business-specific key for idempotent processing.
    }
});
```

# 1.1.7.6. MSHA

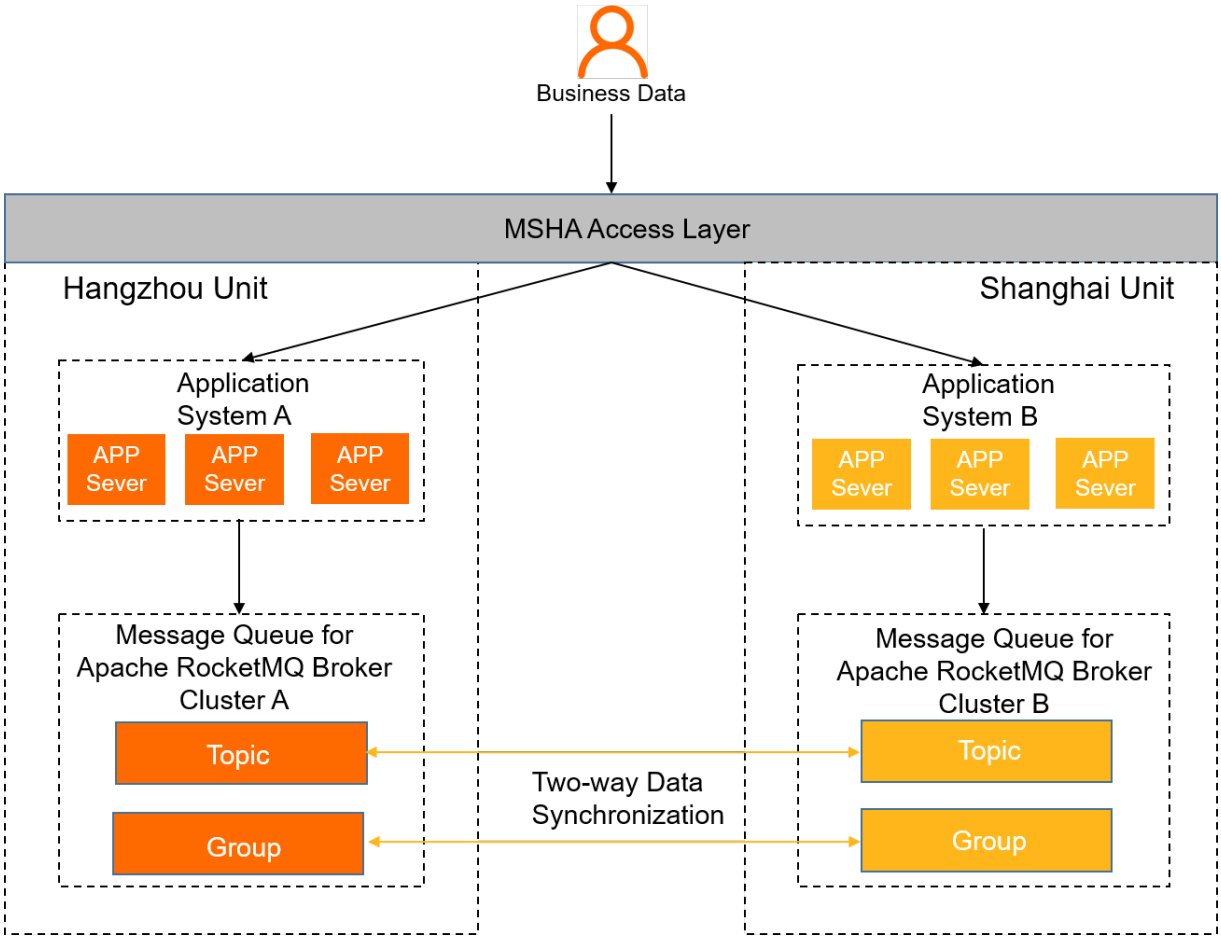
Relying on Alibaba Cloud Express Connect and Multi-Site High Availability (MSHA), supports active geo-redundancy. MSHA decouples business recovery from fault recovery by supporting two-way data synchronization and business traffic switchover among instances. This ensures business continuity when a fault occurs. This topic describes MSHA, its scenarios and benefits, and how to configure Message Queue for Apache RocketMQ to use MSHA.

## What is MSHA?

MSHA is an active geo-redundancy solution that was born and evolved in the e-commerce business environment of Alibaba. MSHA can help decouple business recovery from fault recovery. MSHA provides capabilities such as flexible scheduling based on traffic rules, cross-region and cross-cloud control, and data protection to implement fast business switchover and recovery when a fault occurs.

Message Queue for Apache RocketMQ relies on Alibaba Cloud Express Connect and MSHA to implement two-way synchronization of message data among instances in the same region or different regions. Different from a traditional disaster recovery solution, MSHA allows the systems that are deployed in different units to provide services at the same time. This not only implements disaster recovery but also improves business continuity and achieves remote resource scaling.

The following figure shows how to use MSHA to implement active geo-redundancy for Message Queue for Apache RocketMQ.




- A complete business system is deployed in both the Hangzhou and Shanghai units.
- The MSHA access layer routes business data to the two units based on traffic rules. The application systems and Message Queue for Apache RocketMQ broker clusters in the Hangzhou and Shanghai units process business data in their local regions.

- The broker clusters in the two units are configured to support MSHA. Data is synchronized between Broker Cluster A and Broker Cluster B in a two-way manner. The data includes topics, groups, and consumer offsets. Normally, the business systems in the Hangzhou and Shanghai units process business data only in their local regions, and synchronize message data of the local unit to the cluster of the other unit for disaster recovery.
- Assume that a disaster occurs in the Hangzhou unit and the entire business system in the Hangzhou unit fails. In this case, MSHA switches the business data of the Hangzhou unit to the Shanghai unit. Because of the MSHA configuration, Broker Cluster B of the Shanghai unit stores the business data of the Hangzhou unit and can continue to process the unfinished message data. This allows you to troubleshoot and fix the fault under the premise of ensuring that the business is not interrupted. This way, the business is recovered before the fault is rectified.
- After the Hangzhou unit is recovered from the fault, MSHA switches the business data of the Hangzhou unit back to the business system in the Hangzhou unit. During the whole process, users are not aware of the fault and the user experience is not affected.

## Scenarios

MSHA applies to the following typical business scenarios:

- Business scenarios in which business is divided into different units by region, such as logistics business. You can divide the logistics business based on the regions in which logistics orders are placed and send business data to production centers in different regions so that the data can be simultaneously processed. This improves resource utilization and business concurrency.
- Business scenarios that have strict requirements on the reliability of business data, such as financial and securities systems. If a system fault occurs, the transaction results are greatly affected. In this case, MSHA can switch the business to the disaster recovery site, which can continue to process unfinished message data based on the synchronized data.

## Benefits

- **Availability**

  Compared with a traditional disaster recovery solution, MSHA implements two-way data synchronization among production centers. All product centers can provide services at the same time. This implements traffic balancing and improves resource utilization.

- **Fast fault recovery**

  MSHA effectively ensures business continuity. When one of the production centers fails, MSHA immediately switches the business to other production centers to ensure business continuity. This is different from a traditional solution in which the fault must be located and fixed before the business can be recovered. The decouples business recovery from fault recovery.

- **Remote resource scaling**

  The limited resources in a single place may not meet the requirements as the business rapidly develops. In addition, the business may face bottlenecks such as limited storage and computing network performance. The horizontal scaling capability of Message Queue for Apache RocketMQ allows the business to be scaled in other data centers or regions to reduce cost waste.

## Configuration method

For more information about how to configure Message Queue for Apache RocketMQ to use MSHA, see the topic *Multi-site configuration > Active geo-redundancy > Configure Message Queue for Apache RocketMQ* in the *MSHA User Guide*.

# 1.1.8. Service usage FAQ

## 1.1.8.1. FAQ

### 1.1.8.1.1. Quick start

This topic provides answers to questions frequently asked by new users when they use Message Queue for Apache RocketMQ.

1. Where do consumers identified by a new group ID start to consume?
   - If a consumer identified by the group ID is started for the first time, the consumer ignores the messages that are sent before the consumer is started. This means that the consumer ignores historical messages and starts to consume messages that are sent after the consumer is started.
   - If the consumer is started for the second time, the consumer starts consumption from the previous consumer offset.
   - If you want the consumer to start consumption from a specific offset, you can reset the previous consumer offset in the Message Queue for Apache RocketMQ console to specify a point in time from which the consumer starts to consume messages. Each reset affects only the specific topic under the specific group ID but does not affect other group IDs.
2. How does the Message Queue for Apache RocketMQ broker redeliver a message if the message fails to be consumed?
   - **Clustering consumption**

     In clustering consumption mode, if Action.ReconsumerLater or NULL is returned or an error occurs during consumption, the Message Queue for Apache RocketMQ broker attempts to redeliver the message for up to 16 times. If the message still fails to be consumed after the 16 delivery retries, the message is discarded. The following table describes the intervals between delivery retries.

| Nth delivery retry | Interval |
| --- | --- |
| 1 | 10 seconds |
| 2 | 30 seconds |
| 3 | 1 minute |
| 4 | 2 minutes |
| 5 | 3 minutes |
| 6 | 4 minutes |
| 7 | 5 minutes |
| 8 | 6 minutes |
| 9 | 7 minutes |
| 10 | 8 minutes |

| Nth delivery retry | Interval |
| --- | --- |
| 11 | 9 minutes |
| 12 | 10 minutes |
| 13 | 20 minutes |
| 14 | 30 minutes |
| 15 | 1 hour |
| 16 | 2 hours |

The message.getReconsumeTimes() method can be called to query the serial number of a delivery retry.

- **Broadcasting consumption**

  In broadcasting consumption mode, Message Queue for Apache RocketMQ guarantees that a message can be consumed at least once. If the message fails to be consumed, the Message Queue for Apache RocketMQ broker does not redeliver the message.

3. What do I do if a sent message is not received?

   Message Queue for Apache RocketMQ provides the following methods for Message query:

   - Specify a topic and time range to query all messages received by this topic within the specified time range.
   - Specify a topic and message ID to query messages by using exact match.
   - Specify a topic and message key to query messages with the same message key.

   You can use the preceding methods to query the specific content and consumption information of messages. To track the time and location of each role from the producer to the consumer in the entire trace of a message, you can use the message tracing feature provided by Message Queue for Apache RocketMQ. For more information, see Query the message trace.

4. Can Message Queue for Apache RocketMQ ensure that no duplicate messages are delivered to consumers?

   In most cases, Message Queue for Apache RocketMQ can ensure that no duplicate messages are delivered to consumers. As a distributed messaging middleware, Message Queue for Apache RocketMQ cannot ensure that no duplicate messages are delivered to consumers when exceptions such as network jitter and application processing timeout occur. However, Message Queue for Apache RocketMQ can ensure that no messages are lost.

# 1.1.8.1.2. Configurations

This topic provides answers to frequently asked questions about Message Queue for Apache RocketMQ configurations.

1. How long can messages be retained on the Message Queue for Apache RocketMQ broker?

   Messages can be retained on the Message Queue for Apache RocketMQ broker for up to three days. The system automatically deletes the unconsumed after the three days.

2. What is the maximum message body size in Message Queue for Apache RocketMQ?

The maximum message body size in Message Queue for Apache RocketMQ varies with the message type. The following information shows the maximum message body size for different types of messages:

- A normal or ordered message: 4 MB
- A transactional, scheduled, or delayed message: 64 KB

3. How do I set the number of consumer threads on a Message Queue for Apache RocketMQ consumer?

   To set the number of consumer threads on a Message Queue for Apache RocketMQ consumer, set the ConsumeThreadNums attribute when you start the consumer. The following sample code provides an example on how to set the number of consumer threads:

```
public static void main(String[] args) {
    Properties properties = new Properties();
    properties.put(PropertyKeyConst.GROUP_ID, "GID_001");
    properties.put(PropertyKeyConst.AccessKey, "xxxxxxxxxxx");
    properties.put(PropertyKeyConst.SecretKey, "xxxxxxxxxxx");
    /**
     * Set the number of consumer threads to 20.
     */
    properties.put(PropertyKeyConst.ConsumeThreadNums,20);
    Consumer consumer =ONSFactory.createConsumer(properties);
    consumer.subscribe("TestTopic", "*", new MessageListener() {
        public Action consume(Message message, ConsumeContext context) {
            System.out.println("Receive: " + message);
            return Action.CommitMessage;
        }
    });
    consumer.start();
    System.out.println("Consumer Started");
}
```

4. What do I do if an error in loading DLL or another running error occurs due to invalid .NET client configuration?

   For more information, see *SDK_GUIDE.pdf* in the compressed package of SDK for .NET to verify that the project configuration is the same as that described in the document.

# 1.1.8.1.3. Message tracing

This topic provides answers to frequently asked questions about the message tracing feature of Message Queue for Apache RocketMQ.

1. Why is trace data not found?

   If no trace data is found based on the specified query conditions, check whether the following requirements are met:

   i. Only Java clients of version 1.2.2 or later support the message tracing feature.

   ii. Check whether the query conditions are properly specified. This means that you need to check whether the topic name, message ID, and message key are properly entered.

iii. Check whether the query time range is correct. To accelerate the query, you must specify the range of the message sending time. If you still cannot retrieve the data, expand the time range and try again.

iv. If the preceding settings are correct but the trace data is still not found, contact the technical support and provide the related log file. The path to the log file is */home/{user}/logs/ons.log*.

v. If the preceding settings are correct but the trace data is still not found, submit a ticket to seek help from the technical support and provide the log file. The path to the log file is */home/{user}/logs/ons.log*.

2. What do I do if the consumption information about a consumed message is not included in the trace data and the client IP address and group ID in the trace data are wrong?

   This problem occurs because the client is not updated to the version that supports the message tracing feature. Therefore, the message tracing module of Message Queue for Apache RocketMQ can obtain only some trace data, and the displayed result is abnormal. We recommend that you upgrade your client as soon as possible. For more information about the message tracing feature, see Query the message trace.

3. Why is my group ID not shown in the list of consumers?

   The possible cause is that a large number of downstream consumers have subscribed to messages, and the space in the tracing map is insufficient to display all the data. Move the pointer over the scroll bar and scroll down to see all the data.

4. Why are previous query tasks not displayed?

   A large number of historical query tasks affect the display result. Therefore, Message Queue for Apache RocketMQ regularly cleans up historical query tasks and retains only query tasks created within the recent seven days. If you cannot find a historical task, query it again.

# 1.1.8.1.4. Alert handling

Alert handling is unavailable for Message Queue for Apache RocketMQ.

Apsara Stack provides an isolated cloud-based environment and cannot be connected to the APIs of Internet services, such as the short message service (SMS) gateway. Therefore, the monitoring and alerting module in the console is unavailable.

# 1.1.8.1.5. Ordered messages

This topic provides answers to frequently asked questions about ordered messages in Message Queue for Apache RocketMQ.

1. Do ordered messages support clustering consumption and broadcasting consumption?

   Ordered messages support clustering consumption but do not support broadcasting consumption.

2. Can a message be an ordered message, a scheduled message, and a transactional message at the same time?

   No, a message cannot be an ordered message, a scheduled message, and a transactional message at the same time. Ordered messages, scheduled messages, and transactional messages are different and mutually exclusive message types.

3. What is the usage scope of ordered messages?

Ordered messages are messages that are guaranteed to be consumed in the order they are sent within the same topic. Ordered messages are classified into globally ordered messages and partially ordered messages.

4. Why is the performance of globally ordered messages mediocre?

   Globally ordered messages are processed in first-in-first-out (FIFO) order. If the previous message is not consumed, the next message will be stored in a queue of the corresponding topic until the previous message is consumed. To improve the transactions per second (TPS) of globally ordered messages, upgrade the specifications of the host that runs the message client, and reduce as much as possible the time required by the message client application to process the local business logic.

5. What transmission modes do ordered messages support?

   Ordered messages support only the reliable synchronous transmission mode.

# 1.1.8.2. Exceptions

# 1.1.8.2.1. Usage-related exceptions

This topic describes the exceptions that may occur when you use Message Queue for Apache RocketMQ. This topic also provides solutions.

1. The producer or consumer failed to be started, or duplicate group IDs exist.

   Cause:

   You attempt to start multiple producer or consumer instances identified by the same group ID in one JVM process. This results in client startup failures.

   Solution:

   Perform the following steps:

   i. Make sure that only one producer instance identified by a group ID and one consumer instance identified by a group ID are started in one JVM process. This means that you cannot start multiple producer instances identified by the same group ID or multiple consumer instances identified by the same group ID in the same JVM process.

   ii. Restart your application.

2. In broadcasting consumption mode, an error occurred when the JSON file is loaded for consumer startup.

   Cause:

   The Fastjson version is much earlier. In broadcasting consumption mode, the consumer failed to load the local *offsets.json* file and failed to be started.

   Solution:

   Update Fastjson to a version supported by ons-client and make sure that the *offsets.json* file can be normally loaded. By default, the *offsets.json* file is located in the */home/{user}/.rocketmq_offsets/* directory.

3. The queue list failed to be obtained when the consumer subscribes to messages.

   Cause:

   You did not create this topic in the console. As a result, the consumer failed to obtain the queue information of the topic during startup.

Solution:

Perform the following steps:

i. Log on to the Message Queue for Apache RocketMQ console. In the left-side navigation pane, click **Topics**. On the Topics page, click **Create Topic**.

ii. In the left-side navigation pane, click **Groups**. On the Groups page, click **Create Group ID** to create a group ID as prompted.

iii. Restart your application.

4. The message failed to be sent.

   The message failed to be sent after multiple delivery retries.

   Cause:

   i. The Message Queue for Apache RocketMQ broker returned an error code to the producer. For more information about the error code, see the nested exception that corresponds to this exception.

   ii. After the Message Queue for Apache RocketMQ broker unexpectedly fails and before the producer detects the latest broker list, this exception temporarily occurs.

   iii. The producer timed out when it attempted to send a message. This problem may be caused by heavy load on the broker or unstable network connectivity.

   Solution:

   Perform the following steps:

   i. Try again later. This exception is temporary. The temporary timeout might be caused by the restart of the Message Queue for Apache RocketMQ broker or heavy load on the broker.

   ii. If the problem persists after you try for several times, contact technical support engineers.

5. No exception is recorded.

   Problem description:

   No exception is recorded.

   Solution:

   Contact technical support engineers.

6. The status of the message is Consumed, but the consumer is not aware of this.

   The status of the message is Consumed, but the consumer log shows that the message is not received. This problem is due to the following three reasons:

   - The business code defines that the message is not immediately printed after the message is received.

     If the business logic is directly executed after a message is received, the message information is not recorded in the log if the code misses a specific logic branch. This leads to the false symptom that the message is not received.

     We recommend that you immediately print the message information after you receive a message to keep the information such as messageId, timestamp, and reconsumeTime.

   - Multiple consumer instances are deployed.

A consumer is often restarted multiple times at the debugging stage. If the previous process does not exit before the next process starts, multiple consumption processes coexist. In this case, multiple consumer instances share the message information. This scenario is similar to clustering consumption. A message that fails to be received by one consumer is received by another consumer.

Log on to the Message Queue for Apache RocketMQ console. In the left-side navigation pane, click **Groups**. On the Groups page, select your instance and click **Consumer Status** in the Actions column. In the Consumer Status panel, view **Connection Information**. The deployment information of consumer instances is displayed, including the number of consumer instances and the IP address of each instance. You can check for the problem based on the information.

- An exception that failed to be caught occurred during the consumption of a message. As a result, the message is redelivered.

```
public class MessageListenerImpl implements MessageListener {
    @Override
    public Action consume(Message message, ConsumeContext context) {
        // The message processing logic throws an exception. The message will be redeli
vered.
        doConsumeMessage(message);
        // If an exception that is not caught occurs in the doConsumeMessage() method,
this line of log is not printed.
        log.info("Receive Message, messageId:", message.getMsgID());
        return Action.CommitMessage;
    }
}
```

If the problem persists, contact technical support engineers and provide the local SDK logs.

# 1.1.8.2.2. Nonexistent resources

This topic describes exceptions related to nonexistent resources and provides solutions.

1. Nonexistent group ID

   Cause:

   The group ID is not created in the Message Queue for Apache RocketMQ console. As a result, when the group ID is used to connect to the Message Queue for Apache RocketMQ broker, verification fails on the broker.

   Solution:

   Perform the following steps:

   i. Log on to the Message Queue for Apache RocketMQ console. In the left-side navigation pane, click **Groups**.
      - If the group ID already exists, proceed to the next step.
      - If the group ID does not exist, create the group ID. Then, perform the next step.
   ii. Restart your application.
2. Nonexistent hostname

   Cause:

A possible cause is that the correct hostname or host IP address cannot be retrieved. To verify this assumption, run the **hostname** command.

If the correct hostname cannot be retrieved, this assumption is true. Otherwise, this problem may be due to another reason. In this case, contact technical support engineers.

Solution:

Perform the following steps:

i. On the host for which the error is reported, run the following command to check the hostname:

```
[root@iZ231wxgt6mZ ~]# hostname
iZ231wxgt6mZ
```

If an error is returned, check whether an alias is defined for the hostname. For example, an alias can be alias xxx='hostname' in .bash_profile or .bashrc. Another possible cause is that the command path does not point to $PATH.

ii. Ping the host.

```
[root@iZ231wxgt6mZ ~]# ping iZ231wxgt6mZ
```

If the hostname cannot be pinged, add the local IP address to the */etc/hosts* file. By default, each Elastic Compute Service (ECS) instance establishes a binding relationship between the local IP address and the hostname. Do not manually remove the relationship.

iii. Check the system configurations.

Check whether the hostname recorded in */etc/sysconfig/network* is the same as that added to */etc/hosts*. If the hostname is not the same as that added to /etc/hosts, modify the hostname. If you modify the content in */etc/sysconfig/network*, you must restart the host after you modify the content. This way, the modification can take effect. Exercise caution when you modify configurations in a system file, because this operation may cause other exceptions.

After the preceding three steps are performed, UnknownHostException will no longer be returned when your client starts.

# 1.1.8.2.3. Inconsistent status

This topic describes the exceptions related to inconsistent status and provides solutions.

1. Invalid messages

   **Cause:**

   The message attribute or content is invalid in the following scenarios:

   - The message is empty.
   - The message content is empty.
   - The message content is 0 character in length.
   - The length of the message content exceeds the limit.

   **Solution:**

   Check whether the preceding exceptions occur to the message and handle the exceptions as prompted.

2. Invalid parameters

   **Cause:**

   The following table lists the cases in which the parameters are invalid.

| Nested exception | Description |
| --- | --- |
| consumeThreadMin Out of range [1, 1000] | The specified number of consumer threads is inappropriate. |
| consumeThreadMax Out of range [1, 1000] | The specified number of consumer threads is inappropriate. |
| messageListener is null | messageListener is not configured. |
| consumerGroup is null | The group ID is not specified. |
| msg delay time more than 40 day | The delay for the delivery of a scheduled message cannot exceed 40 days. |

   **Solution:**

   Perform the following steps:

   i. Modify the parameter settings for the client as prompted and make sure that the new parameter values are within the valid ranges.

   ii. Restart your application.

3. Abnormal client status

   **Cause:**

   i. After the consumer or producer is created, the return code does not show that the start() method is called to start the consumer or producer.

   ii. After the consumer or producer is created, the consumer or producer fails to start due to an exception in the start() process.

   iii. After the consumer or producer is created and the start() method is called, the return code shows that the shutdown() method is called to shut down the consumer or producer.

   **Solution:**

   Perform the following steps:

   i. Make sure that the start() method is called after the group ID is created. Make sure that the producer or consumer is started.

   ii. Check *ons.log* for exceptions that occur during the startup of the producer or consumer.

4. Subscription inconsistency

   **Problem description:**

   Multiple consumer instances are started in different JVM processes. Consumer instances identified by the same group ID subscribe to different topics, or subscribe to the same topic but different tags. As a result, the subscriptions of the consumer instances are inconsistent, and messages cannot be received as expected.

   **Sample code of inconsistent subscriptions:**

- Example 1: The consumer instance on JVM 1 and the consumer instance on JVM 2 use the same group ID GID-MQ-FAQ. The two consumer instances subscribe to different topics. The consumer instance on JVM 1 subscribes to MQ-FAQ-TOPIC-1, whereas the consumer instance on JVM 2 subscribes to MQ-FAQ-TOPIC-2.

  **Code on JVM-1:**

```
Properties properties = new Properties();
properties.put(PropertyKeyConst.GROUP_ID, "GID-MQ-FAQ");
Consumer consumer = ONSFactory.createConsumer(properties);
consumer.subscribe("MQ-FAQ-TOPIC-1", "NM-MQ-FAQ", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        System.out.println("Receive: " + message);
        return Action.CommitMessage;
    }
});
consumer.start();
```

  **Code on JVM-2:**

```
 Properties properties = new Properties();
properties.put(PropertyKeyConst.GROUP_ID, "GID-MQ-FAQ");
Consumer consumer = ONSFactory.createConsumer(properties);
consumer.subscribe("MQ-FAQ-TOPIC-2", "NM-MQ-FAQ", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        System.out.println("Receive: " + message);
        return Action.CommitMessage;
    }
});
consumer.start();
```

- Example 2: The consumer instance on JVM 1 and the consumer instance on JVM 2 use the same group ID GID-MQ-FAQ and subscribe to the same topic. However, the two consumer instances subscribe to different tags. The consumer instance on JVM 1 subscribes to NM-MQ-FAQ-1, whereas the consumer instance on JVM 2 subscribes to NM-MQ-FAQ-2.

  **Code on JVM-1:**

```
Properties properties = new Properties();
properties.put(PropertyKeyConst.GROUP_ID, "GID-MQ-FAQ");
Consumer consumer = ONSFactory.createConsumer(properties);
consumer.subscribe("MQ-FAQ-TOPIC-1", "NM-MQ-FAQ-1", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        System.out.println("Receive: " + message);
        return Action.CommitMessage;
    }
});
consumer.start();
```

  **Code on JVM-2:**

```
 Properties properties = new Properties();
properties.put(PropertyKeyConst.GROUP_ID, "GID-MQ-FAQ");
Consumer consumer = ONSFactory.createConsumer(properties);
consumer.subscribe("MQ-FAQ-TOPIC-1", "NM-MQ-FAQ-2", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        System.out.println("Receive: " + message);
        return Action.CommitMessage;
    }
});
consumer.start();
```

**Solution:**

If you start multiple consumer instances identified by the same group ID in different JVM processes, make sure that the topics and tags to which the consumer instances subscribe are the same.

# 1.1.8.3. Troubleshooting

# 1.1.8.3.1. Unexpected consumer connections

This topic describes the symptoms of an unexpected consumer connection, analyzes causes, provides a solution, and verifies the solution.

## Problem description

- [Symptom 1]: Some messages are sent but not received. After you query message traces in the Message Queue for Apache RocketMQ console, the returned information shows that some messages are sent to the Message Queue for Apache RocketMQ broker, but the broker does not deliver the messages to consumers. To query message traces, log on to Message Queue for Apache RocketMQ console. In the left-navigation pane, click **Message Tracing**. On the Message Tracing page, click **Create Query Task**. In the Create Query Task dialog box, click the **By Message ID** tab.
- [Symptom 2]: Some consumer IP addresses are not within the expected range and messages are accumulated on the consumers that correspond to these IP addresses. To query connection information about consumers, log on to the Message Queue for Apache RocketMQ console. In the left-side navigation pane, click **Groups**. On the Groups page, find the group ID whose connection information you want to view and click **Consumer Status** in the Actions column. In the Consumer Status panel, view the connection information in the **Connection Information** section.

## Problem analysis

**Analysis**: Log on to the Message Queue for Apache RocketMQ console. In the left-side navigation pane, click **Groups**. On the Groups page, find the group ID whose connection information you want to view and click **Consumer Status** in the Actions column. In the Consumer Status panel, view the connection information in the **Connection Information** section. The connection information about all consumers identified by the group ID are displayed. You can check the IP address and process ID of the unexpected consumer and check whether the configurations loaded by the process are valid. The configurations include the AccessKey ID, AccessKey secret, topic, and group ID. If the configurations are invalid, the consumer process occupies some queues but cannot properly consume messages.

**Cause**: In the same environment, if a consumer identified by the group ID and configured with an invalid AccessKey ID, AccessKey secret, and topic is started, this consumer process may occupy some queues of the topic but cannot properly consume messages. As a result, messages are accumulated on the Message Queue for Apache RocketMQ broker and cannot be properly delivered to downstream consumers whose IP addresses are within the expected range.

- **Confirmation**: Locate the faulty process based on the connection status and check the AccessKey ID, AccessKey secret, and topic of the process based on the */{user.home}/logs/ons.log* file or program code.
- **Solution**: This is a quick solution. Shut down the faulty consumer process first. Then, the accumulated messages will be immediately rebalanced and delivered to proper consumers. After the fault is rectified, restart the faulty process.

### Verification

Log on to the Message Queue for Apache RocketMQ console. In the left-side navigation pane, click **Groups**. On the Groups page, find the group ID that you want to view and click **Consumer Status** in the Actions column. In the Consumer Status panel, view the connection information of consumers identified by the group ID in the **Connection Information** section. The displayed information shows that IP addresses of all consumers are within the expected range and the value of **Consistent Subscription** is **Yes**.

## 1.1.8.3.2. Inconsistent subscriptions

This topic describes the symptoms of inconsistent subscriptions, analyzes causes, provides a solution, and verifies the solution.

### Problem description

- Consumers identified by a group ID failed to receive some messages to which they want to subscribe. To query messages, log on to the Message Queue for Apache RocketMQ console. In the left-side navigation pane, click **Message Query**. On the Message Query page, click the **By Message ID** tab. Specify the corresponding topic and message ID. The displayed information shows that the message has been consumed at least once. However, the message is considered unconsumed based on the consumption logic.
- The subscriptions of consumers identified by the group ID are inconsistent. To check whether the subscriptions of consumers are consistent, log on to the Message Queue for Apache RocketMQ console. In the left-side navigation pane, click **Groups**. On the Groups page, find the group ID and click **Consumer Status** in the Actions column. In the Consumer Status panel, the value of **Consistent Subscription** is **No**.

### Problem analysis

In Message Queue for Apache RocketMQ, a group ID represents a consumer instance group. For most distributed applications, multiple consumer instances are attached to the same group ID. Subscription consistency means that the topics and tags of all consumer instances identified by the same group ID must be identical.

If the consumer instances identified by the same group ID subscribe to different topics, or subscribe to the same topic but different tags, the subscriptions are inconsistent. If the subscriptions are inconsistent, errors occur in the message consumption logic and messages may be lost.

- **[Cause 1]:** The topics subscribed to by consumers with the same group ID are different.

**Example 1:** Two consumers identified by the group ID GID-MQ-FAQ subscribe to different topics: MQ-FAQ-TOPIC-1 and MQ-FAQ-TOPIC-2.

**Code on JVM-1:**

```
Properties properties = new Properties();
properties.put(PropertyKeyConst.GROUP_ID, "GID-MQ-FAQ");
Consumer consumer = ONSFactory.createConsumer(properties);
consumer.subscribe("MQ-FAQ-TOPIC-1", "NM-MQ-FAQ", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        System.out.println("Receive: " + message);
        return Action.CommitMessage;
    }
});
consumer.start();
```

**Code on JVM-2:**

```
Properties properties = new Properties();
properties.put(PropertyKeyConst.GROUP_ID, "GID-MQ-FAQ");
Consumer consumer = ONSFactory.createConsumer(properties);
consumer.subscribe("MQ-FAQ-TOPIC-2", "NM-MQ-FAQ", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        System.out.println("Receive: " + message);
        return Action.CommitMessage;
    }
});
consumer.start();
```

- **[Cause 2]:** Two consumers identified by the same group ID subscribe to the same topic but different tags.

  **Example:** Two consumers identified by the group ID GID-MQ-FAQ subscribe to the same topic but different tags: NM-MQ-FAQ-1 and NM-MQ-FAQ-2.

  **Code on JVM-1:**

```
Properties properties = new Properties();
properties.put(PropertyKeyConst.GROUP_ID, "GID-MQ-FAQ");
Consumer consumer = ONSFactory.createConsumer(properties);
consumer.subscribe("MQ-FAQ-TOPIC-1", "NM-MQ-FAQ-1", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        System.out.println("Receive: " + message);
        return Action.CommitMessage;
    }
});
consumer.start();
```

  **Code on JVM-2:**

```
Properties properties = new Properties();
properties.put(PropertyKeyConst.GROUP_ID, "GID-MQ-FAQ");
Consumer consumer = ONSFactory.createConsumer(properties);
consumer.subscribe("MQ-FAQ-TOPIC-1", "NM-MQ-FAQ-2", new MessageListener() {
    public Action consume(Message message, ConsumeContext context) {
        System.out.println("Receive: " + message);
        return Action.CommitMessage;
    }
});
consumer.start();
```

## Solution

Perform the following steps:

1. Check the subscription code of different consumers. Make sure that the subscriptions of all consumers identified by the same group ID are consistent. This means that the topics and tags subscribed to by the consumers are all identical.
2. Restart all consumer applications.

## Verification

- Consumers can receive messages as expected.
- Log on to the Message Queue for Apache RocketMQ console. In the left-side navigation pane, click **Groups**. On the Groups page, find the group ID that you want to view and click **Consumer Status** in the Actions column. In the Consumer Status panel, the value of **Consistent Subscription** is **Yes**.

# 1.1.8.3.3. Message accumulation

This topic describes the symptoms of message accumulation, analyzes causes, provides a solution, and verifies the solution.

## Problem description

- The value of **Accumulated Messages** is higher than expected. To query the number of accumulated messages, log on to the Message Queue for Apache RocketMQ console. In the left-side navigation pane, click **Groups**. On the Groups page, find the group ID that you want to view and click **Consumer Status** in the Actions column. In the Consumer Status panel, check the value of Accumulated Messages in the Connection Information section.
- Some messages have been sent to the Message Queue for Apache RocketMQ broker but are not delivered to consumers. To query message traces, log on to the Message Queue for Apache RocketMQ console. In the left-side navigation pane, click **Message Tracing**. On the Message Tracing page, click **Create Query Task**. In the Create Query Task dialog box, click the **By Message ID** tab. Specify the corresponding topic and message ID to query the trace of a message.

## Problem analysis

In Message Queue for Apache RocketMQ, messages are first sent to the broker. Then, consumers identified by the group ID pull some messages from the broker to the on-premises machine for consumption based on the current consumer offset. In the consumption process, it may take a long time to consume a single message due to various reasons, such as access to locked shared resources, competition for I/O and network resources, and no timeout set for HTTP calls. As a result, messages start to accumulate on the broker.

If messages are not accumulated, check whether the threshold value is excessively small and causes alerts on message accumulation.

## Solution

**Perform the following operations for troubleshooting:**

- Log on to the Message Queue for Apache RocketMQ console. In the left-side navigation pane, click **Resource Statistics**. On the Resource Statistics page, click the **Message Consumption** tab. Enter the information to query historical consumption records. If message writing is faster than message consumption, modify the code or scale out the consumer.
- Print the Jstack information **jstack -l {PID} | grep ConsumeMessageThread** in the application. If messages are blocked, print the Jstack information for five consecutive times and identify the spot where the consumer thread is stuck. Then, rectify the fault and restart the application. Check whether messages can be consumed.

## Verification

- Print the Jstack information **jstack -l {PID} | grep ConsumeMessageThread** in the application. Verify that no consumer thread is blocked.
- Log on to the Message Queue for Apache RocketMQ console. In the left-side navigation pane, click **Groups**. On the Groups page, find the group ID that you want to view and click **Consumer Status** in the Actions column. In the Consumer Status panel, check whether the value of **Real-time Consumption Speed** increases and the value of **Accumulated Messages** decreases.

# 1.1.8.3.4. Message accumulation in Java processes

## Problem description

In the **Consumer Status** panel of the Message Queue for Apache RocketMQ console, the number of real-time accumulated messages of the group ID is higher than expected, and the performance is much lower.

## Cause

The number of real-time accumulated messages of the group ID is higher than expected due to an excessive number of messages accumulated in the Java process.

## Solution

## Procedure

1. Log on to the Message Queue for Apache RocketMQ console. Navigate to the Consumer Status panel, obtain the host IP address of the consumer instance that has accumulated messages, and then log on to the host or container.
2. Run one of the following commands to view the PID of the Java process and record the PID:

   ```
   ps -ef |grep java
   ```

   ```
   jps -lm
   ```

3. Run the following command to view the stack information:

   ```
   jstack -l pid > /tmp/pid.jstack
   ```

4. Run the following command to view the information about `ConsumeMessageThread` and focus on the thread status and stack:

```
cat /tmp/pid.jstack|grep ConsumeMessageThread -A 10 --color
```

The following figure shows an example of command output.


For more information about the thread status, see official Java documentation.

> **Note** Message Queue for Apache RocketMQ can support 1 billion accumulated messages without compromising the performance. If the problem of compromised performance is not solved after you perform the preceding steps, contact O&M engineers and provide the following information:
>
> - The *heap.bin* file. Run the `jmap -dump:format=b,file=heap.bin [$PID]` command to obtain this file. Then, run the `gzip heap.bin` command to generate a compressed package.[$PID] represents the PID of the Java process recorded in Step 2.
> - The local *ons.log* file of the consumer client where messages are accumulated.
> - The version of the consumer client.

# 1.1.8.3.5. Application OOM due to message caching on the client

This topic describes the symptoms of application out of memory (OOM), analyzes causes, provides solutions, and verifies each solution.

## Problem description

- [Symptom 1]: The memory is exhausted on the machine where the application is deployed.
- [Symptom 2]: The keyword OutOfMemory can be found in `/{user.home}/logs/ons.log`.
- [Symptom 3]: In the Message Queue for Apache RocketMQ console, **Real-time Accumulated Messages** in the **Consumer Status** panel of the **Groups** page shows that a large number of messages are accumulated. The Connection Information section displays the number of accumulated messages for each connected consumer client. In addition, the result of check by running the jstack command indicates that `ConsumeMessageThread_` is not blocked.

## Analysis

**Analysis**: A Message Queue for Apache RocketMQ consumer proactively pulls messages from the Message Queue for Apache RocketMQ broker and caches them to the client. Then, the messages are consumed based on the consumption logic of the client. In versions earlier than 1.7.0.Final, the client caches up to 1,000 messages for each queue of each topic by default. Assume that each topic has 16 queues (two primary brokers and two secondary brokers, and eight queues on each broker). The average size of a message in this topic is 64 KB. The final message size cached for this topic on the client is calculated by using the following formula: 16 × 1000 × 64 KB = 1 GB. If you subscribe to eight topics at the same time and caches messages of all these topics in the client memory, the memory consumed will exceed the memory size specified in the JVM configuration. In this case, OOM occurs.

- [Cause 1]:An ons-client version earlier than 1.7.0.Final is depended on, and the average size of a message in each topic exceeds 4 KB. In addition, message consumption is slow. This is prone to message caching in the memory of the client.
  - **Confirmation**: Check whether the keyword OutOfMemory can be found in `/{user.home}/logs/ons.log`, or run the `jmap -dump:live,format=b,file=heap.bin <pid>` command to detect the objects that occupy a large amount of memory.
  - **Solution**: Update the ons-client version to 1.7.0.Final or later and set the `com.aliyun.openservices.ons.api.PropertyKeyConst#MaxCachedMess ageSizeInMiB` parameter to an appropriate value for the corresponding ConsumerBean. Then, restart the application.
- [Cause 2]:ons-client-1.7.0.Final or later is depended on, and the default maximum memory consumed is 512 MB, which is the total cache capacity of all topics to which consumer instances identified by a group ID subscribe. If the application still suffers OOM, set the `com.aliyun.openservices.ons.api.PropertyKeyConst#MaxCachedMessageSizeInMiB` parameter to a value within the valid range from 16 MB to 2048 MB to customize the maximum memory that can be consumed during the startup of ConsumerBean.
  - **Confirmation**: Check the ons-client version used by the application and check the memory size allocated to the process based on the JVM configuration.
  - **Solution**: Set the `com.aliyun.openservices.ons.api.PropertyKeyConst#MaxCachedMess ageSizeInMiB` parameter for the corresponding ConsumerBean, based on the memory usage of the machine where the application runs. Then, restart the application.

## Verification

- [Verification 1]: The keyword OutOfMemory disappears from `/{user.home}/logs/ons.log`.
- [Verification 2]: Log on to the Message Queue for Apache RocketMQ console. In the left-side navigation pane, click **Groups**. On the Groups page, select your instance and click **Consumer Status** in the Actions column. In the Consumer Status panel, the value of **Real-time Consumption Speed** increases, whereas the value of **Real-time Accumulated Messages** decreases.

# 1.1.8.3.6. AuthenticationException reported due to failure in sending or receiving messages

## Problem description

The application cannot send messages and `AuthenticationException` is reported in the *{user.home}/logs/ons.log* log of the host.

## Cause

A wrong AccessKey ID or AccessKey secret is used.

## Solution

## Procedure

1. Check whether you use an Apsara Stack tenant account or a Resource Access Management (RAM) user.

   The following table describes the Apsara Stack tenant account and RAM user.

| Apsara Uni-manager | RocketMQ |
| --- | --- |
| Organization administrator | Apsara Stack tenant account |
| Resource user | RAM user |

   You can create roles in the Apsara Uni-manager Management Console. If you want a role to become a resource user, the selected permissions must be consistent with the default configuration in the system.

2. Check the permissions of the user who creates resources.

   The Apsara Stack tenant account can create a topic and a group ID in the Message Queue for Apache RocketMQ console. The created resources are of the current organization level. A RAM user cannot create a topic, but can create a group ID. The created resource is of the RAM user level.

   - If you need to use a RAM user to send and receive messages, use the Apsara Stack tenant account to create a topic in the Message Queue for Apache RocketMQ console. For example, you can create a topic named Topic_bumen. Then, grant the permissions on the topic to a RAM user. At this point, the RAM user can view Topic_bumen in the Message Queue for Apache RocketMQ console. The RAM user can create its own group ID, for example, GID_zizhanghao. Then, the messaging program of the client can send and receive messages by using Topic_bumen, GID_zizhanghao, and the AccessKey ID and AccessKey secret of the RAM user.
   - If you need to use the topics and group IDs created by the Apsara Stack tenant account to send and receive messages, the AccessKey ID and AccessKey secret of the organization level must be configured because the topics and group IDs created by the Apsara Stack tenant account are of the organizational level and do not belong to the account itself.

# 2.API Gateway

## 2.1. User Guide

### 2.1.1. What is API Gateway?

API Gateway provides a comprehensive suite of API hosting services that help you share capabilities, services, and data with partners in the form of APIs.

- API Gateway provides multiple security mechanisms to secure APIs and reduce the risks arising from open APIs. These mechanisms include protection against replay attacks, request encryption, identity authentication, permission management, and throttling.
- API Gateway provides API lifecycle management that allows you to define, publish, and unpublish APIs. This improves API management and iteration efficiency.

API Gateway allows enterprises to reuse and share their capabilities with each other so that they can focus on their core business.

API Gateway


### 2.1.2. Log on to the API Gateway console

This topic describes how to log on to the API Gateway console.

#### Prerequisites

- The URL of the Apsara Uni-manager Management Console is obtained from the deployment personnel before you log on to the Apsara Uni-manager Management Console.
- We recommend that you use the Google Chrome browser.

#### Procedure

1. In the address bar, enter the URL of the Apsara Uni-manager Management Console. Press the Enter key.
2. Enter your username and password.

   Obtain the username and password that you can use to log on to the console from the operations administrator.

> **Note** When you log on to the Apsara Uni-manager Management Console for the first time, you must change the password of your username. Your password must meet complexity requirements. The password must be 10 to 32 characters in length and must contain at least two of the following character types:
>
> - Uppercase or lowercase letters
> - Digits
> - Special characters, which include ! @ # $ %

3. Click **Log On**.
4. In the top navigation bar, choose **Products > Application Services > API Gateway**.

# 2.1.3. Quick start

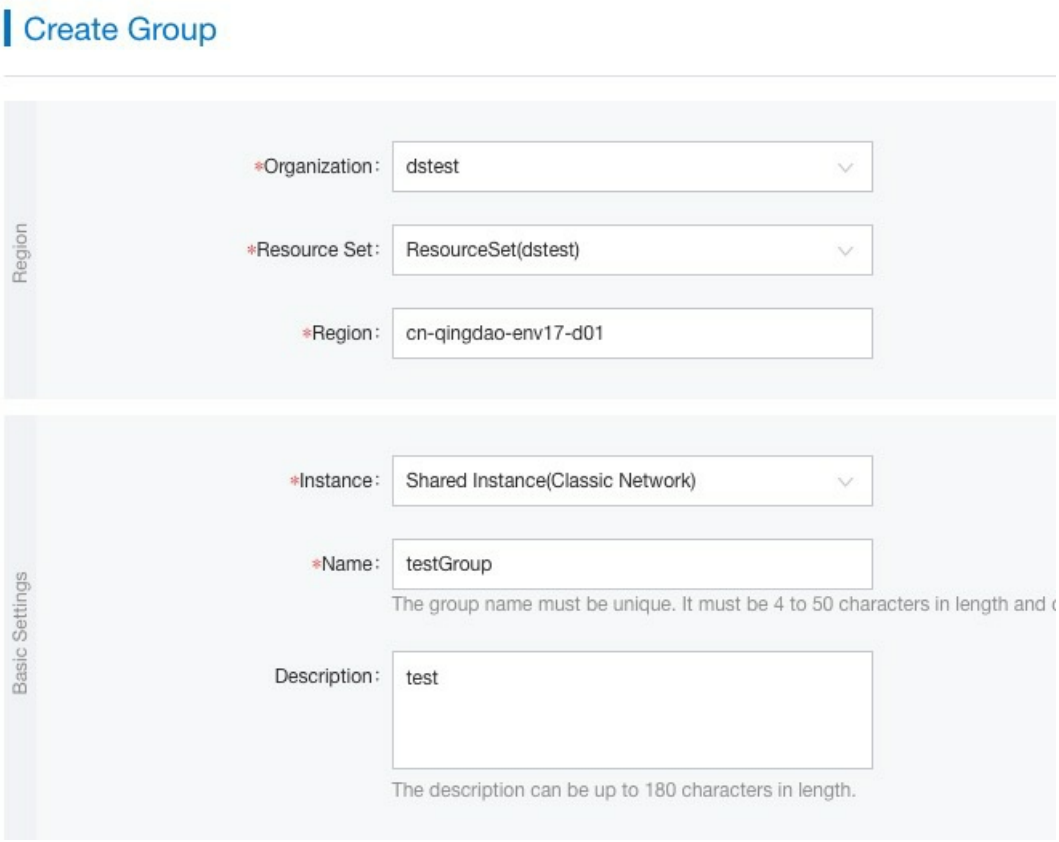
## 2.1.3.1. Create an API with HTTP as the backend service

This topic describes how to create and publish an API with HTTP as the backend service in API Gateway. This topic also describes how to call the API by using the AppKey and AppSecret pair of an app, with Security Certification set to Alibaba Cloud APP. You need to perform the following operations in sequence: **create an API group**, **create an API**, **create an app and an API authorization**, **debug the API**, and **call the API**.

### Create an API group

APIs are managed in API groups. Before you create an API, you must create an API group.

1. **Create a group**: In the left-side navigation pane of the API Gateway console, choose **Publish APIs > API Groups**. On the Group List page, click **Create Group** in the upper-right corner. On the Create Group page, specify **Organization**, **Resource Set**, and **Region**, set Name to testAppkeyGroup, and then click **Submit**. After you specify Organization, the Instance parameter is automatically set to Shared Instance(Classic Network).

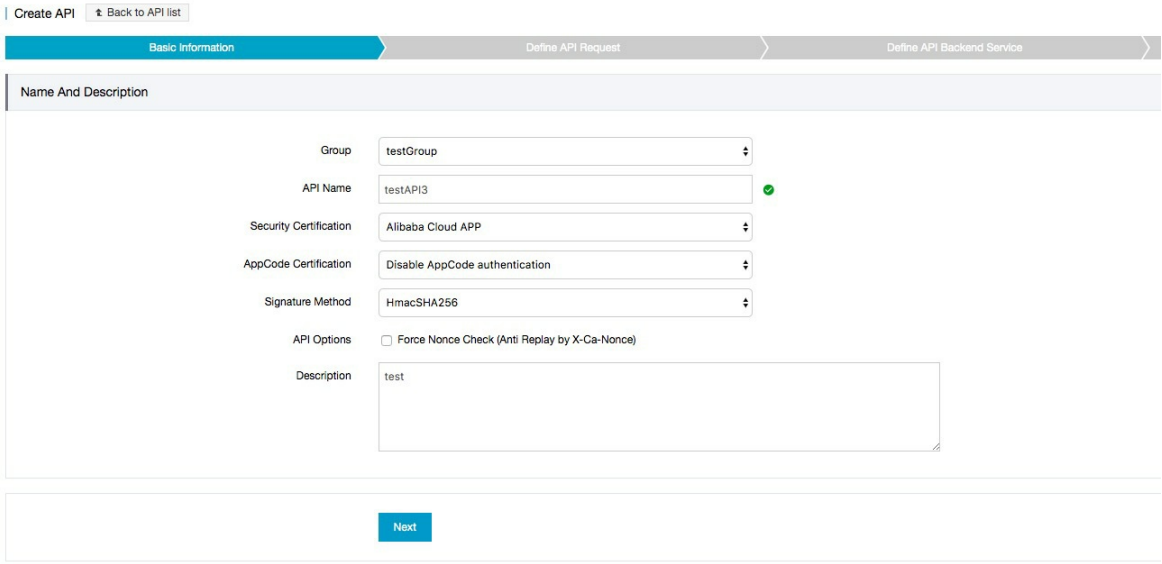



2. **View group information**: In the Submitted message, click Back to Console. On the **Group List** page, click **Refresh** in the upper-right corner. The group that you created is displayed. You can click the group name to go to the Group Details page and perform operations such as binding a domain name and modifying basic information. A second-level domain is automatically created for the API group. It can be used in Apsara Stack to call all APIs within this group. In this example, the domain name is used for tests.

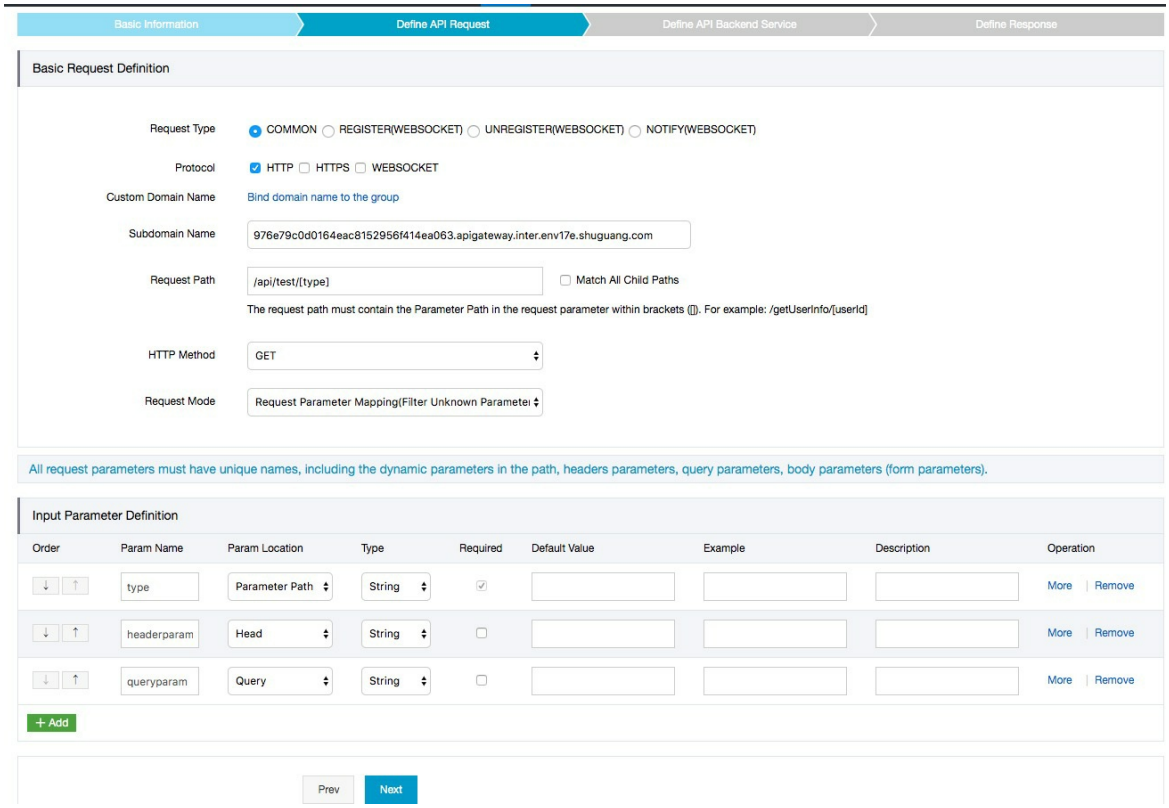
## Create an API

In the left-side navigation pane of the API Gateway console, choose **Publish APIs > APIs**. On the API List page, click **Create API** in the upper-right corner. On the Create API page, perform the following steps:

1. **Specify basic information**. In this step, specify basic information, including Group, API Name, Security Certification, and Description. In this example, set Group to testAppkeyGroup, Security Certification to Alibaba Cloud APP, and AppCode Certification to Disable AppCode authentication, configure other parameters as required, and then click Next.

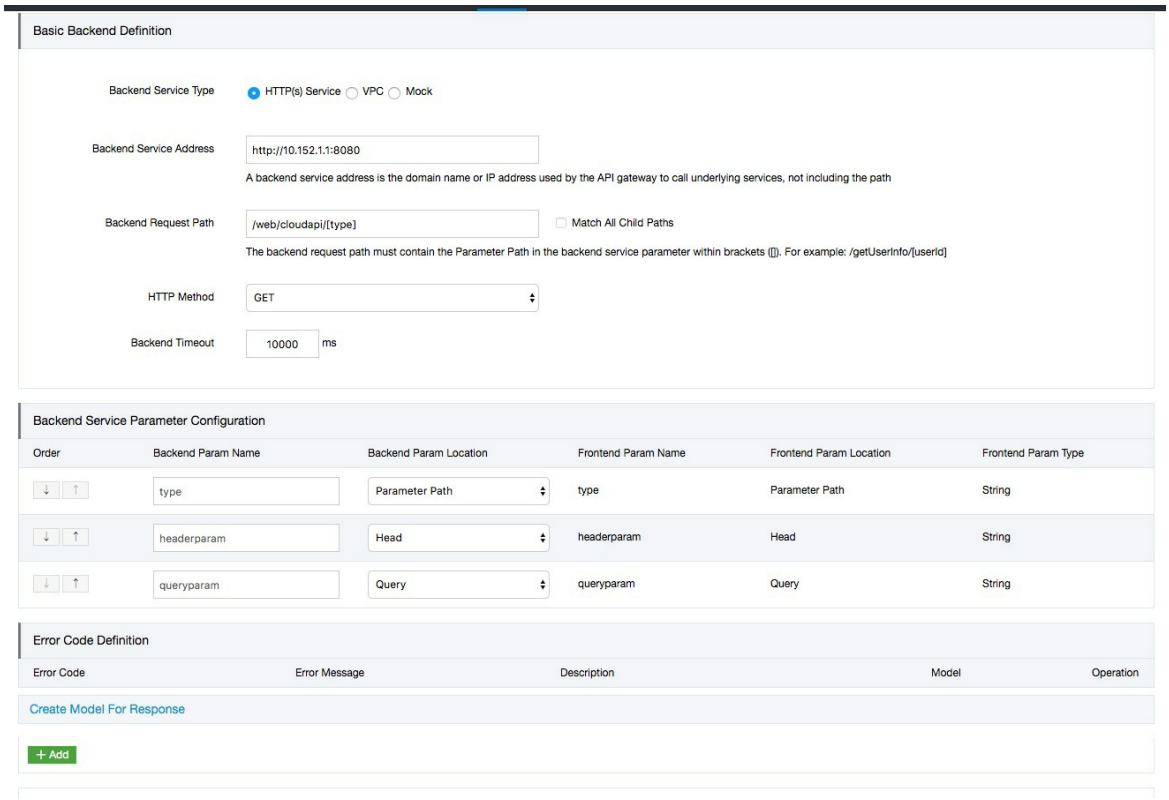



2. **Define an API request**. In this step, define how a client, such as a browser, a mobile app, or a business system, sends a request for the API. The parameters that need to be specified in this step include Request Type, Protocol, Request Path, HTTP Method, Request Mode, and those in the Input Parameter Definition section. Then, click Next. In this example, enter /web/cloudapi in the Request Path field and configure a path parameter, a query parameter, and a header parameter in the Input Parameter Definition section.

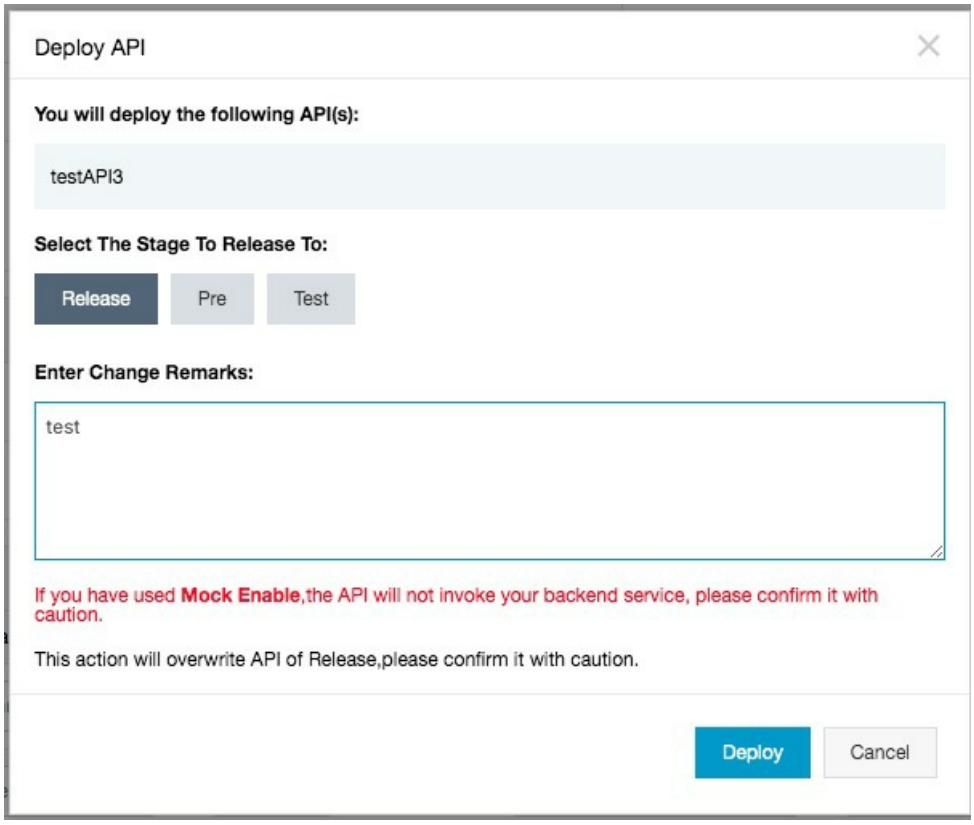



3. **Specify API backend service information**. In this step, configure a backend service type and a backend service address of the API and the mappings between request and response parameters. In this example, set Backend Service Type to HTTP(s) Service and Backend Service Address to an address that you can use to access API Gateway. For information about other backend service types, see API Gateway documentation. Configure other parameters such as Backend Request Path

as prompted and click Next.



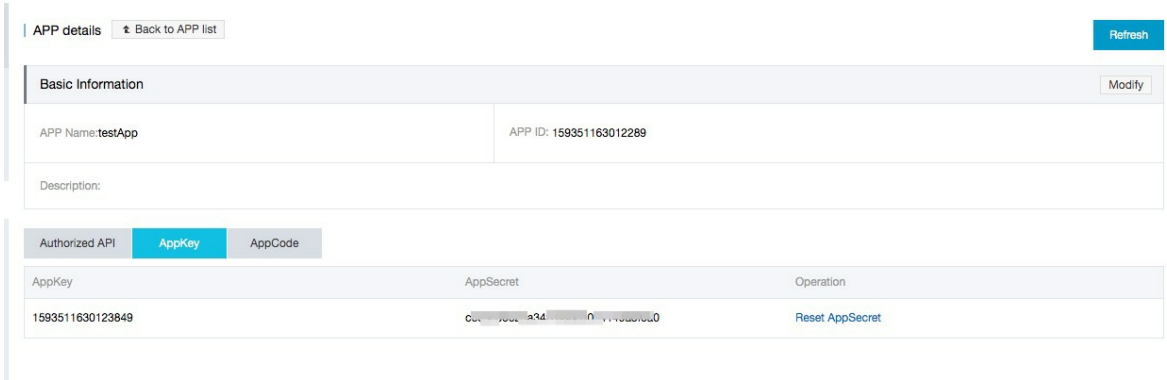

4. **Define return results**. In this step, configure response information to generate API documentation. The documentation helps API callers better understand APIs. You can configure parameters such as ContentType of Response, Sample of Returned Results, and Sample of Returned Failure. The configurations in this step are not involved in this example. Click **Create**.
5. **Publish the API**. API Gateway provides three environments to which you can publish an API: Release, Pre, and Test. All configurations you perform on an API can take effect only after you publish the API to a required environment. In this example, click Deploy in the message that indicates successful API creation. Alternatively, find the created API on the API List page and click Deploy in the Operation column. In the Deploy API dialog box, set Select The Stage To Release To to Release, specify Enter Change Remarks, and then click Deploy.




## Create an app and an API authorization

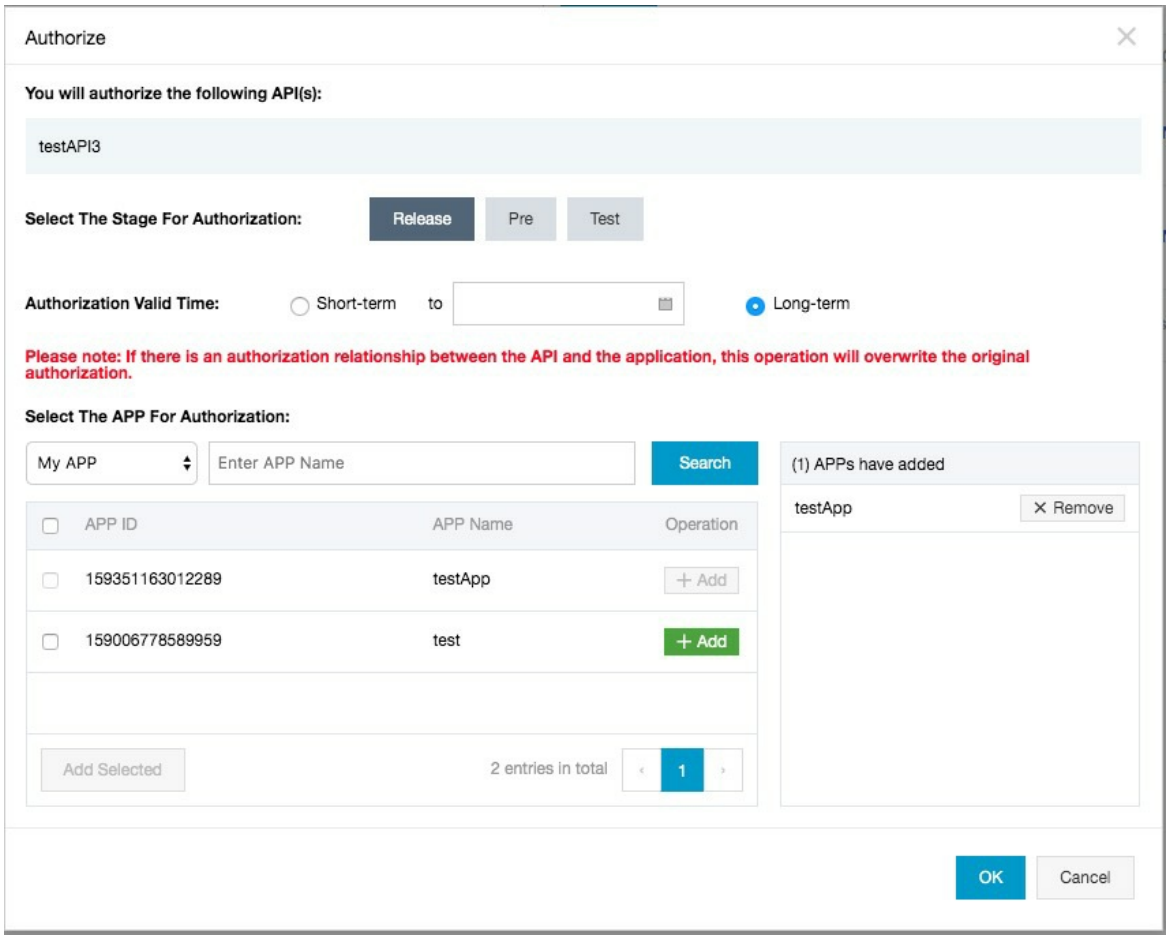
Apps are the identities that you use to call APIs. In Step 1 of the "Create an API" section, Security Certification is set to Alibaba Cloud APP. Therefore, after you publish the API, you must create an app and authorize the app to call the API.

1. **Create an app**: In the left-side navigation pane of the API Gateway console, choose **Consume APIs > APPs**. On the APP List page, click Create APP in the upper-right corner to create an app. Then, click the name of the created app to go to the APP details page, as shown in the following figure. Two authentication modes are provided: an AppKey and AppSecret pair and AppCode. Each app has an AppKey and AppSecret pair. It works in a way similar to an account and password pair. When you call an API, you must pass in the AppKey as an input parameter. AppSecret is used to calculate the signature string. API Gateway authenticates the key pair to verify your identity.




2. **Authorize an API**: On the **API List** page, find the created API and click **Authorize** in the Operation column. In the dialog box that appears, set Select The Stage For Authorization to the environment in which you published the API. In this example, set this parameter to Release. Click **Search** to search for the created app and click **+Add** in the Operation column to add this app to

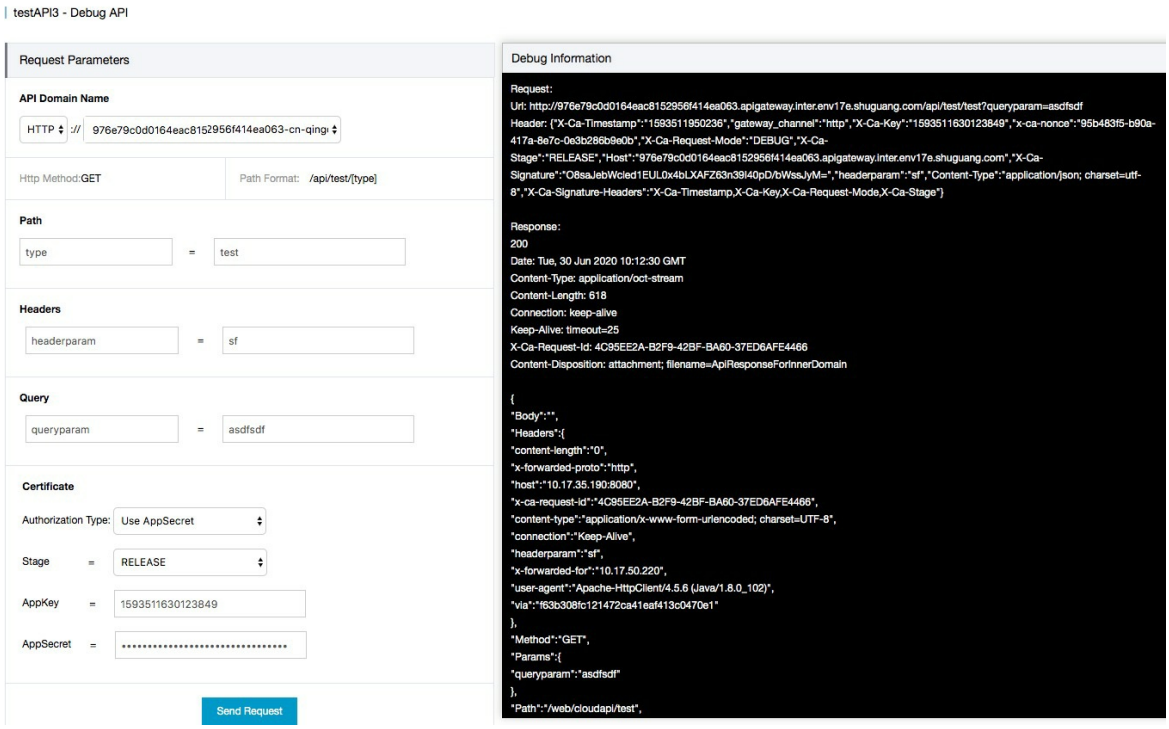
the pane on the right. Then, click **OK**. A success message appears.




## Debug an API

API Gateway supports online debugging. We recommend that you use this feature to check whether an API is correctly configured before you call this API at the client side.

1. In the left-side navigation pane of the API Gateway console, choose **Consume APIs > APPs**. On the APP List page, click the name of the app that has been authorized to call the created API. On the APP details page, click **Authorized API**, find the desired API, and then click **Debug API** in the Operation column. On the API debugging page, if you have defined input parameters for this API, you can enter different values for the input parameters to check whether the API is correctly configured.




Only the APIs that are published to a required environment and can be called by authorized apps are displayed after you click **Authorized API**.

## Call an API

After you debug and publish an API to a Release environment, you can use SDKs for API Gateway to call the API in your business system.

1. In the left-side navigation pane of the API Gateway console, choose **Consume APIs > Authorized APIs SDK**. On the Authorized APIs SDK Auto-Generation page, find the desired app and click the required programming language in the Authorized APIs SDK Auto-Generation column to download the related SDK package. The SDK package contains the API documentation and the SDK for the created API. For information about how to use the SDK, see the Readme file in the SDK package.

   Only the SDKs for APIs that are published to a Release environment are supported.

# 2.1.4. Call an API

# 2.1.4.1. Manage applications

# 2.1.4.1.1. Create an app

Apps are the identities that you use to call APIs. You can own multiple apps. Your apps can be authorized to call different APIs based on your business requirements. User accounts cannot be authorized to call APIs. In the API Gateway console, you can create, modify, or delete apps, view the details of apps, manage key pairs, and view the APIs that can be called by authorized apps.

Each app has an **AppKey** and **AppSecret** pair. It works in a way similar to an account and password pair. When you call an API, you must pass in the **AppKey** as an input parameter. **AppSecret** is used to calculate the signature string. API Gateway authenticates the key pair to verify your identity. An app must be authorized to call an API. Both authorization and authentication are intended for apps.

You can create apps on the **APP List** page in the API Gateway console.

### Procedure

1. Log on to the API Gateway console.
2. In the left-side navigation pane, click **Consume APIs** and then **APPs**.
3. On the APP List page, click **Create APP** in the upper-right corner.

   The app name must be globally unique. It must be 4 to 26 characters in length and can contain only letters, digits, and underscores (_). It must start with a letter.

   After an app is created, the system automatically assigns an **AppKey** and **AppSecret** pair to the app. You must use the **AppSecret** to calculate a signature string. When you call an API, you must include the signature string in the request. API Gateway verifies your identity based on the signature string.

   On the **APP List** page, click the app name to go to the APP details page that displays the **AppKey** and **AppSecret** information. If the key pair is missing, you can reset it.

## 2.1.4.1.2. View app details

You can view details of created apps.

### Procedure

1. Log on to the API Gateway console.
2. In the left-side navigation pane, click **Consume APIs** and then **APPs**.
3. On the APP List page, click the name of the app that you want to view.

   On the APP details page, you can view basic app information. You can also click **AppKey** or **Authorized API** to view key pair information and APIs that can be called by authorized apps.

## 2.1.4.1.3. Edit an app

You can edit a created app.

### Procedure

1. Log on to the API Gateway console.
2. In the left-side navigation pane, click **Consume APIs** and then **APPs**.
3. On the APP List page, find the target app and click **Edit** in the Operation column.
4. In the Modify APP dialog box, modify app information and click **OK**.

## 2.1.4.1.4. Delete an app

You can delete a created app.

### Procedure

1. Log on to the API Gateway console.
2. In the left-side navigation pane, click **Consume APIs** and then **APPs**.
3. On the APP List page, find the target app and click **Delete** in the Operation column.
4. In the Confirm Deletion message, click **OK**.

# 2.1.4.2. View created APIs

You can view created APIs in the API Gateway console.

## Procedure

1. Log on to the API Gateway console.
2. In the left-side navigation pane, choose **Publish APIs > APIs**.

# 2.1.4.3. Authorize an application

Authorization is the process of authorizing an application to call an API. Your applications must be authorized before they can call APIs.

You must provide your application IDs to the API provider for authorization. After authorization, you can view the APIs that your applications have been authorized to call in the API Gateway console.

The APIs that your applications have been authorized to call are displayed in the **Callable APIs** section on the application details page.

After the API provider authorizes your applications to call APIs, you do not need to and cannot authorize your applications.

# 2.1.4.4. Encrypt a signature

When you call an API, you must construct a signature string and add the calculated signature string to the request header. API Gateway uses symmetric encryption to verify the identity of the request sender.

- Add the calculated signature string to the request header.
- Organize the request parameters into a string-to-sign based on Request signatures. Then, use the algorithm provided in the SDK sample to calculate the signature. The result is the calculated signature string.
- Both HTTP and HTTPS requests must be signed.

For more information about how to construct a string-to-sign, see Request signatures. Replace the AppKey and AppSecret in the SDK sample with your own AppKey and AppSecret. Then, construct a string-to-sign based on Request signatures. After creating the string-to-sign, you can use it to initiate a request.

# 2.1.4.5. Request signatures

## Endpoint

- Each API belongs to an API group, and each API group has a unique endpoint. An endpoint is an independent domain name that is bound to an API group by the API provider. API Gateway uses endpoints to locate API groups.

- An endpoint must be in the *www.[Independent domain name].com/[Path]?[HTTPMethod]* format.
- API Gateway locates a unique API group by endpoint, and locates a unique API in the group through the combination of Path and HTTPMethod.
- After you purchase an API, you can obtain the API documentation from the **Purchased APIs** list in the API Gateway console. If you have not purchased an API, you must obtain authorization from the API provider for your applications to call the API. After authorization, you can obtain the API documentation from the **Callable APIs** list on the application details page.

## System-level header parameters

- (Required) X-Ca-Key: AppKey.
- (Required) X-Ca-Signature: the signature string.
- (Optional) X-Ca-Timestamp: the timestamp passed in by the API caller. This value is a UNIX timestamp representing the number of milliseconds that have elapsed since January 1, 1970 00:00:00 UTC. The timestamp is valid for 15 minutes by default.
- (Optional) X-Ca-Nonce: the UUID generated by the API caller. To prevent replay attacks, you must specify both the X-Ca-Nonce header and the X-Ca-Timestamp header.
- (Optional) Content-MD5: When the request body is not a form, you can calculate the MD5 value of the request body. Then, you can send the value to API Gateway for MD5 verification.
- (Optional) X-Ca-Stage: the stage of the API. Valid values: TEST, PRE, and RELEASE. Default value: RELEASE. If the API that you intend to call has not been published to the release environment, you must specify the value of this parameter. Otherwise, a URL error will be reported.

## Signature validation

### Construct the signature calculation strings

```
String stringToSign=
HTTPMethod + "\n" +
Accept + "\n" +                // We recommend that you specify the Accept header in the re
quest. If the request header is not set, some HTTP clients will use the default value */*,
causing signature verification to fail.
Content-MD5 + "\n"
Content-Type + "\n" +
Date + "\n" + ,
Headers +
Url
```

An HTTP method must be uppercase, such as POST.

If Accept, Content-MD5, Content-Type, and Date are empty, add a line break `\n` after each of them. If Headers is empty, `\n` is not required.

### Content-MD5

Content-MD5 indicates the MD5 value of the request body. The value is calculated as follows:

```
String content-MD5 = Base64.encodeBase64(MD5(bodyStream.getbytes("UTF-8")));
```

bodyStream indicates a byte array.

**Headers**

Headers indicates the string constructed by the keys and values of the header parameters that are used for Headers signature calculation. We recommend that you use the parameters starting with X-Ca and custom header parameters for signature calculation.

> **Notice**
>
> The following parameters are not used for Headers signature calculation: X-Ca-Signature, X-Ca-Signature-Headers, Accept, Content-MD5, Content-Type, and Date.

Headers construction method:

Sort the header keys used for Headers signature calculation **in alphabetical order**. Construct the string based on the following rules: If the value of a header parameter is empty, use `HeaderKey + ":" + "\n"` for signature calculation. The key and colon (:) must be retained.

```
String headers =
HeaderKey1 + ":" + HeaderValue1 + "\n"\+
HeaderKey2 + ":" + HeaderValue2 + "\n"\+
...
HeaderKeyN + ":" + HeaderValueN + "\n"
```

The keys of the header parameters used for Headers signature calculation must be separated with commas (,), and placed in the request headers. The key is X-Ca-Signature-Headers.

**Url**

Url indicates the **Form** parameter in **Path + Query + Body**. For **Query + Form**, sort keys specified by **Key** in alphabetical order and construct the string based on the following rules: If **Query** or **Form** is empty, no question marks `?` are required for `Url = Path`. If **Value** of a parameter is empty, only **Key** is used for signature calculation and an equal sign (=) is not required.

```
String url =
Path +
"?" +
Key1 + "=" + Value1 +
"&" + Key2 + "=" + Value2 +
...
"&" + KeyN + "=" + ValueN
```

> **Notice**
>
> Note: The **Query** parameter or the **Form** parameter may have multiple values specified by **Value**. If both parameters have multiple values, only the first value of each parameter is used for signature calculation.

## Signature calculation

```
Mac hmacSha256 = Mac.getInstance("HmacSHA256");
byte[] keyBytes = secret.getBytes("UTF-8");
hmacSha256.init(new SecretKeySpec(keyBytes, 0, keyBytes.length, "HmacSHA256"));
String sign = new String(Base64.encodeBase64(Sha256.doFinal(stringToSign.getBytes("UTF-8"))
,"UTF-8"));
```

`secret` indicates the AppSecret.

### Signature passing

Add the calculated signature to the request header. The key is X-Ca-Signature.

### Signature troubleshooting

If signature verification fails, API Gateway places the returned stringToSign value in the HTTP response header and sends the response to the client. The key is X-Ca-Error-Message. Compare the stringToSign value calculated by the client with the one returned by the server.

If the stringToSign values from the client and server are the same, check the AppSecret used for signature calculation.

HTTP headers do not support line breaks. Line breaks in stringToSign values are filtered out. Ignore the line breaks when you make a comparison.

### Signature demo

For detailed demo (Java) of signature calculation, please refer to the API Gateway console.

## 2.1.4.6. API call examples

You can edit an HTTP or HTTPS request to call an API. The API Gateway console provides API call examples of multiple programming languages for you to test the call.

### Part 1: Request

Request URL

When you call an API over an internal network, the second-level domain of the API group to which this API belongs is used by default. To view a second-level domain, choose **Publish APIs > API Groups** in the left-side navigation pane of the API Gateway console. Click the name of the target group to go to the Group Details page. If this group is bound with an independent domain, you can use this independent domain to initiate an access request.

```
http://e710888d3ccb4638a723ff8d03837095-cn-qingdao.aliapi.com/demo/post
```

Request method

`POST`

Request body

```
FormParam1=FormParamValue1&FormParam2=FormParamValue2   //HTTP request body
```

Request header

```
Host: e710888d3ccb4638a723ff8d03837095-cn-qingdao.aliapi.com
Date: Mon, 22 Aug 2016 11:21:04 GMT
User-Agent: Apache-HttpClient/4.1.2 (java 1.6)
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
// The request body type. Set the request body type based on the actual request you want to make.
Accept: application/json
// The response body type. Some APIs can return data in the appropriate format based on the specified response type. We recommend that you manually specify the request header. If the request header is not specified, some HTTP clients will use the default value */*, which causes a signature error.
X-Ca-Request-Mode: debug
// Specifies whether to enable the debug mode. This parameter is not case-sensitive. If it is not specified, the debug mode is disabled. Enable this mode in the API debugging phase.
X-Ca-Version: 1
// The API version number. Currently, all APIs support only version 1. You can leave this request header unspecified. The default version number is 1.
X-Ca-Signature-Headers: X-Ca-Request-Mode,X-Ca-Version,X-Ca-Stage,X-Ca-Key,X-Ca-Timestamp
// The custom request headers involved in signature calculation. The server reads the request headers based on this configuration to sign the request. This configuration does not include the Content-Type, Accept, Content-MD5, and Date request headers, which are already included in the basic signature structure. For more information about the signature, see Request signatures.
X-Ca-Stage: RELEASE
// The stage of the API. Valid values: TEST, PRE, and RELEASE. This parameter is not case-sensitive. The API provider can select the stage to which the API is published. The API can be called only after it is published to the specified stage. Otherwise, the system will prompt that the API cannot be found or that the request URL is invalid.
X-Ca-Key: 60022326
// The AppKey of the request. You must obtain the AppKey in the API Gateway console. Apps can call APIs only after they have been authorized.
X-Ca-Timestamp: 1471864864235
// The request timestamp. This value is a UNIX timestamp that represents the number of milliseconds that have elapsed since January 1, 1970 00:00:00 UTC. The timestamp is valid for 15 minutes by default.
X-Ca-Nonce:b931bc77-645a-4299-b24b-f3669be577ac
// The unique ID of the request. AppKey, API, and Nonce must be unique within the last 15 minutes. To prevent replay attacks, you must specify both the X-Ca-Nonce header and the X-Ca-Timestamp header.
X-Ca-Signature: FJleSrCYPGCU7dMlLTG+UD3Bc5Elh3TV3CWHtSKh1Ys=
// The request signature.
CustomHeader: CustomHeaderValue
// The custom request headers. CustomHeaderValue is used as an example. You can configure multiple custom request headers in requests based on the definition of the API that is being called.
```

## Part 2: Response

Status code

```
400  // The status code of the response. If the value is greater than or equal to 200 but l
ess than 300, the call succeeded. If the value is greater than or equal to 400 but less tha
n 500, a client-side error has occurred. If the value is greater than 500, a server-side er
ror has occurred.
```

Response header

```
X-Ca-Request-Id: 7AD052CB-EE8B-4DFD-BBAF-EFB340E0A5AF
// The unique ID of the request. When API Gateway receives a request, it generates a reques
t ID and returns the request ID to the client in the X-Ca-Request-Id header. We recommend t
hat you record the request ID in both the client and backend server for troubleshooting and
tracking.
X-Ca-Error-Message: Invalid Url
// The error message returned by API Gateway. If a request fails, API Gateway returns the e
rror message to the client in the X-Ca-Error-Message header.
X-Ca-Debug-Info: {"ServiceLatency":0,"TotalLatency":2}
// The message returned only when the debug mode is enabled. The message is used only for r
eference at the debugging stage.
```

Regardless of whether you call an API by using HTTP or HTTPS, the request must include the signature information. For information about how to calculate and deliver an encrypted signature, see Request signatures.

# 2.1.5. APIs

## 2.1.5.1. Manage groups

### 2.1.5.1.1. Create an API group

You can create an API group in the API Gateway console.

#### Procedure

1. Log on to the API Gateway console.
2. In the left-side navigation pane, choose **Publish APIs > API Groups**.
3. On the Group List page, click **Create Group** in the upper-right corner.
4. On the Create Group page, specify **Organization**, **Resource Set**, and **Region** in the Region section. Then specify **Name** and **Description** in the Basic Settings section and click **Submit**.

   The group name must be unique. It must be 4 to 50 characters in length and can contain only letters, digits, and underscores (_). It must start with a letter.

### 2.1.5.1.2. Manage domain names

In Apsara Stack, you can use the second-level domain of a group to directly call an API that belongs to this group. You can also bind your domain name to the group so that you can use your domain name to call APIs that belong to the group.

#### Context

If you want to use your domain name to directly call APIs that belong to a group, you must bind the domain name to the group and add a DNS record to your domain name. The domain name must be resolved to the second-level domain of the group or the IP address that corresponds to the second-level domain.

### Bind an independent domain

1. Log on to the API Gateway console.
2. In the left-side navigation pane, choose **Publish APIs > API Groups**.
3. On the Group List page, find the target group and click **Bind Domain** in the **Operation** column.
4. In the Bind Domain Name dialog box, specify **Domain Name** and click **OK**.

### Delete an independent domain

1. Log on to the API Gateway console.
2. In the left-side navigation pane, choose **Publish APIs > API Groups**.
3. On the **Group List** page, find the target group and click its name to go to the **Group Details** page.
4. In the **Custom Domain Name** section, click **Delete Domain** in the Operation column.
5. In the Confirm Deletion message, click **OK**.

## 2.1.5.1.3. Manage certificates

To use HTTPS on an independent domain, you must upload an SSL certificate.

### Context

To perform HTTPS API calls, you must use a domain name that supports HTTPS and set Protocol to HTTPS in the Basic Request Definition section when you define an API request.

### Upload a certificate

1. Log on to the API Gateway console.
2. In the left-side navigation pane, choose **Publish APIs > API Groups**.
3. On the **Group List** page, click the name of the target group to go to the **Group Details** page.
4. In the **Custom Domain Name** section, click **Create Certificate** in the **SSL Certificate** column.
5. In the **Create Certificate** dialog box, specify **Certificate Name**, **Certificate Content**, and **Private Key**, and click **OK**.

### Delete a certificate

1. Log on to the API Gateway console.
2. In the left-side navigation pane, choose **Publish APIs > API Groups**.
3. On the **Group List** page, click the name of the target group to go to the **Group Details** page.
4. In the **Custom Domain Name** section, click **Delete Certificate** in the **Operation** column.
5. In the Confirm Deletion message, click **OK**.

## 2.1.5.1.4. Delete an API group

You can delete a created API group.

## Procedure

1. Log on to the API Gateway console.
2. In the left-side navigation pane, choose **Publish APIs > API Groups**.
3. On the Group List page, find the target group and click **Delete** in the Operation column.
4. In the Delete Group message, click **Delete**.

> **Note** Before you delete a group, you must delete APIs that belong to this group.

# 2.1.5.1.5. Manage environments

To understand environment management, you must be familiar with two concepts: environment and environment variable.

- An **environment** is an API group configuration. You can configure several environments for a group. APIs that have not been published are considered defined APIs. An API can provide external services only after it is published to an environment.
- **Environment variables** are environment-specific variables that you can create and manage. For example, you can create an environment variable named `Path` in the Release environment. The value of this variable is `/stage/release`.

When you define an API, you can add variables, in the format of #Variable name#, to Path values. For example, specify `Path` in the format of `#Path#` when you define an API.

When you publish the API to the Release environment, the value of `#Path#` is `/stage/release`.

When you publish the API to another environment that does not have the environment variable `#Path#`, the variable value cannot be obtained and the API cannot be called.

Environment variables allow backend services to run in different runtime environments. You can access various backend services by configuring the same API definition but different backend service endpoints and paths across different environments. When you use environment variables, consider the following limits:

- Variable names are case-sensitive.
- If you configure a variable in the API definition, you must configure the name and value of the variable for the environment to which the API is published. Otherwise, no value is assigned to the variable and the API cannot be called.

## Create an environment variable

1. Log on to the API Gateway console.
2. In the left-side navigation pane, choose **Publish APIs > API Groups**.
3. On the Group List page, find the desired group and click **View Stages** in the Operation column.
4. On the Stage Management page, click **Add Variable** in the upper-right corner. In the Add Variable dialog box, specify *Name* and *Value* and click **Add**.

> **Notice**
>
> The variable names for the Release, Pre, and Test environments must be the same. However, the variable values for the three environments can be different. After an API is published to a specified environment, the variable value will be automatically replaced.

### Delete an environment variable

1. Log on to the API Gateway console.
2. In the left-side navigation pane, choose **Publish APIs > API Groups**.
3. On the Group List page, find the desired group and click **View Stages** in the Operation column.
4. On the Stage Management page, select the runtime environment, find the desired variable, and then click **Delete** in the Operation column.
5. In the Confirm Deletion message, click **OK**.

# 2.1.5.2. Create an API

# 2.1.5.2.1. Overview

Creating an API is the process of defining the API in the API Gateway console. When creating an API, you must define the basic information, back-end service information, API request information, and response information of the API.

- API Gateway enables you to configure verification rules for input parameters. API Gateway can be configured to pre-verify and forward API requests that contain valid parameters.
- API Gateway enables you to configure mappings between front-end and back-end parameters. API Gateway can map a front-end parameter at one location to a back-end parameter at a different location. For example, you can configure API Gateway to map a `Query` parameter in an API request to a `Header` parameter in a back-end service request. In this way, you can encapsulate your back-end services into standard API operations.
- API Gateway enables you to configure constant and system parameters. These parameters are not visible to your users. API Gateway can add these parameters to requests based on your business requirements before sending the requests to your back-end services. If you want API Gateway to attach the keyword `apigateway` to each request that API Gateway forwards to your back-end services, you can configure `apigateway` as a constant parameter and specify where it is received.

# 2.1.5.2.2. Create an API

### Procedure

1. Log on to the API Gateway console.
2. In the left-side navigation pane, choose **Publish APIs > APIs**.
3. On the API List page, click **Create API** in the upper-right corner.
4. Specify basic information of the API and click **Next**.

| Parameter | Description |
| --- | --- |
| Group | The basic management unit of APIs. Before you create an API, you must create an API group. When you select a group, a region is selected for the API. |
| API Name | The name of the API to be created. |
| Security Certification | The authentication mode of API requests. Valid values: *Alibaba Cloud APP* and *No Certification*.<br>○ *Alibaba Cloud APP*: This mode requires the requester to pass the app authentication to call an API.<br>○ *No Certification*: This mode allows all users who know the request definition of an API to initiate a request. API Gateway directly forwards the request to your backend service without the need to verify the identity of a requester. |
| Signature Method | The algorithm that is used to sign API requests. Valid values:<br>○ HmacSHA256<br>○ HmacSHA1 and HmacSHA256: If you set this parameter to this value, both the algorithms are supported. |
| Description | The description of the API. |

5. Define an API request. In this step, define how users call your API, with the following parameters specified: Request Type, Protocol, Request Path, HTTP Method, and Request Mode.

| Parameter | Description |
| --- | --- |
| Request Type | The request type. Only the COMMON request type is supported. Valid values: COMMON, REGISTER(WEBSOCKET), UNREGISTER(WEBSOCKET), and NOTIFY(WEBSOCKET).<br>○ *COMMON*: indicates common HTTP or HTTPS requests.<br>○ *REGISTER(WEBSOCKET)*: indicates the bidirectional control signaling to register devices. It is sent from the client to the server.<br>○ *UNREGISTER(WEBSOCKET)*: indicates the bidirectional control signaling to deregister devices. It is sent from the client to the server. After devices are deregistered, server-to-client notifications are no longer received.<br>○ *NOTIFY(WEBSOCKET)*: After the backend service receives the registration signaling sent from the client, the backend service records the device ID and sends a server-to-client notification to API Gateway. Then, API Gateway sends the notification to the device. If the device is online, API Gateway sends the server-to-client notification to the device. |
| Protocol | The supported protocol. Valid values: *HTTP*, *HTTPS*, and *WEBSOCKET*. |

| Parameter | Description |
| --- | --- |
| Request Path | The API request path that corresponds to the service host. The request path can be different from the actual backend service path. You must specify a valid and semantically accurate path as the request path. You can configure dynamic parameters in the request path. This requires that you specify path parameters in the request. In addition, the path parameters can be mapped to query and header parameters that are received by the backend service. |
| HTTP Method | The HTTP request method. Valid values: *PUT*, *GET*, *POST*, *PATCH*, *DELETE*, and *HEAD*. |
| Request Mode | The request mode. Valid values: *Request Parameter Mapping(Filter Unknown Parameters)*, Request Parameter Mapping(Passthrough Unknown Parameters), and *Request Parameter Passthrough*.<br>○ Request Parameter Mapping(Filter Unknown Parameters): You must configure request and response data mappings for query, path, and body form parameters. API Gateway transparently passes only the configured parameters to the backend service. Other parameters are filtered out.<br>○ Request Parameter Mapping(Passthrough Unknown Parameters): API Gateway maps and verifies only configured request parameters and transparently passes unknown parameters in a request to the backend service.<br>○ Request Parameter Passthrough: You do not need to configure query and body form parameters, but must configure path parameters in the Input Parameter Definition section. All parameters sent from the client are transparently passed by API Gateway to the backend service. |

6. Define request parameters.

   In this step, define the request parameters of your API. You can specify different request parameters for different parameter paths. You can select Head, Query, Body, or Parameter Path from the Param Location drop-down list. When you configure a dynamic path parameter, you must provide a description of this dynamic parameter in the Input Parameter Definition section. The following data types are supported: String, Int, and Boolean.

   - The names of all parameters must be unique.
   - You can use the shortcut keys in the Order column to adjust the parameter order.
   - To delete a parameter that is no longer required, you can click **Remove** in the Operation column that corresponds to the parameter.

7. Configure parameter verification rules.

   To configure verification rules of a parameter, you can click **More** in the Operation column that corresponds to the parameter. For example, you can specify Max Length and Enumeration. API Gateway pre-verifies requests based on the verification rules. Requests with invalid parameters are not sent to your backend service. This significantly reduces the workload on your backend service.

8. Configure the backend service and click **Next**.

   In this step, define mappings between request and response parameters, and specify the API configurations of your backend service. Backend service configurations include Backend Service Address, Backend Request Path, Backend Timeout, and configurations in the Backend Service Parameter Configuration, Constant Parameter, and System Parameter sections. After API Gateway

receives a request, it converts the format of the request into the format that is required by your backend service based on the backend service configuration. Then, API Gateway forwards the request to your backend service.

> **Note** You can configure the following parameters: dynamic path parameters, header parameters, query parameters, body parameters (non-binary), constant parameters, and system parameters. Each parameter name must be globally unique. For example, you cannot specify a header parameter and a query parameter that have the same name.

i. Specify related parameters in the Basic Backend Definition section.

| Parameter | Description |
| --- | --- |
| Backend Service Type | ■ HTTP(s) Service: This option is selected by default. It indicates that API Gateway accesses the backend service over HTTP or HTTPS. If API Gateway can directly communicate with the backend service, select this option.<br>■ VPC: If the backend service is deployed in a virtual private cloud (VPC), select this option.<br>■ Mock: If you want to simulate expected return results, select this option. |
| VPC ID | The ID of the VPC where your backend service is deployed. This parameter is required when Backend Service Type is set to VPC. |
| Backend Service Address | The host of the backend service.<br>■ If Backend Service Type is HTTP(s) Service, set this parameter to a domain name or a value in the `http(s)://host:port` format.<br>■ If Backend Service Type is VPC, set this parameter to a value in the `http://ip:port` format. |
| Backend Request Path | The actual request path of your API on your backend server. If you want to receive dynamic parameters in the backend path, you must specify the locations and names of the corresponding request parameters to declare parameter mappings. |
| HTTP Method | The HTTP request method. Valid values: *PUT*, *GET*, *POST*, *PATCH*, *DELETE*, and *HEAD*. |
| Backend Timeout | The response time for API Gateway to access the backend service after API Gateway receives an API request. The response time starts from the time when API Gateway sends an API request to the backend service and ends at the time when API Gateway receives a response returned by the backend service. The response time cannot exceed 30s. If API Gateway does not receive a response from the backend service within 30s, API Gateway stops accessing the backend service and returns an error message. |

ii. Configure parameters in the Backend Service Parameter Configuration section.

API Gateway can set up mappings between request and response parameters, including name mappings and parameter location mappings. API Gateway can map a path, header, query, or body request parameter to a response parameter at a different location. This way, you can package your backend service into a standardized and professional API form. This part declares the mappings between request and response parameters.

> **Note** The request and response parameters must be globally unique.

iii. Configure constant parameters in the Constant Parameter section.

If you want API Gateway to attach the apigateway tag to each request that API Gateway forwards to your backend service, you can configure this tag as a constant parameter. Constant parameters are not visible to your users. After API Gateway receives requests, it automatically adds constant parameters to the specified locations and then forwards the requests to your backend service.

iv. Configure system parameters in the System Parameter section.

By default, API Gateway does not send its system parameters to your backend service. If you require the system parameters, you can configure the related locations and names. The following table lists the system parameters.

| Parameter | Description |
| --- | --- |
| CaClientIp | The IP address of the client that sends a request. |
| CaDomain | The domain name from which a request is sent. |
| CaRequestHandleTime | The time when a request is sent. It must be in GMT. |
| CaAppId | The ID of the app that sends a request. |
| CaRequestId | The unique ID of the request. |
| CaApiName | The name of the API. |
| CaHttpSchema | The protocol that is used to call an API. The protocol can be HTTP or HTTPS. |
| CaProxy | The proxy (AliCloudApiGateway). |

9. Define responses and click **Create**.

In this step, specify ContentType of Response, Sample of Returned Results, and Sample of Returned Failure, and add configurations in the Error Code Definition section. API Gateway does not parse responses, but forwards the responses to API requesters.

# 2.1.5.2.3. Security authentication

The security authentication methods that are supported by API Gateway include Alibaba Cloud applications and none.

- Alibaba cloud applications: An application must be authorized by the API provider to call an API. An API caller must provide an AppKey and encrypted signature. Otherwise, the API request validation will fail. For more information about the signature method, see Encrypt a signature.
- None: The API can be called without authorization after it is published. The AppKey and encrypted signature are not required when you make an API request.

# 2.1.5.2.4. Configure a network protocol

HTTPS domain names are not supported in the API Gateway console. To use an HTTPS domain name, you can call the API operations of API Gateway.

To configure a network protocol, perform the following operations: Find the target API on the API List page in the API Gateway console, and click Manage in the Operation column. On the **API Definition** page, click **Edit** in the upper-right corner. On the page that appears, specify Protocol in the **Define API Request** step.

Valid values of Protocol:

- HTTP
- HTTPS
- WEBSOCKET

# 2.1.5.2.5. Configure a request body

You can configure a request body when the HTTP method is POST, PUT, or PATCH. You can use the following methods to configure the request body. The methods are mutually exclusive.

- Form-based request body: Add a request parameter in the Input Parameter Definition section of the Define API Request step on the Create API page, and select Body from the Param Location drop-down list. The configured request body can only be used to transmit form data.
- Non-form-based request body: If the body content to be transmitted is in the JSON or XML format, select Non-Form data, such as JSON, Binary data in the Request Body section of the Define API Request step on the Create API page. The size of a request body cannot exceed 8 MB.

# 2.1.5.2.6. Configure an API in Mock mode

In most cases, business partners can work in combination to develop a project. The project development process is hindered due to the interdependence among business partners. Misunderstandings can also arise and affect the development progress or even cause severe delays to the project. The Mock mode is used to simulate the predetermined API responses in the project development process. This reduces misunderstandings and improves development efficiency.

API Gateway provides a simple configuration process of an API in Mock mode.

## Configure an API in Mock mode

Log on to the API Gateway console. In the left-side navigation pane, choose **Publish APIs > APIs**. On the API List page, find the target API and click **Manage** in the Operation column. On the API Definition page, click **Edit** in the upper-right corner.

On the page that appears, configure the Mock mode in the **Define API Backend Service** step.

1. Set **Backend Service Type** to **Mock**.

2. Specify Mock Result in the Mock Configuration section.

   Enter your responses as the Mock-based response body. Responses can be in the JSON, XML, or text format. Example:

   ```
    {
   "result": {
       "title": " Mock test for API Gateway",
   }
   }
   ```

   Save the settings and then publish the API to the Test or Release environment for testing.

3. Specify **HTTP Status Code** based on HTTP status code specifications. Enter 200 to indicate a successful API request.
4. Specify **Mock Header**. You can click +Add Item to add a Mock response header based on your business requirements.

# 2.1.5.2.7. Return the Content-Type header

The value of the Content-Type header is only used to generate API documentation. It does not affect responses returned by the back-end service. The Content-Type header is returned by the back-end service.

# 2.1.5.3. API management

# 2.1.5.3.1. View and modify an API

You can view and modify an API based on your business requirements.

> **Note**
>
> If you modify an API that is published, the modifications are not immediately applied. You must republish the modified API to synchronize the changes to the Release environment.

## Procedure

1. Log on to the API Gateway console.
2. In the left-side navigation pane, choose **Publish APIs > APIs**.
3. On the API List page, find the desired API.
   - Click **Manage** in the Operation column. On the API Definition page, you can view the information of the API.
   - Click **Edit** in the upper-right corner to edit the API based on your business requirements.

     The procedure to create an API is similar to that to modify an API. For more information about how to create an API, see Create an API. If you want to cancel modifications before they are submitted, click Cancel Edit in the upper-right corner of the edit page.

# 2.1.5.3.2. Publish an API

After you create an API, you must publish the API to the Test, Pre, or Release environment before it can be called.

- When you use a second-level or independent domain to access an API that is published to a specified environment, you must specify the environment in the request header.
- If you publish an API that already has a running version in the Test or Release environment, the running version is automatically overwritten by the new version within 15s. However, all historical versions and definitions are recorded. This allows you to roll the API back to an earlier version.
- You can unpublish an API in the Test or Release environment. The plug-in binding relationship or the app authorization relationship is retained after you unpublish an API. These relationships take effect again if the API is republished. You can also perform related operations to remove the authorization or unbind a required plug-in.

### Step 1: Publish an API

After you create an API, you can publish the API to the Test environment to test the API first.

API Gateway allows you to manage different versions of APIs in the Test or Release environment. You can publish or unpublish the API, and switch the version of the API. The version switch takes effect in real time.

1. Log on to the API Gateway console.
2. In the left-side navigation pane, choose **Publish APIs > APIs**.
3. On the API List page, find the desired API and click **Deploy** in the Operation column.
4. In the Deploy API dialog box, specify Enter Change Remarks and click **Deploy**.

### Step 2: Test the API

To simulate API requests, you can create an app and authorize the app to call your API.

You can compile code based on actual scenarios, or use the SDK samples provided by API Gateway to call your API.

You can publish the API to the Test or Release environment. If no independent domain is mapped to the group to which the API belongs, you can test or call the API by using a second-level domain. When you make an API request, set the X-Ca-Stage header to TEST, PRE, or RELEASE to specify the environment of the API. If you do not specify the header, the API will be invoked to the Release environment.

## 2.1.5.3.3. Authorize an app

You must authorize an app before it can call an API. After you publish an API to the Release environment, you must authorize apps to call the API. You can grant or revoke the authorization of an app to call an API. API Gateway verifies the authorization relationship.

> Note
>
> - You can authorize one or more apps to call one or more APIs.
> - If an API is published to both the Test and Release environments and an app is authorized to call the API in the Test environment, the app can call only the API in the Test environment.
> - You can find an app based on its ID.
> - If you want to revoke the authorization of an app to call an API, go to the Authorization page of the API. Then select the required app and click Revoke Authorization in the lower-left corner.

An app indicates the identity of a requester. Before testing or calling an API, you or your users must create an app that is used as the identity of a requester. Then, you must authorize the app to call the API.

> Note Authorizations are environment-specific. If you want to use an app to call an API in both the Test and Release environments, you must authorize the app in both environments. Otherwise, errors may occur due to the inconsistency between the authorized environment and the requested environment.

## Procedure

1. Log on to the API Gateway console.
2. In the left-side navigation pane, choose **Publish APIs > APIs**.
3. On the API List page, find the target API and click **Authorize** in the Operation column.
4. In the Authorize dialog box, specify Select The Stage For Authorization and Select The APP For Authorization.

   **My APP** is automatically selected from the drop-down list on the left. Click **Search**. Apps created under your account appear.

   If you want to authorize an app created under a different account, select **APP ID** from the drop-down list on the left, enter the app ID in the search bar, and click **Search**.

   To view the ID of an app, click **Consume APIs** and then **APPs** in the left-side navigation pane. On the APP List page, click the name of the target app to go to the APP details page.
5. Select an app to be authorized and click +Add in the Operation column to add this app to the right pane. Alternatively, you can select multiple apps to be authorized at a time and click Add Selected in the lower-left corner of the page to add these apps to the right pane.
6. Click **OK** to complete the authorization.
7. Click Manage in the Operation column that corresponds to the target API. On the API Definition page, click **Authorization** in the left-side navigation pane to view the authorized apps.

# 2.1.5.3.4. Revoke an authorization

You can revoke the authorization of an app to call an API.

## Procedure

1. Log on to the API Gateway console.

2. In the left-side navigation pane, choose **Publish APIs > APIs**.
3. On the API List page, click the name of the desired API for which you want to revoke the authorization. On the API Definition page, click **Authorization** in the left-side navigation pane.
4. Select desired apps and click **Revoke Authorization** in the lower-left corner.
5. In the Confirm authorization revocation message, click **OK**.

# 2.1.5.3.5. Unpublish an API

You can unpublish an API.

You can unpublish an API in the Test or Release environment. The binding or authorization relationships of policies, keys, and apps are retained after you unpublish an API. These relationships will take effect again if the API is republished. For more information about how to remove these relationships, see Revoke an authorization.

## Procedure

1. Log on to the API Gateway console.
2. In the left-side navigation pane, choose **Publish APIs > APIs**.
3. On the API List page, find the desired API and click **Undeploy** in the Operation column.
4. In the Undeploy API message, click **Undeploy**.

# 2.1.5.3.6. View the version history of an API

You can view the version history of an API, including the version number, description, environment, publish time, and specific definition of each version.

## Procedure

1. Log on to the API Gateway console.
2. In the left-side navigation pane, choose **Publish APIs > APIs**.
3. On the API List page, find the target API and click **Manage** in the Operation column to go to the API Definition page.
4. Click **Deployment History** in the left-side navigation pane. You can view the version history of this API.
5. On the Deployment History page, find the target version and click **View** in the Operation column.

# 2.1.5.3.7. Change the version of an API

When you view the version history of an API, you can select a different version to switch the API to this version. The selected version then replaces the previous version and takes effect in the specified environment.

## Procedure

1. Log on to the API Gateway console.
2. In the left-side navigation pane, choose **Publish APIs > APIs**.
3. On the API List page, find the target API and click **Manage** in the Operation column to go to the API

Definition page.

4. Click **Deployment History** in the left-side navigation pane.
5. Find the target version and click **Switch to this version** in the Operation column.
6. In the API Version Switch dialog box, enter the description and click **Switch**.

# 2.1.5.4. Plugin management

# 2.1.5.4.1. Use parameters and conditional expressions

In an `access control plugin`, `throttling plugin`, `backend routing plugin`, or `error code mapping plugin`, you can obtain parameters from requests, responses, and system context. Then, you can use conditional expressions to evaluate these parameters. This topic describes how to define parameters and write conditional expressions.

## 1. Define parameters

1. Definition method

   Before you use a conditional expression, you must explicitly define all the parameters required in this conditional expression in the parameters field. Example:

   ```
   ---
   parameters:
     method: "Method"
     appId: "System:CaAppId"
     action: "Query:action"
     userId: "Token:UserId"
   ```

   The parameters specified in `parameters` are key-value pairs of the string type.

   - `key` indicates the name of a variable to be used in a conditional expression. The name must be unique and must conform to the following regular expression: `[a-zA-Z_][a-zA-Z0-9]+`.
   - `value` indicates the location of a parameter. It is specified in the `{location}` or `{location}:{name}` format.
     - `location` indicates the location of a parameter. For more information, see the following table.
     - `name` indicates the name of a parameter, which is used to locate the parameter at a specific location. For example, `Query:q1` indicates the first value of the query string named q1.

2. Parameter locations

   Before you use a conditional expression, you must define the parameters that are required in this conditional expression. The following table describes parameters at specific locations that can be used by various plugins.

| Location | Included in | Description |
| --- | --- | --- |
| Method | Request | The HTTP request method, in uppercase, such as GET or POST. |
| Path | Request | The complete HTTP request path, such as `/path/to/query`. |
| StatusCode | Response | The HTTP status code in a backend response, such as 200 or 400. |
| ErrorCode | Response | Error codes. |
| Header | Request/Response | Use `Header:{Name}` to obtain the first value of the HTTP header that is specified by `{Name}`. |
| Query | Request | Use `Query:{Name}` to obtain the first value of the query string that is specified by `{Name}`. |
| Form | Request | Use `Form:{Name}` to obtain the first value of the form that is specified by `{Name}`. |
| Host | Request | Use `Host:{Name}` to obtain the template parameters of the matched wildcard domain names. |

| Location | Included in | Description |
| --- | --- | --- |
| Parameter | Request | Use `Parameter:{Name}` to obtain the first value of the custom API parameter that is specified by {Name}. |
| BodyJsonField | Response | Use `BodyJsonField:{JPath}` to obtain the JSON string in the body of an API request or a backend response in JSONPath mode. |
| System | Request/Response | Use `System:{Name}` to obtain the value of the system parameter that is specified by `{Name}`. |
| Token | Request/Response | If JWT is used for authentication, use `Token:{Name}` to obtain the value of the parameter that is specified by `{Name}` in a token. |

Rules for use:

- You can use the following plugins at the request phase: `access control plugin`, `throttling plugin`, and `backend routing plugin`. These plugins support only the parameters at the following locations: `Method`, `Path`, `Header`, `Query`, `Form`, `Parameter`, `System`, and `Token`.
- You can also use the `error code mapping plugin` at the response phase. This plugin supports only the parameters at the following locations: `StatusCode`, `ErrorCode`, `Header`, `BodyJsonField`, `System`, and `Token`.

- Parameters at the `Method`, `Path`, `StatusCode`, and `ErrorCode` locations are defined in the {location} format.
- If you use parameters at the `Header` location in a plugin at the request phase, headers from client requests are read. If you use these parameters at the response phase, headers from backend responses are read.
- Parameters at the `Parameter` location are available only for plugins at the request phase. A `frontend parameter`, instead of a `backend parameter`, is used to search for the parameter with the same name in the API definition. If no parameter with the same name exists, a null value is returned.
- A complete request path is returned from `Path`. If you require a parameter at the Path location, use the corresponding parameter at the `Parameter` location.
- Parameters at the `BodyJsonField` location are available only for the `error code mapping plugin`. Obtain the JSON string in the body of a backend response in `JSONPath` mode. For more information, see Usage notes of JSONPath.
- If `JWT` is used for authentication, use `Token:{CliamName}` to obtain the value of the parameter specified by `{CliamName}` in a token. For more information, see the plugin documentation.

3. Usage notes of JSONPath

   JSONPath is available only for the `error code mapping` plugin at the `BodyJsonField` location. It is used to extract the `JSON` string in the body of a backend response. For more information about JSONPath, see the JSONPath overview documentation.

   Example: When you use the expression `code:"BodyJsonField:$.result_code"`, you can obtain the value of `result_code` from the following body. `code:ok` is parsed from the following body.

   ```
   { "result_code": "ok", "message": ... }
   ```

4. System parameters

| Parameter | Description | Value |
| --- | --- | --- |
| CaClientIp | The IP address of the request client. | Example value: 37.78.3.3. |
| CaDomain | The full domain name in a request, with a Host header. | Example value: api.foo.com. |

| Parameter | Description | Value |
| --- | --- | --- |
| CaAppId | The ID of the application that sends the request. | Example value: 49382332. |
| CaAppKey | The key of the application that sends the request. | Example value: 12983883923. |
| CaRequestId | The unique ID of the request generated by API Gateway. | Example value: CCE4DEE6-26EF-46CB-B5EB-327A9FE20ED1. |
| CaApiName | The API name. | Example value: TestAPI. |
| CaHttpSchema | The protocol used by the client to call operations. | Valid values: http, https, and ws. |
| CaClientUa | The UserAgent header of the client. | Used to transparently pass values uploaded by the client. |
| CaStage | The running environment of API Gateway. | Valid values: TEST, PRE, and RELEASE. |

## 2. Write conditional expressions

You can use conditional expressions in plugins or other scenarios to evaluate parameters in a wide variety of scenarios.

1. Basic syntax
   - Conditional expressions are similar to SQL statements. Example: `$A > 100 and '$B = 'B'`.
   - An expression is in the following format: `{Parameter} {Operator} {Parameter}`. In the preceding example, you can specify a `variable` or a `constant` for `$A > 100`.
   - A `variable` starts with $ and references a parameter defined in the context. For example, `q1:"Query:q1"` is defined in `parameters`. You can use the variable `$q1` in your expression. The value of this variable is the value of the `q1` query parameter in the request.

- A `constant` can be a `string`, `number`, or `Boolean value`. Examples: `"Hello",' foo', 100, -1, 0.1, and true`. For more information, see Value types and evaluation rules.
- The following `operators` are supported:
  - `=` and `==`: equal to.
  - `<>` and `!=`: not equal to.
  - `>`, `>=`, `<`, and `<=`: comparison.
  - `like` and `!like`: check whether a specific string matches a specified pattern. The percent sign `%` is used as a wildcard in the evaluation. Example: `$Query like 'Prefix%'`.
  - `in_cidr` and `!in_cidr`: specify the mask of an IP address. Example: `$ClientIp in_cidr '47.89.0.0/24'`.
- You can use `null` to check whether a parameter is empty. Example: `$A == null` or `$A != null`.
- You can use the operators `and`, `or`, and `xor` to combine different expressions in a right-to-left order by default.
- You can use parentheses `()` to specify the priority of conditional expressions.
- You can use `!()` to perform the logical negation operation on the enclosed expression. For example, the result of `!(1=1)` is false.
- The following built-in functions are used for evaluation in some special scenarios:
  - `Random()`: generates a parameter of the floating-point number type. The parameter value ranges from 0 to 1. This parameter is used in scenarios where random input is required, such as blue-green release.
  - `Timestamp()`: returns a UNIX timestamp representing the number of milliseconds that have elapsed since the epoch time January 1, 1970, 00:00:00 UTC.
  - `TimeOfDay()`: returns the number of milliseconds from the current time to 00:00 of the current day in GMT.

2. Value types and evaluation rules
   - The following value types are supported in expressions:

- `STRING`: The value can be a string. Single quotation marks (' ') or double quotation marks (" ") can be used to enclose a string. Examples: `"Hello"` and `'Hello'`.
- `NUMBER`: The value can be an integer or a floating-point number. Examples: `1001`, `-1`, `0.1`, and `-100.0`.
- `BOOLEAN`: The value can be a Boolean value. Valid values: `true` and `false`.

- For the operator types `equal to`, `not equal to`, and `comparison`, the following evaluation rules apply:
  - `STRING` type: uses the string order for evaluation. Examples:
    - `'123' > '10000'`: The result is true.
    - `'A123' > 'A120'`: The result is true.
    - `'' <'a'`: The result is true.
  - `NUMBER` type: uses numerical values for evaluation. Examples:
    - `123 > 1000`: The result is false.
    - `100.0 == 100`: The result is true.
  - `BOOLEAN` type: For Boolean values, true is greater than false. Examples:
    - `true == true`: The result is true.
    - `false == false`: The result is true.
    - `true > false`: The result is true.
- For the operator types `equal to`, `not equal to`, and `comparison`, if the value types before and after an operator are different, the following evaluation rules apply:
  - Assume that a value before an operator is of the `STRING` type and that after the operator is of the `NUMBER` type. If the value type before the operator can be changed to `NUMBER`, use numerical values for evaluation. Otherwise, use the string order for evaluation. Examples:
    - `'100' == 100.0`: The result is true.
    - `'-100' > 0`: The result is false.

- Assume that a value before an operator is of the `STRING` type and that after the operator is of the `BOOLEAN` type. If the value type before the operator can be changed to `BOOLEAN` and the value is not case-sensitive, use `BOOLEAN` values for evaluation. Otherwise, except for the evaluation result of `!=`, all the other evaluation results are `false`. Examples:
  - `'True' == true`: The result is true.
  - `'False' == false`: The result is true.
  - `'bad' == false`: The result is false.
  - `'bad' != false`: The result is true. If the value before the operator is not `true` or `false`, only the result for `!=` is true.
  - `'bad' != true`: The result is true.
  - `'0' > false`: The result is false.
  - `'0' <= false`: The result is false.
- Assume that a value before an operator is of the `NUMBER` type and that after the operator is of the `BOOLEAN` type. The result is false.

- The `null` value is used to check whether a parameter is empty. For the operator types `equal to`, `not equal to`, and `comparison`, the following evaluation rules apply:
  - If the `$A` parameter is empty, the result of `$A == null` is true, and the result of `$A != null` is false.
  - If the empty string `''` is not equal to `null`, the result of `'' == null` is false, and the result of `'' == ''` is true.
  - For the `comparison` operator type, if the value on either side of the operator is `null`, the result is false.
- `like` and `!like` operators are used to match the prefix, suffix, and inclusion of a string. The following evaluation rules apply:
  - In an expression, the value after the operator must be a constant of the `STRING` type. Example: `$Path like '/users/%'`.

- The `'%'` wildcard character in the value after the operator is used to match the prefix, suffix, or inclusion of a string. Examples:
  - Prefix matching: `$Path like '/users/%'` and `$Path !like '/admin/%'`
  - Suffix matching: `$q1 like '%search'` and `$q1 !like '%.do'`
  - Inclusion relation matching: `$ErrorCode like '%400%'` and `$ErrorCode !like '%200%'`
- If the value type before an operator is not `NUMBER` or `BOOLEAN`, change the type to `STRING` and then perform the evaluation.
- If the value before an operator is `null`, the result is `false`.

- `in_cidr` and `!in_cidr` operators are used to identify the mask of a CIDR block. The following evaluation rules apply:
  - The value after an operator must be a constant of the `STRING` type and must be an IPv4 or IPv6 CIDR block. Examples:
    - `$ClientIP in_cidr '10.0.0.0/8'`
    - `$ClientIP !in_cidr '0:0:0:0:0:FFFF::/96'`
  - If the value type before an operator is `STRING`, the value is considered an IPv4 CIDR block for evaluation.
  - If the value type before an operator is `NUMBER` or `BOOLEAN` or the value is empty, the result is `false`.
  - The `System:CaClientIp` parameter specifies the IP address of the client, which is used for evaluation.

## 3. Use cases

- The following expression indicates that the probability is less than 5%:

```
Random() < 0.05
```

- The following expression indicates that the requested API is published to the Test environment:

```
parameters:
    stage: "System:CaStage"
```

```
$CaStage='TEST'
```

- The following expression indicates that the custom parameter UserName is set to Admin and the source IP address is `47.47.74.0/24`:

```
parameters:
  UserName: "Token:UserName"
  ClientIp: "System:CaClientIp"
```

```
$UserName = 'Admin' and $CaClientIp in_cidr '47.47.74.0/24'
```

- The following expression indicates that the AppId parameter is set to 1001, 1098, or 2011, and the protocol that is used by the API request is HTTPS:

```
pameters:
  CaAppId: "System:CaAppId"
  HttpSchema: "System:CaHttpSchema"
```

```
$CaHttpScheme = 'HTTPS' and ($CaAppId = 1001 or $CaAppId = 1098 or $CaAppId = 2011)
```

- The following expression indicates that the JSON string in a body contains result_code that is not ok when StatusCode in a response is 200:

```
parameters:
  StatusCode: "StatusCode"
  ResultCode: "BodyJsonField:$.result_code"
```

```
$StatusCode = 200 and ($ResultCode <> null and $ResultCode <> 'ok')
```

### 4. Limits

- A maximum of 16 parameters can be specified in a plugin.
- A conditional expression can contain a maximum of 512 characters.
- The size of a request or response body specified by BodyJsonField cannot exceed 16 KB. Otherwise, the settings will not take effect.

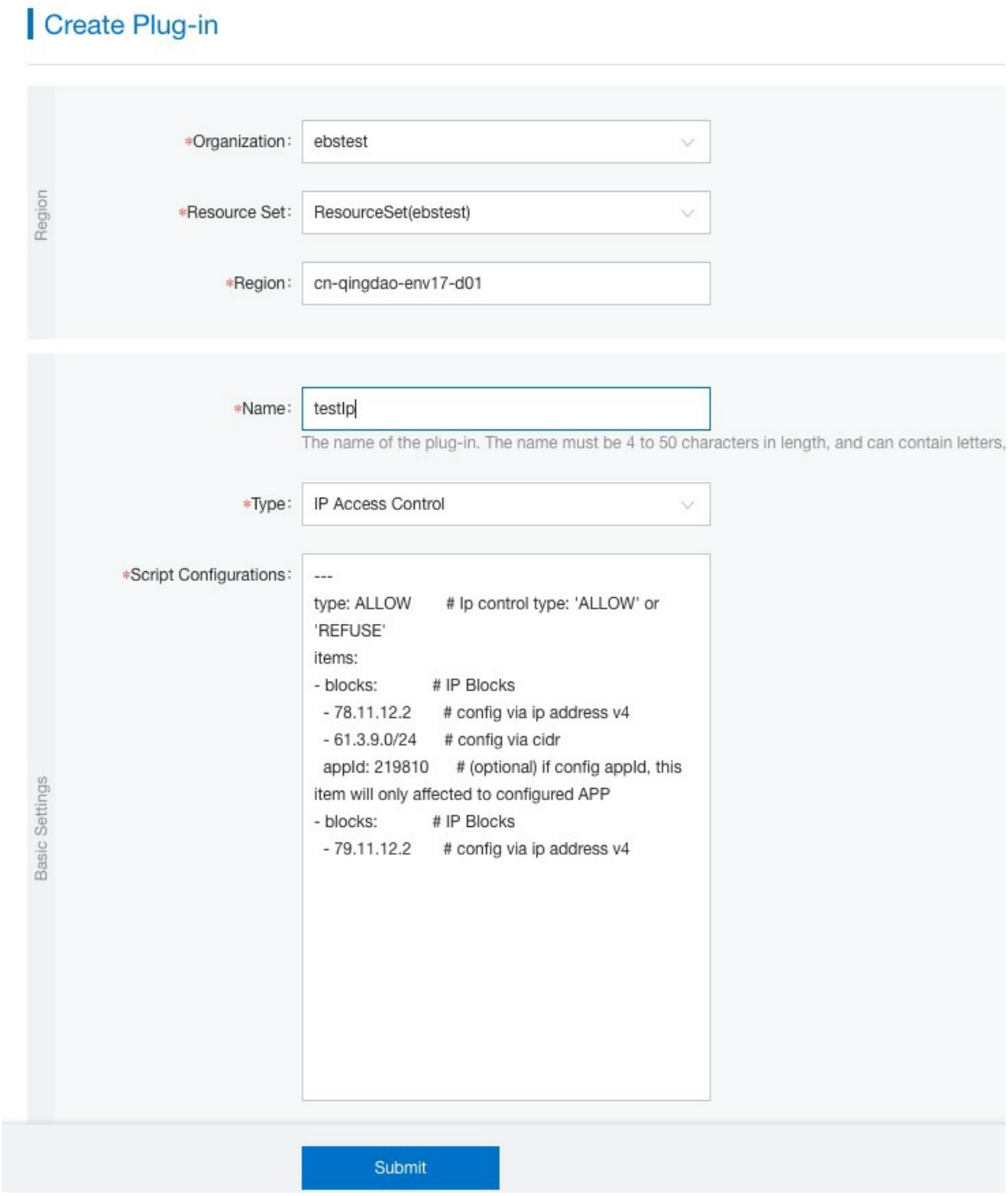
# 2.1.5.4.2. Create a plugin

# 2.1.5.4.2.1. Create an IP address-based access control plug-in

IP address-based access control helps API providers configure an IP address whitelist or blacklist for API calls. This topic describes how to create an IP address-based access control plug-in.

### Procedure

1. Log on to the API Gateway console.
2. In the left-side navigation pane, choose **Publish APIs > Plugin**.
3. On the Plugins list page, click **Create Plugin** in the upper-right corner. On the Create Plug-in page, specify **Organization**, **Resource Set**, **Region**, and **Plug-in Name**, and set Plug-in Type to **IP Access Control**. A plug-in definition template in the YAML format is automatically loaded in the **Script Configuration** field. Modify template content.




| Parameter | Description |
| --- | --- |

| Parameter | Description |
| --- | --- |
| type | ◦ ALLOW: You can configure a whitelist to allow the API requests that meet specific requirements. The following types of whitelists are supported:<br>▪ You can configure a whitelist that contains only IP addresses. In this case, only API requests from the IP addresses in the whitelist are allowed.<br>▪ You can configure a whitelist that contains apps and their IP addresses. In this case, each app can send API requests only from its IP addresses in the whitelist.<br>◦ REFUSE: You can configure an IP address blacklist. API Gateway rejects all API requests from the IP addresses in the blacklist. |

Script template of the IP address-based access control plug-in

```
---
type: ALLOW       # The type of access control. You can set this parameter to ALLOW to
apply a whitelist or to REFUSE to apply a blacklist.
items:
- blocks:         # The IP address segment.
  - 78.11.12.2    # Specifies an IP address.
  - 61.3.9.0/24   # Specifies a CIDR block.
  appId: 219810   # Optional. If you specify this parameter, this IP address-based acce
ss control policy applies only to the app specified by this parameter.
- blocks:         # The IP address segment.
  - 79.11.12.2    # Specifies an IP address.
```

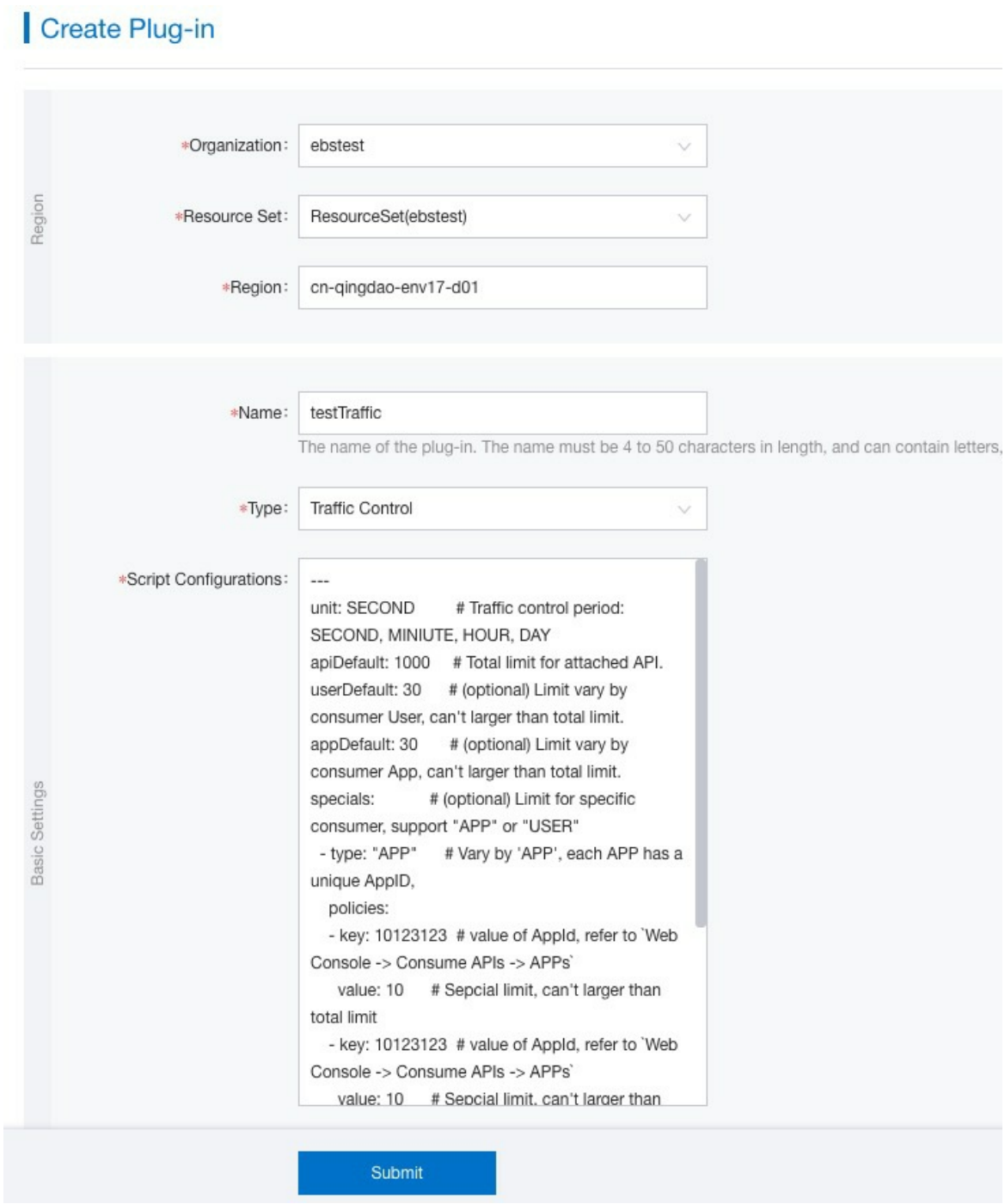
4. Click **Submit**.

# 2.1.5.4.2.2. Create a throttling plug-in

You can use a throttling plug-in to limit the number of API requests. A throttling plug-in helps prevent a backend service from being overwhelmed by a large number of API requests. This topic describes how to create a throttling plug-in.

## Procedure

1. Log on to the API Gateway console .
2. In the left-side navigation pane, choose **Publish APIs > Plugin**.
3. On the Plugins list page, click **Create Plugin** in the upper-right corner. On the Create Plug-in page, specify **Organization**, **Resource Set**, and **Region**.Then, specify **Plug-in Name**, set Plug-in Type to **Traffic Control**, and modify configurations in the Script Configuration field.




| Parameter | Description |
| --- | --- |
| unit | The unit of time. Valid values: SECOND, MINUTE, HOUR, and DAY. |
| apiDefault | The default API-level throttling threshold. It indicates the maximum number of times that an API bound with a throttling policy can be called within a specific unit of time. This parameter is set based on the backend service capability. This parameter is required. |

| Parameter | Description |
| --- | --- |
| userDefault | The default user-level throttling threshold. It indicates the maximum number of times that each user can call an API that is bound with a throttling policy within a specific unit of time. The user-level throttling threshold cannot be greater than the API-level throttling threshold. This parameter is optional. |
| appDefault | The default app-level throttling threshold. It indicates the maximum number of times that each app can call an API that is bound with a throttling policy within a specific unit of time. The app-level throttling threshold cannot be greater than the user-level throttling threshold. This parameter is optional. |
| specials | The special throttling settings. This parameter is optional. You can set throttling thresholds for special apps or users in a throttling policy. After this parameter is specified, the special throttling settings prevail for special apps or users. |

Script template

```
---
unit: SECOND         # The unit of time. Valid values: SECOND, MINUTE, HOUR, and DAY.
apiDefault: 1000     # The default API-level throttling threshold.
userDefault: 30      # Optional. The default user-level throttling threshold. If you se
t this threshold to 0, user-level throttling is not performed. The user-level throttlin
g threshold cannot be greater than the API-level throttling threshold.
appDefault: 30       # Optional. The default app-level throttling threshold. If you set
this threshold to 0, app-level throttling is not performed. The app-level throttling th
reshold cannot be greater than the user-level throttling threshold.
specials:            # Optional. The special throttling settings. You can set throttlin
g thresholds for special apps or users in a throttling policy.
  - type: "APP"      # The special throttling type. The value APP indicates that thrott
ling is performed for special apps based on their AppKeys.
    policies:
    - key: 10123123  # The app ID. You can obtain the ID of an app from the app details
page. To go to this page, choose Consume APIs > APPs in the left-side navigation pane o
f the API Gateway console and click the name of the app.
      value: 10      # The special throttling threshold for the app. This threshold can
not be greater than the user-level throttling threshold in the throttling policy.
    - key: 10123121  # The app ID.
      value: 10      # The special throttling threshold for the app. This threshold can
not be greater than the user-level throttling threshold in the throttling policy.
  - type: "USER"     # The special throttling type. The value USER indicates that throt
tling is performed for special Apsara Stack tenant accounts.
    policies:
    - key: 123455    # The ID of an Apsara Stack tenant account. You can move the point
er over the profile picture in the upper-right corner of the Alibaba Cloud Management C
onsole to obtain the ID.
      value: 100     # The special throttling threshold for the Apsara Stack tenant acc
ount. This threshold cannot be greater than the API-level throttling threshold in the t
hrottling policy.
```

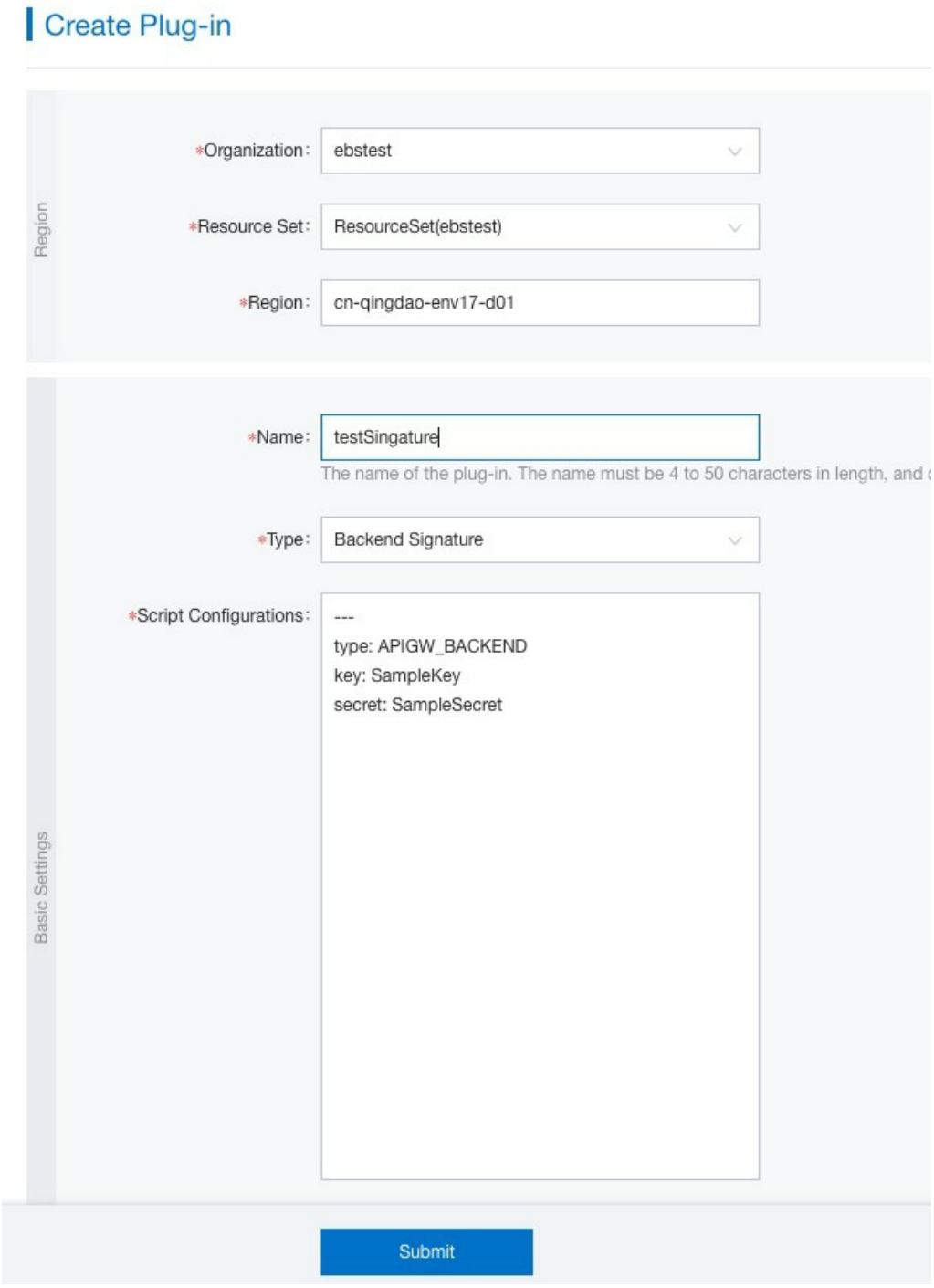
4. Click **Submit**.

# 2.1.5.4.2.3. Create a backend signature plugin

A backend signature plugin is used for signature verification between API Gateway and your backend service. A backend signature is a key-secret pair that you create and issue to API Gateway. It works in a way similar to an account and password pair. When API Gateway sends a request to your backend service, API Gateway uses the backend signature to calculate a signature string and pass it to your backend service. Your backend service obtains the signature string and authenticates API Gateway by using symmetric calculation. Perform the following steps to create a backend signature plugin:

## Procedure

1. Log on to the API Gateway console.
2. In the left-side navigation pane, choose **Publish APIs > Plugin**.
3. On the Plugins list page, click **Create Plugin** in the upper-right corner. On the Create plugin page, specify **Organization**, **Resource Set**, **Region**, and **Name**. Set Type to **Backend Signature**.




Configure the plugin parameters as required.

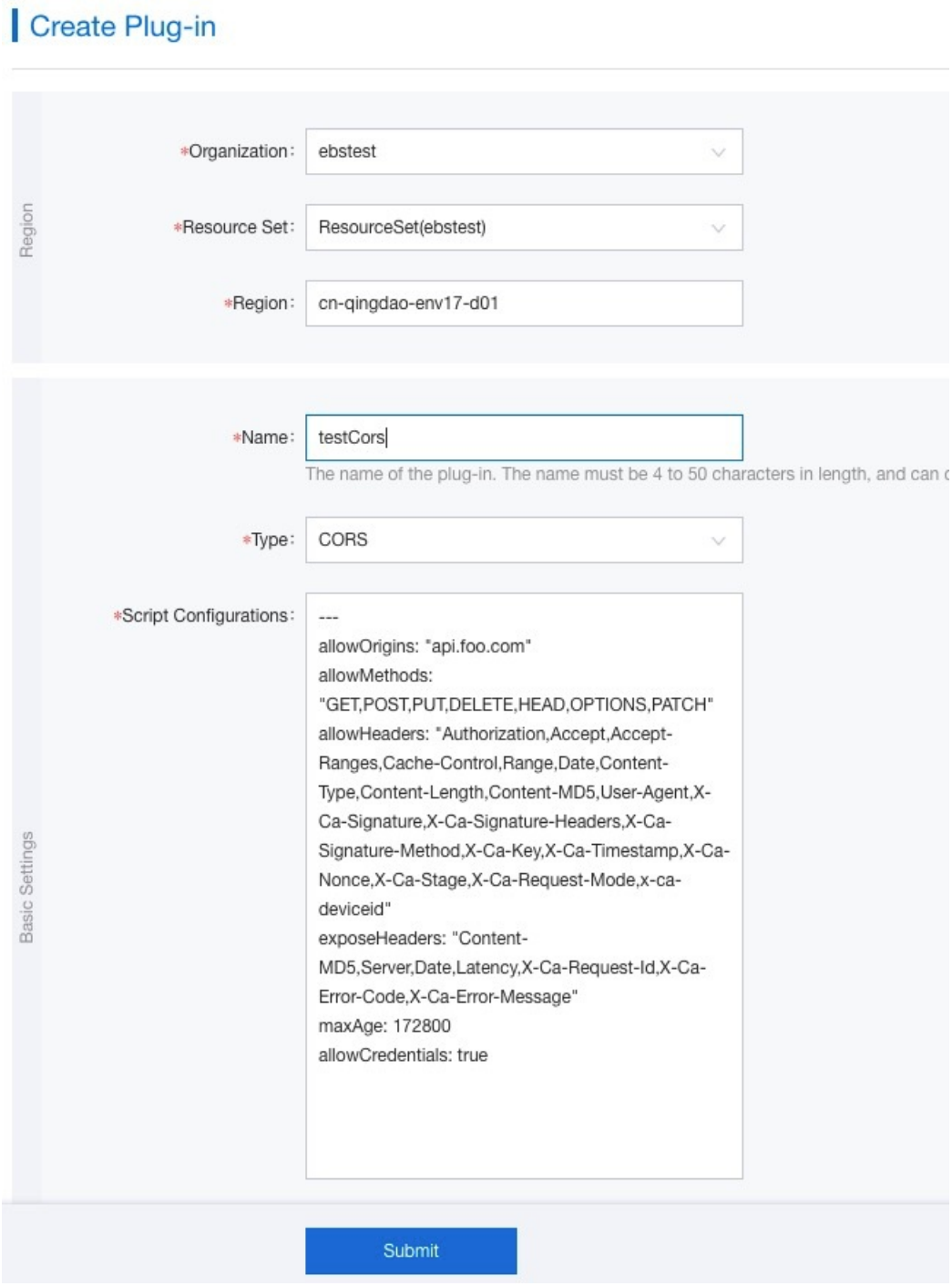
4. Click **Submit**.

# 2.1.5.4.2.4. Create a CORS plugin

This topic describes how to create a cross-origin resource sharing (CORS) plugin. If a resource requests another resource from a different domain or port of a different server, the former resource initiates a cross-domain HTTP request. For security purposes, the browser blocks the request and reports an error message. In this case, you need to use a CORS plugin to troubleshoot the issue.

## Procedure

1. Log on to the API Gateway console.
2. In the left-side navigation pane, choose **Publish APIs > Plugin**.
3. On the Plugins list page, click **Create Plugin** in the upper-right corner. On the Create plugin page, specify **Organization**, **Resource Set**, **Region**, and **Name**. Set Type to **CORS**.




Cross-domain access template

```
---
allowOrigins: api.foo.com,api2.foo.com    # The allowed origins. Separate origins with
commas (,). Default value: *.
allowMethods: GET,POST,PUT                # The allowed HTTP methods. Separate methods
with commas (,).
allowHeaders: X-Ca-RequestId            # The allowed request headers. Separate headers
with commas (,).
exposeHeaders: X-RC1,X-RC2              # The headers that can be exposed to the XMLHtt
pRequest object. Separate headers with commas (,).
allowCredentials: true                    # Controls whether cookies are allowed.
maxAge: 172800
```

Configure the plugin parameters as required.

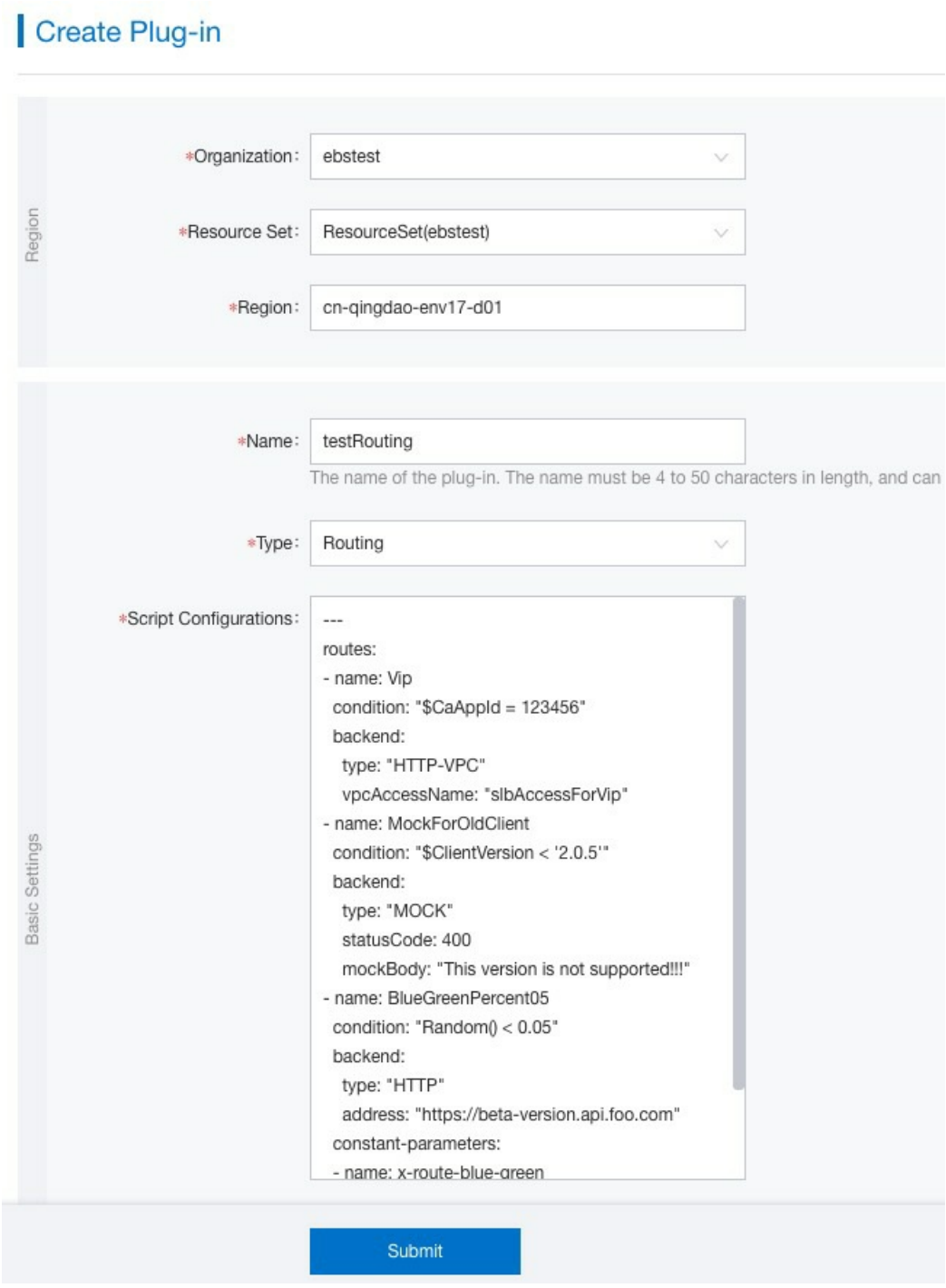
4. Click **Submit**.

# 2.1.5.4.2.5. Create a backend routing plug-in

A backend routing plug-in is used to route API requests to different backend services by changing the backend service type, backend service address, backend request path, and response parameters based on request and system parameters in API requests. Backend routing plug-ins can be used for multi-tenant routing and blue-green release. They can also be used to distinguish between different environments.

## Create a backend routing plug-in

1. Log on to the API Gateway console.
2. In the left-side navigation pane, choose **Publish APIs > Plugin**.
3. On the Plugins list page, click **Create Plugin** in the upper-right corner. On the Create Plug-in page, specify **Organization**, **Resource Set**, **Region**, and **Plug-in Name**. Set Type to **Routing**.




4. Modify configurations in the Script Configuration field and click **Submit**.

## Configurations

Configuration template

1. You can configure a backend routing plug-in in the JSON or YAML format because these two formats have the same schema. You can use the yaml to json tool to convert the plug-in configuration format. The following example describes a plug-in configuration template in the YAML format:

```
---
routes:
# Responses that are no longer supported are returned to clients of an earlier version.
ClientVersion is a custom parameter in the API.
- name: MockForOldClient
  condition: "$ClientVersion < '2.0.5'"
  backend:
    type: "MOCK"
    statusCode: 400
    mockBody: "This version is not supported!!!"
# Blue-green release scenarios: Five percent of requests are routed to the backend of a
blue-green release.
- name: BlueGreenPercent05
  condition: "Random() < 0.05"
  backend:    type: "HTTP"
    address: "https://beta-version.api.foo.com"
  constant-parameters:
  - name: x-route-blue-green
    location: header
    value: "route-blue-green"
```

The template has a root object routes that contains multiple route objects. Each route object is used to specify a routing rule. Each routing rule consists of the following parts:

- name: the name of the routing rule. The name must be unique within each plug-in and can contain only letters and digits. If an API request hits the rule, an HTTP header X-Ca-Routing-Name that contains the name of the rule is added to the request before the request is routed to your backend service.
- condition: the conditional expression of the routing rule. If an API request meets the condition, the request hits the routing rule. The backend routing plug-in checks the routing rules based on the order in which they are configured. The API request is routed to your backend service in the first routing rule that the request hits. After this occurs, the plug-in does not check the remaining routing rules. If you configure multiple routing rules, make sure that they are configured in the order that meets your service expectations.
- backend: the description of your backend service. The description must be consistent with the Swagger specification files for API Gateway. The backend configurations for an API in a backend routing plug-in override the original backend configurations in the API. If the backend configurations are incomplete after they are overridden, the X-Ca-Error-Code: I504RB error is reported to the client. If this error is returned, check whether your backend configurations are complete.
- constant-parameters: the constant parameters that you can customize in the routing rule. Constant parameters are included in an API request before the request is routed to your backend service. These parameters are used in the business logic of your backend service. A constant parameter can be a query or header parameter.

## Conditional expressions

Basic syntax

- The syntax of conditional expressions in backend routing plug-ins are similar to that of SQL statements. The basic format is $A = 'A' and '$B = 'B'.
- Each parameter starts with $. You can reference the request parameters that are defined in an API to

which a plug-in is bound. The request mode of the API can be set to Request Parameter Mapping(Filter Unknown Parameters), Request Parameter Mapping(Passthrough Unknown Parameters), or Request Parameter Passthrough. If you define a request parameter named query1 when you configure an API, you can use $query1 to reference this parameter in conditional expressions.

- The following constant parameter types are supported:
  - STRING: the string data type. Single or double quotation marks can be used to enclose a string. Example: "Hello".
  - INTEGER: the integer data type. Example: 1001 and -1.
  - NUMBER: the floating point data type. Example: 0.1 and 100.0.
  - BOOLEAN: the Boolean data type. Valid values: true and false.
- You can use and and or operators to connect different expressions.
- You can use parentheses () to specify the priority of conditional expressions.
- Random() is a built-in function. It generates a NUMBER-type parameter that returns a random number in the range of [0, 1).
- You can use $CaAppId to reference system parameters of the current request. You can reference system parameters without the need to define them in an API. However, if you have defined a parameter in the API with the same name as a system parameter, the value of the system parameter is overwritten by that of the defined parameter. The following system parameters apply to backend routing plug-ins:
  - CaStage: the environment to which the requested API is published. Valid values: RELEASE, PRE, and TEST.
  - CaDomain: the domain name of the API group to which the requested API belongs.
  - CaRequestHandleTime: the time in UTC at which the current request is received.
  - CaAppId: the value of the AppId parameter in the current request.
  - CaAppKey: the value of the AppKey parameter in the current request.
  - CaClientIp: the IP address of the client from which the current request is sent.
  - CaApiName: the name of the requested API.
  - CaHttpScheme: the protocol used by the current request. Valid values: HTTP, HTTPS, and WS.
  - CaClientUa: the UserAgent field uploaded from the client.
- If you use an unknown parameter in a conditional expression, such as $UnknonwParameter = 1, the result of the expression is false.

Conditional expression examples

- The following expression indicates that the probability is less than 5%:

```
Random() < 0.05
```

- The following expression indicates that the requested API is published to the Test environment:

```
$CaStage = 'TEST'
```

- The following expression indicates that the custom parameter UserName is set to Admin and the source IP address is 47.47.74.77.

```
$UserName = 'Admin' and $CaClientIp = '47.47.74.77'
```

- The following expression indicates that the AppId parameter is set to 1001, 1098, or 2011, and the protocol that is used by the API request is HTTPS:

```
$CaHttpScheme = 'HTTPS' and ($CaAppId = 1001 or $CaAppId = 1098 or $CaAppId = 2011)
```

## Backend configuration and overriding rules

The structure of a backend service is consistent with the Swagger definitions imported into API Gateway. The following examples show the supported backend service types and configuration samples. The backend configurations in a backend routing plug-in override the backend configurations in an API that is bound to the plug-in. If you do not need to change the backend service type, specify only the parameters whose values you want to change.

- HTTP

```
---
backend:
  type: HTTP
  address: "http://10.10.100.2:8000"
  path: "/users/{userId}"
  method: GET
  timeout: 7000
```

- HTTP-VPC

```
---
backend:
  type: HTTP-VPC
  vpcId: vpc-xxxx
  vpcInstance: 172.168.1.1
  vpcInstancePort: 80
  path: "/users/{userId}"
  method: GET
  timeout: 10000
```

- MOCK

```
---
backend:
  type: MOCK
  mockResult: "mock resul sample"
  mockStatusCode: 200
  mockHeaders:
    - name: server
      value: mock
    - name: proxy
      value: GW
```

## Limits

- The metadata of a backend routing plug-in can be a maximum of 16,384 bytes in size. If this limit is exceeded, the InvalidPluginData.TooLarge error is reported.
- A maximum of 16 routing rules can be configured in a backend routing plug-in. If this limit is exceeded, the InvalidPluginData.TooManyRoutes error is reported.

- The size of a single conditional expression cannot exceed 512 bytes. If this limit is exceeded, the InvalidPluginData.ConditionTooLong error is reported.
- Configuration updates in a plug-in are synchronized in real time to all the APIs bound to the plug-in. An interval of at least 45s is required between two updates. If you update a plug-in twice within less than 45s, the InvalidPluginData.UpdateTooBusy error is reported.

## Typical scenarios

- Configure multi-tenant routing. Different backend service addresses are allocated based on the AppId settings. For example, users whose app ID is 10098 or 10099 are VIP customers. API requests from these users are required to be routed to an independent server cluster.

```
---
-routes:
# If the AppId value for an API caller is 10098 or 10099, requests to the API are routed
to an independent address.
# In this example, the VPC access name is set to slbAddressForVip.
- name: http1
  condition: "$CaAppId = 10098 or $CaAppId = 10099"
  backend:
    type: "HTTP"
    address: "https://test-env.foo.com"
```

- Configure routing based on environment settings (Test, Pre, and Release). All requests for the APIs that are published to the same environment are required to be routed to the same server.

```
---
routes:
# Route all requests for APIs that are published to the Test environment to the test serv
er on the Internet.
- name: Vip
  condition: "$CaStage = 'TEST'"
  backend:
    type: "HTTP"
    address: "https://test-env.foo.com"
```

- Perform a blue-green release. Five percent of requests are required to be directed to a group of beta servers to perform a blue-green release.

```
---
routes:
# Blue-green release scenarios: Five percent of requests are routed to the backend of a b
lue-green release.
- name: BlueGreenPercent05
  condition: "Random() < 0.05"
  backend:
    type: "HTTP"
    address: "https://beta-version.api.foo.com"
```

# 2.1.5.4.2.6. Create a caching plugin

You can bind a caching plugin to an API to cache the responses from your backend service. This reduces the load on the backend service and shortens the response time.

## 1. Usage notes

- Caching plugins can cache only the responses to API requests that use the GET method.
- When you configure a caching plugin, you can use the following parameters to sort responses in a cache:
  - varyByApp: controls whether to match and serve cached responses based on the app IDs of API callers.
  - varyByParameters: controls whether to match and serve cached responses based on the values of specific parameters. The plugin uses the same request parameters of APIs that are bound to the plugin to sort the responses to API requests.
  - varyByHeaders: controls whether to match and serve cached responses based on different request headers. For example, match and serve cached responses based on the `Accept` or `Accept-Language` header.
- API Gateway provides each user with 5 MB of cache space in each region. Caches are cleared after expiration. If a cache reaches its space limit, no more responses are stored in the cache.
- If `Cache-Control` is specified in a response from your backend service, the response is stored in a cache based on the specified cache policy. If Cache-Control is not specified in a response, after the response expires, the response is stored in a cache based on the default cache policy and is stored for the period of time specified by the duration parameter.
- A response can be stored in a cache for a maximum of 48 hours (172,800 seconds) after it expires. Configurations made after the 48 hours are invalid.
- API Gateway determines how to process the `Cache-Control` headers of client requests based on the clientCacheControl settings. By default, API Gateway does not process the Cache-Control headers. You can set clientCacheControl to the following modes:
  - off: API Gateway ignores the `Cache-Control` headers of all client requests.
  - all: API Gateway processes the `Cache-Control` headers of all client requests.
  - app: API Gateway processes only the `Cache-Control` headers of client requests whose `app IDs` are included in the configured `apps` list.
- By default, API Gateway caches only the `Content-Type`, `Content-Encoding`, and `Content-Language` headers in responses. If you need to cache more headers, add the headers in the `cacheableHeaders` parameter of the caching plugin.

## 2. Configurations

You can configure a caching plugin in the JSON or YAML format because these two formats have the same schema. You can use the yaml to json tool to convert the plugin configuration format. The following example describes a plugin configuration template in the YAML format:

```
---
varyByApp: false    # Controls whether to match and serve cached responses based on the app IDs of API callers. Default value: false.
varyByParameters:   # Controls whether to match and serve cached responses based on the values of specific parameters.
- userId            # The name of a backend parameter. If the backend parameter is mapped to a parameter with a different name, set this parameter to the mapped parameter name.
varyByHeaders:      # Controls whether to match and serve cached responses based on different request headers.
- Accept            # Cached responses are matched and served based on the Accept header.
clientCacheControl: # API Gateway determines how to process the Cache-Control headers of client requests based on the clientCacheControl settings.
  mode: "app"       # Valid values: off, all, and apps. Default value: off. off indicates that API Gateway ignores the Cache-Control headers of all client requests. all indicates that API Gateway processes the Cache-Control headers of all client requests. apps indicates that API Gateway processes only the Cache-Control headers of client requests whose app IDs are included in the configured apps list.
  apps:             # A list of app IDs. If mode is set to app, API Gateway processes only the Cache-Control headers of client requests whose app IDs are in this list.
  - 1992323         # A sample app ID. It is not an AppKey.
  - 1239922         # A sample app ID. It is not an AppKey.
cacheableHeaders:   # The cacheable response headers. By default, API Gateway caches only the Content-Type and Content-Length headers of backend responses.
- X-Customer-Token  # The name of the cacheable response header.
duration: 3600      # The default grace period, in seconds.
```

### 3. Working mechanism

If an API request hits the cache of an API, the `X-Ca-Caching: true` header is included in the response to the API request.

### 4. Limits

- The metadata of a caching plugin can be a maximum of 16,380 bytes in size.
- A response body that exceeds 128 KB in size cannot be cached.
- Each user has a maximum of 30 MB of total cache space in each region.

## 2.1.5.4.2.7. JWT authentication plug-in

RFC 7519-compliant JSON Web Token (JWT) is a simple method used by API Gateway to authenticate requests. `API Gateway` hosts the `public JSON Web Keys (JWKs)` of users and uses these JWKs to sign and authenticate JWTs in requests. Then, API Gateway forwards `claims` to backend services as backend parameters. This simplifies the development of backend applications.

Compared with the `OpenID Connect` feature, the `JWT authentication plug-in` can implement the functions of this feature and bring the following benefits:

- You do not need to configure an additional `authorization API`. `JWTs` can be generated and distributed in multiple ways. API Gateway is only responsible for `JWT` authentication by using `public JWKs`.
- `JWKs` without `kid` specified are supported.

- Multiple `JWKs` can be configured.
- You can read token information from the `header` of a request or a `query` parameter.
- If you want to transmit a `JWT` in an Authorization header, such as `Authorization bearer {token}`, you can set `parameter` to Authorization and `parameterLocation` to header so that the token information is correctly read.
- The `jti` claim-based anti-replay check is supported if you set `preventJtiReplay` to true.
- Requests that do not include tokens can be forwarded to backend services without verification if you set `bypassEmptyToken` to true.
- The verification on the `exp` setting for tokens can be skipped if you set `ignoreExpirationCheck` to true.

If you configure a `JWT authentication plug-in` and bind it to an `API` for which the `OpenID Connect` feature is configured, the JWT authentication plug-in takes effect in place of the `OpenID Connect` feature.

## 1. Obtain a JWK

RFC 7517-compliant JWK is used to sign and authenticate JWTs. If you want to configure a `JWT authentication plug-in`, you must generate a valid `JWK` manually or by using an online `JWK generator` such as mkjwk.org. The following example shows a valid `JWK`. In the JWK example, the private key is used to sign the token, and the public key is configured in the `JWT authentication plug-in` to authenticate the signature.

```
{
  "kty": "RSA",
  "e": "AQAB",
  "kid": "O9fpdhrViq2zaaaBEWZITz",
  "use": "sig",
  "alg": "RS256",
  "n": "qSVxcknOm0uCq5vGsOmaorPDzHUubBmZZ4UXj-9do7w9X1uKFXAnqfto4TepSNuYU2bA_-tzSLAGBsR-Bqv
T6w9SjxakeiyQpVmexxnDw5WZwpWenUAcYrfSPEoNU-0hAQwFYgqZwJQMN8ptxkd0170PFauwACOx4Hfr-9FPGy8NCo
IO4MfLXzJ3mJ7xqgIZp3NIOGXz-GIAbCf13ii7kSStpYqN3L_zzpvXUAos1FJ9IPXRV84tIZpFVh2lmRh0h8ImK-vI4
2dwlD_hOIzayL1Xno2R0T-d5AwTSdnep7g-Fwu8-sj4cCRWq3bd61Zs2QOJ8iustH0vSRMYdP5oYQ"
}
```

*The preceding JWK is in the JSON format. If you want to configure a JWT authentication plug-in in the YAML format, you must use a JWK in the YAML format.**

- For a `JWT authentication plug-in`, you need only to configure a `public key`. Keep your `private key` safe. The following table lists the signature algorithms supported by the JWT authentication plug-in.

| Signature algorithm | Supported `alg` setting |
| --- | --- |
| RSASSA-PKCS1-V1_5 with SHA-2 | RS256, RS384, RS512 |
| Elliptic Curve (ECDSA) with SHA-2 | ES256, ES384, ES512 |
| HMAC using SHA-2 | HS256, HS384, HS512 |

*When you configure a key of the HS256, HS384, or HS512 type, the key value is base64url encoded. If the signature is invalid, check whether your key is in the same format as the key used to generate the token.*

## 2. Plug-in configurations

You can configure a JWT authentication plug-in in the JSON or YAML format because these two formats have the same schema. You can use the `yaml to json` tool to convert the plug-in configuration format. The following example shows a plug-in configuration template in the YAML format:

```
---
parameter: X-Token           # The parameter from which the JWT is read. It corresponds to
a parameter in an API request.
parameterLocation: header    # The location from which the JWT is read. Valid values: query
and header. This parameter is optional if Request Mode for the bound API is set to Request
Parameter Mapping(Filter Unknown Parameters) or Request Parameter Mapping(Passthrough Unkno
wn Parameters). This parameter is required if Request Mode for the bound API is set to Requ
est Parameter Passthrough.
preventJtiReplay: false      # Controls whether to enable the anti-replay check for jti. De
fault value: false.
bypassEmptyToken: false      # Controls whether to forward requests that do not include tok
ens to backend services without verification.
ignoreExpirationCheck: false # Controls whether to ignore the verification of the exp setti
ng.
claimParameters:             # The claims to be converted into parameters. API Gateway maps
JWT claims to backend parameters.
- claimName: aud             # The name of the JWT claim, which can be public or private.
  parameterName: X-Aud       # The name of the backend parameter, to which the JWT claim is
mapped.
  location: header           # The location of the backend parameter, to which the JWT clai
m is mapped. Valid values: query, header, path, and formData.
- claimName: userId          # The name of the JWT claim, which can be public or private.
  parameterName: userId      # The name of the backend parameter, to which the JWT claim is
mapped.
  location: query            # The location of the backend parameter, to which the JWT clai
m is mapped. Valid values: query, header, path, and formData.
#
# Public key in the JWK
jwk:
  kty: RSA
  e: AQAB
  use: sig
  alg: RS256
  n: qSVxcknOm0uCq5vGsOmaorPDzHUubBmZZ4UXj-9do7w9X1uKFXAnqfto4TepSNuYU2bA_-tzSLAGBsR-BqvT6w
9SjxakeiyQpVmexxnDw5WZwpWenUAcYrfSPEoNU-0hAQwFYgqZwJQMN8ptxkd0170PFauwACOx4Hfr-9FPGy8NCoIO4
MfLXzJ3mJ7xqgIZp3NIOGXz-GIAbCf13ii7kSStpYqN3L_zzpvXUAos1FJ9IPXRV84tIZpFVh2lmRh0h8ImK-vI42dw
lD_hOIzayL1Xno2R0T-d5AwTSdnep7g-Fwu8-sj4cCRWq3bd61Zs2QOJ8iustH0vSRMYdP5oYQ
#
# You can configure multiple JWKs and use them together with the jwk field.
# If multiple JWKs are configured, kid is required. If the JWT does not include kid, the co
nsistency check on kid fails.
jwks:
- kid: O9fpdhrViq2zaaaBEWZITz  # If only one JWK is configured, kid is optional. If the JWT
includes kid, API Gateway checks the consistency of kid.
  kty: RSA
```

```
  kty: RSA
  e: AQAB
  use: sig
  alg: RS256
  n: qSVxcknOm0uCq5v....
- kid: 10fpdhrViq2zaaaBEWZITz  # If only one JWK is configured, kid is optional. If the JWT
includes kid, API Gateway checks the consistency of kid.
  kty: RSA
  e: AQAB
  use: sig
  alg: RS256
  n: qSVxcknOm0uCq5v...
```

- The `JWT authentication plug-in` retrieves JWTs based on the settings of `parameter` and `parameterLocation`. For example, if `parameter` is set to X-Token and `parameterLocation` is set to header, the JWT is read from the `X-Token` header.
- If the parameter configured in an API has the same name as the parameter specified by `parameter`, do not specify `parameterLocation`. Otherwise, an error is reported when the API is called.
- If you want to transmit a token in an Authorization header, such as `Authorization bearer {token}`, you can set `parameter` to Authorization and `parameterLocation` to header so that the token information can be correctly read.
- If `preventJtiReplay` is set to true, the JWT authentication plug-in uses `jti` in `claims` to perform an anti-replay check.
- If `bypassEmptyToken` is set to true and a token is not included in a request, API Gateway skips the check and directly forwards the request to a backend service.
- If `ignoreExpirationCheck` is set to true, API Gateway skips the verification of the `exp` setting. Otherwise, API Gateway checks whether a token expires.
- If API Gateway is required to forward `claims` in tokens to backend services, you can set `tokenParameters` to configure the following parameters to be forwarded:
  - `claimName`: the name of the claim in a token, which can be `kid`.
  - `parameterName`: the name of the parameter forwarded to a backend service.
  - `location`: the location of the parameter forwarded to a backend service. Valid values: `header`, `query`, `path`, and `formData`.
    - If this parameter is set to `path`, the backend path must contain a parameter with the same name, such as `/path/{userId}`.
    - If this parameter is set to `formData`, the body of a received request in a backend service must be of the `Form` type.
- You can configure only one key in the `jwk` field. You can also configure multiple keys in the `jwks` field.
  - You can configure only one key with `kid` not specified.
  - You can configure multiple keys with `kid` specified. `kid` must be unique.

## 3. Verification rules

- A JWT authentication plug-in obtains tokens based on the settings of `parameter` and `parameterT`

oken. If API Gateway is required to forward requests to backend services even when tokens are not included in the requests, set bypassEmptyToken to true.

- If you want to configure multiple keys, abide by the following principles:
  - Preferentially select a key whose ID is the same as the value of kid in a token for signature and authentication.
  - You can configure only one key with kid not specified. If no key whose ID is the same as the value of kid in a token exists, use the key with kid not specified for signature and authentication.
  - If all the configured keys have specified kid settings, and the token in a request does not contain kid or no keys match kid, an A403JK error is reported.
- If a token contains iat, nbf, and exp, the JWT authentication plug-in verifies the validity of their time formats.
- By default, API Gateway verifies the setting of exp. If you want to skip the verification, set ignoreExpirationCheck to true.
- tokenParameters is configured to extract the required parameters from the claims of a token. These parameters are forwarded to backend services.

# 4. Configuration examples

## 4.1 Configure a single JWK

```
---
parameter: X-Token          # The parameter from which the JWT is read. It corresponds to a
parameter in an API request.
parameterLocation: header   # The location from which the JWT is read. Valid values: query a
nd header. This parameter is optional if Request Mode for the bound API is set to Request P
arameter Mapping(Filter Unknown Parameters) or Request Parameter Mapping(Passthrough Unknow
n Parameters). This parameter is required if Request Mode for the bound API is set to Reque
st Parameter Passthrough.
claimParameters:            # The claims to be converted into parameters. API Gateway maps J
WT claims to backend parameters.
- claimName: aud            # The name of the JWT claim, which can be public or private.
  parameterName: X-Aud      # The name of the backend parameter, to which the JWT claim is m
apped.
  location: header          # The location of the backend parameter, to which the JWT claim
is mapped. Valid values: query, header, path, and formData.
- claimName: userId         # The name of the JWT claim, which can be public or private.
  parameterName: userId     # The name of the backend parameter, to which the JWT claim is m
apped.
  location: query           # The location of the backend parameter, to which the JWT claim
is mapped. Valid values: query, header, path, and formData.
preventJtiReplay: false     # Controls whether to enable the anti-replay check for jti. Defa
ult value: false.
#
# Public key in the JWK
jwk:
  kty: RSA
  e: AQAB
  use: sig
  alg: RS256
  n: qSVxcknOm0uCq5vGsOmaorPDzHUubBmZZ4UXj-9do7w9X1uKFXAnqfto4TepSNuYU2bA_-tzSLAGBsR-BqvT6w
9SjxakeiyQpVmexxnDw5WZwpWenUAcYrfSPEoNU-0hAQwFYgqZwJQMN8ptxkd0170PFauwACOx4Hfr-9FPGy8NCoIO4
MfLXzJ3mJ7xqgIZp3NIOGXz-GIAbCf13ii7kSStpYqN3L_zzpvXUAos1FJ9IPXRV84tIZpFVh2lmRh0h8ImK-vI42dw
lD_hOIzayL1Xno2R0T-d5AwTSdnep7g-Fwu8-sj4cCRWq3bd61Zs2QOJ8iustH0vSRMYdP5oYQ
```

## 4.2 Configure multiple JWKs

```
---
parameter: Authorization   # The parameter from which the token is obtained.
parameterLocation: header  # The location from which the token is obtained.
claimParameters:           # The claims to be converted into parameters. API Gateway maps J
WT claims to backend parameters.
- claimName: aud           # The name of the JWT claim, which can be public or private.
  parameterName: X-Aud     # The name of the backend parameter, to which the JWT claim is m
apped.
  location: header         # The location of the backend parameter, to which the JWT claim
is mapped. Valid values: query, header, path, and formData.
- claimName: userId        # The name of the JWT claim, which can be public or private.
  parameterName: userId    # The name of the backend parameter, to which the JWT claim is m
apped.
  location: query          # The location of the backend parameter, to which the JWT claim
is mapped. Valid values: query, header, path, and formData.
preventJtiReplay: true     # Controls whether to enable the anti-replay check for jti. Defa
ult value: false.
jwks:
- kid: O9fpdhrViq2zaaaBEWZITz  # kid must be set to different values for different JWKs.
  kty: RSA
  e: AQAB
  use: sig
  alg: RS256
  n: qSVxcknOm0uCq5v....
- kid: 10fpdhrViq2zaaaBEWZITz  # kid must be set to different values for different JWKs.
  kty: RSA
  e: AQAB
  use: sig
  alg: RS256
  n: qSVxcknOm0uCq5v...
```

## 5. Error codes

| HTTP status code | Error code | Error message | Description |
| --- | --- | --- | --- |
| 400 | I400JR | JWT required | The error message returned because no JWT-related parameters are found. |
| 403 | S403JI | Claim `jti` is required when `preventJtiReplay:true` | The error message returned because no valid `jti` claims are included in the request when preventJtiReplay is set to true in a `JWT authentication plug-in`. |

| HTTP status code | Error code | Error message | Description |
| --- | --- | --- | --- |
| 403 | S403JU | Claim `jti` in JWT is used | The error message returned because the `jti` claim that is included in the request has been used when preventJtiReplay is set to true in a `JWT authentication plug-in`. |
| 403 | A403JT | Invalid JWT: ${Reason} | The error message returned because the `JWT` that is read from the request is invalid. |
| 400 | I400JD | JWT Deserialize Failed: `${Token}` | The error message returned because the `JWT` that is read from the request fails to be parsed. |
| 403 | A403JK | No matching JWK, `kid:${kid}` not found | The error message returned because no `JWK` matches `kid` configured in the `JWT` included in the request. |
| 403 | A403JE | JWT is expired at `${Date}` | The error message returned because the `JWT` that is read from the request expires. |
| 400 | I400JP | Invalid JWT plugin config: ${JWT} | The error message returned because the `JWT authentication plug-in` is incorrectly configured. |

*If an HTTP response message includes an unexpected response code specified by `ErrorCode` in the `X-Ca-Error-Code` header, such as `A403JT` or `I400JD`, you can visit the jwt.io website to check the token validity and format.*

## 6. Limits

- The metadata of a JWT authentication plug-in can contain a maximum of **16,380** characters.
- You can configure a maximum of **16** parameters to be forwarded. Both the `claimName` and `parameterName` parameters cannot exceed **32** characters in length. Only the following regular expression is supported: [A-Za-z0-9-_].

- `alg` can be set to RS256, RS384, RS512, ES256, ES384, ES512, HS256, HS384, or HS512 for JWKs.

# 2.1.5.4.2.8. Access control plugin

## 1. Overview

In an access control plugin, you can define conditions based on the request parameters or context of an API to which the plugin is bound. This allows you to determine whether to deliver an API request to a backend service. For information about how to define parameters and use conditional expressions, see Use parameters and conditional expressions.

## 2. Configurations

Assume that the API request path is `/{userId}/...`. JWT authentication is enabled for APIs. Two claims, userId and userType, are available in the JWT. The following plugin verification conditions apply:

- If userType is set to admin, requests in all paths are allowed.
- If userType is set to user, only the requests in the same /{userId} path are allowed.

```
---
#
# Assume that the API request path is /{userId}/... in this example.
# JWT authentication is enabled for APIs. Two claims, userId and userType, are available in the JWT.
# The following plugin verification conditions apply:
# - If userType is set to admin, requests in all paths are allowed.
# - If userType is set to user, only the requests in the same /{userId} path are allowed.
parameters:
  userId: "Token:userId"
  userType: "Token:userType"
  pathUserId: "path:userId"
#
# Rules are defined based on the preceding parameters. For each API request, the plugin checks the rules in sequence. If a condition in a rule is met, the result is true and the action that is specified by ifTrue is performed. If a condition in a rule is not met, the result is false and the action that is specified by ifFalse is performed.
# The action ALLOW indicates that the request is allowed. The action DENY indicates that the request is denied and an error code is returned to the client. After the ALLOW or DENY action is performed, the plugin does not check the remaining conditions.
# If neither the ALLOW nor DENY action is performed, the plugin proceeds to check the next condition.
rules:
  - name: admin
    condition: "$userType = 'admin'"
    ifTrue: "ALLOW"
  - name: user
    condition: "$userId = $pathUserId"
    ifFalse: "DENY"
    statusCode: 403
    errorMessage: "Path not match ${userId} vs /${pathUserId}"
    responseHeaders:
      Content-Type: application/xml
    responseBody:
      <Reason>Path not match ${userId} vs /${pathUserId}</Reason>
```

## 3. Relevant errors

| Error code | HTTP status code | Message | Description |
| --- | --- | --- | --- |
| A403AC | 403 | Access Control Forbidden by ${RuleName} | The error message returned because the request is rejected by the access control plugin that is bound to the API. |

## 4. Limits

- A maximum of **16** parameters can be specified in an access control plugin.
- Each conditional expression can contain a maximum of **512** characters.
- The metadata of an access control plugin can contain a maximum of **16,380** characters.

- A maximum of 16 `rules` can be configured in each access control plugin.

# 2.1.5.4.2.9. Error code mapping plug-in

An `error code mapping plug-in` is used to map backend error responses to expected error responses based on mapping rules that are defined by clients.

## 1. Overview

An error code mapping plug-in is used to map backend error responses to expected error responses based on mapping rules that are defined by clients.

## 2. Quick start

The following example shows an error response that is returned by a backend service. The HTTP status code is 200, but the response body contains an error message in a JSON string.

```
HTTP 200 OK
Content-Type:application/json
{"req_msg_id":"d02afa56394f4588832bed46614e1772","result_code":"ROLE_NOT_EXISTS"}
```

- Clients want to receive an HTTP status code other than 200 but do not want to modify backend configurations. The clients expect the following sample error response:

```
HTTP 404
X-Ca-Error-Message: Role Not Exists, ResultId=d02afa56394f4588832bed46614e1772
```

In this case, you can use the following sample to configure an error code mapping plug-in and bind the plug-in to the related APIs:

```
---
# The parameters that are involved in a mapping.
parameters:
  statusCode: "StatusCode"
  resultCode: "BodyJsonField:$.result_code"
  resultId: "BodyJsonField:$.req_msg_id"
# The mapping condition.
errorCondition: "$statusCode = 200 and $resultCode <> 'OK'"
# The parameter in an error response that is used to specify the error code and hit mapping
rules.
errorCode: "resultCode"
# Mapping rules.
mappings:
  - code: "ROLE_NOT_EXISTS"
    statusCode: 404
    errorMessage: "Role Not Exists, RequestId=${resultId}"
  - code: "INVALID_PARAMETER"
    statusCode: 400
    errorMessage: "Invalid Parameter, RequestId=${resultId}"
# Optional. The default mapping rule.
defaultMapping:
  statusCode: 500
  errorMessage: "Unknown Error, ${resultCode}, RequestId=${resultId}"
```

In this example, the HTTP status code and the `result_code` parameter in an error response are used to define the mapping condition. If the HTTP status code of an error response is 200 but the value of the `result_code` parameter is not `'OK'`, the mapping starts. The `result_code` parameter is used to define the mapping rules. If the value of the result_code parameter is `ROLE_NOT_EXISTS`, the original HTTP status code is mapped to 404. If the value of the result_code parameter is `INVALID_PARAMETER`, the original HTTP status code is mapped to 400. If the value of the result_code parameter is neither of the preceding values, the original HTTP status code is mapped to 500.

# 3. Plug-in configurations and mapping rules

## 3.1 Plug-in configurations

You can configure an error code mapping plug-in in the `JSON` or `YAML` format. The following parameters can be specified:

- `parameters`: required. The parameters that are involved in a mapping. These parameters are specified as key-value pairs in the `map` format. For information about how to define parameters and write conditional expressions, see Use parameters and conditional expressions.
- `errorCondition`: required. The condition under which a response is considered an error response. If the result of the conditional expression is `true`, the mapping starts.

- `errorCode`: optional. The parameter that is used to specify the error code in an error response and hit mapping rules. The error code that is specified by this parameter is compared with the value of the `code` parameter in the mapping rules specified by `mappings`.
- `mappings`: required. The mapping rules. API Gateway reconstructs error responses based on the setting of `errorCode` or `errorCondition`. A mapping rule may contain the following parameters:
  - `code`: optional. The value of this parameter must be unique among all mapping rules. If the error code of an error response is the same as the value of the `code` parameter in the current mapping rule, the error response is mapped based on the current mapping rule.
  - `condition`: optional. The condition under which an error response needs to be mapped based on the current mapping rule. If the result of the conditional expression is `true`, the error response is mapped based on the current mapping rule.
  - `statusCode`: required. The HTTP status code that replaces the original HTTP status code of an error response if the error response needs to be mapped based on the current mapping rule.
  - `errorMessage`: optional. The error message that is returned to the client after a mapping. The value of this parameter is obtained from the parameters in the original backend error response and is also stored in the `errorMessage` parameter in error logs. In the error response after the mapping, this parameter is displayed as the value of the `X-Ca-Error-Message` header.
  - `responseHeaders`: optional. The response headers that are included in an error response after a mapping if the current mapping rule is hit. This parameter is specified as key-value pairs in the map format.
  - `responseBody`: optional. The response body that overwrites the original response body of an error response if the error response needs to be mapped based on the current mapping rule.
- `defaultMapping`: optional. The default mapping rule. If all the rules that are defined in `mappings` are not hit by an error response, the error response is mapped based on this default mapping rule.
  - `statusCode`: required. The HTTP status code that replaces the original HTTP status code of an error response if the error response needs to be mapped based on the current mapping rule.
  - `errorMessage`: optional. The error message that is returned to the client after a mapping. The value of this parameter is obtained from the parameters in the original backend error response and is also stored in the `errorMessage` parameter in error logs. In the error response after the mapping, this parameter is displayed as the value of the `X-Ca-Error-Message` header.

- `responseHeaders`: optional. The response headers that are included in an error response after a mapping if the current mapping rule is hit. This parameter is specified as key-value pairs in the map format.
- `responseBody`: optional. The response body that overwrites the original response body of an error response if the error response needs to be mapped based on the current mapping rule.

Take note of the following points when you configure an error code mapping plug-in:

- The parameters that are used to write conditional expressions in `mappingCondition` and `mappings[].condition` must be defined in `parameters`. Otherwise, the plug-in does not work and reports an error. For information about how to define parameters and write conditional expressions, see Use parameters and conditional expressions.
- The value of the `errorCode` parameter must be the name of a parameter that is defined in `parameters`.
- When you configure a mapping rule specified by `mappings`, you must specify `code` or `condition`. When you specify `code`, the value of this parameter must be unique among all mapping rules. When you specify `condition`, you must write conditional expressions in the order that meets your requirements. This is because the order of conditions determines their priorities.
- You can specify `errorMessage` and `responseBody` in a format similar to `"${Code}: ${Message}"` and obtain the parameter values from those specified in `parameters`.
- You can specify `responseHeaders` in the `${Message}` format.
- If you do not specify `responseBody`, the body of an error response returned to the client after a mapping is the same as that of the original error response.
- You can use the `responseHeaders` parameter to specify headers and their settings to replace corresponding headers in a backend error response. If you specify the value of a header as `''`, this header will be deleted after a mapping. If you do not specify this parameter, the headers of the error response returned to the client after the mapping are the same as those of the original error response.
- If you do not specify `defaultMapping`, the error code mapping does not take effect. The original error response from your backend service is returned to the client.

## 3.2 Parameters involved in a mapping

As described in the following code, you must specify the parameters that are involved in a mapping as key-value pairs in `parameters`. Each key is the name of a parameter. Each value is specified in the `Location:Name` format. This format indicates that the value of a parameter is obtained from a specific location in the response or system context.

```
---
# The parameters that are involved in a mapping.
parameters:
  statusCode: "StatusCode"
  resultCode: "BodyJsonField:$.result_code"
  resultId: "BodyJsonField:$.req_msg_id"
```

An error code mapping plug-in supports the parameters at specific locations in the following table.

| Location | Included in | Description |
| --- | --- | --- |
| StatusCode | Response | The HTTP status code in a backend error response, such as `200` or `400`. |
| ErrorCode | Response | The error code of a system error response in API Gateway. |
| ErrorMessage | Response | The system error message in API Gateway. |
| Header | Response | Use `Header:{Name}` to obtain the first value of the HTTP header that is specified by `{Name}`. |
| BodyJsonField | Response* | Use `BodyJsonField:{JPath}` to obtain the JSON string in the body of an API request or a backend response in `JSONPath` mode. |
| System | Response | Use `System:{Name}` to obtain the value of the system parameter that is specified by `{Name}`. |

| Location | Included in | Description |
| --- | --- | --- |
| Token | Response | If `JWT` is used with `OAuth2` for authentication, use `Token:{Name}` to obtain the value of the parameter that is specified by `{Name}` in a token. |

- `ErrorCode` and `ErrorMessage` are used to return system error codes and detailed system error information in API Gateway. For more information, see Error codes.
- `BodyJsonField` can be used to obtain the JSON string in the body of a backend response. However, if the size of the response body exceeds **15,360** bytes, the string obtained is `null`.

## 3.3 Working mechanism

The following operations describe how an error code mapping plug-in works:

1. The plug-in obtains the values of the parameters from a backend error response and the system context based on the list of parameters that are defined in `parameters`.
2. The plug-in uses the parameters and obtained values to execute the conditional expression that is written in `errorCondition`. If the result is `true`, go to the next step. If the result is `false`, the process ends.
3. If `errorCode` is specified, the plug-in obtains the value of `errorCode`. Then, the plug-in checks whether a mapping rule exists, which indicates that the errorCode setting is the same as the setting of `code`. The mapping rule is specified by `mappings`.
4. If no mapping rule meets requirements, the plug-in executes in sequence the conditional expressions that are written in `condition` in mapping rules.
5. If a mapping rule is hit in Step 3 or Step 4, the original error response is mapped based on the mapping rule. Otherwise, the original error response is mapped based on the default mapping rule.

## 3.4 Mapping of system error codes and error logs

- In API Gateway, system errors may occur in processes such as check, verification, throttling, and plug-in operations. For more information, see Error codes. You can use `ErrorCode` as a location to obtain information in a system error response. For example, clients support only HTTP status code 200 and want to map HTTP status code 429 that is returned by API Gateway to HTTP status code 200.

- For a system error response, the values that are obtained from locations such as `StatusCode`, `Header`, and `BodyJsonField` are all `null`. When you define a mapping condition for an error code mapping plug-in, the value that is obtained from the `ErrorCode` location is `OK` for a backend error response.
- The error code of a system error response is specified by the `X-Ca-Error-Code` header and by the `errorCode` parameter in error logs. This value cannot be overwritten by an `error code mapping plug-in`.
- The `statusCode` parameter in error logs records the value of the HTTP status code that is sent from API Gateway to the client. This value can be overwritten by an `error code mapping plug-in`.

## 4. Configuration examples

### 4.1 Use the error codes in error responses for a mapping

Mapping

```
---
# The parameters that are involved in a mapping.
parameters:
  statusCode: "StatusCode"
  resultCode: "BodyJsonField:$.result_code"
  resultId: "BodyJsonField:$.req_msg_id"
# The mapping condition.
errorCondition: "$statusCode = 200 and $resultCode <> 'OK'"
# The parameter in an error response that is used to specify the error code and hit mapping
rules.
errorCode: "resultCode"
# Mapping rules.
mappings:
  - code: "ROLE_NOT_EXISTS"
    statusCode: 404
    errorMessage: "Role Not Exists, RequestId=${resultId}"
  - code: "INVALID_PARAMETER"
    statusCode: 400
    errorMessage: "Invalid Parameter, RequestId=${resultId}"
# Optional. The default mapping rule.
defaultMapping:
  statusCode: 500
  errorMessage: "Unknown Error, ${resultCode}, RequestId=${resultId}"
```

## 5. Limits

- A maximum of 16 parameters can be specified in an error code mapping plug-in.
- A single conditional expression can contain a maximum of 512 characters.

- If you use the `BodyJsonField` location to obtain the JSON string in the body of an error response, the size of the response body cannot exceed **16,380** bytes. If the size of the response body exceeds this limit, the obtained string is null.
- The metadata of an error code mapping plug-in can contain a maximum of `16,380` characters.
- For an error code mapping plug-in, you can configure a maximum of **20** mapping rules by using the `condition` parameter defined in `mappings`.

# 2.1.5.4.3. Bind a plugin to an API

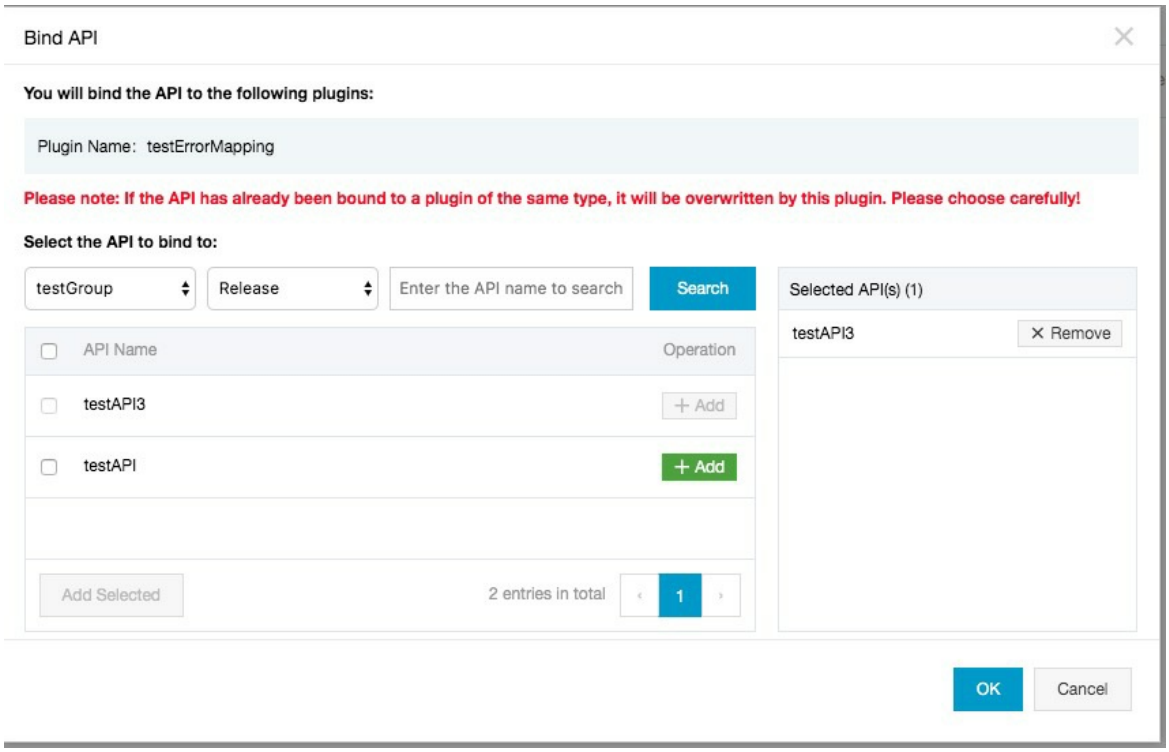
After you create a plugin, you must bind the plugin to an API for the plugin to take effect.

## Context

You can bind a plugin to multiple APIs. The plugin will individually take effect on each API. For each type of plugin, you can bind only one plugin of such type to an API. If you bind two plugins of the same type to an API, the new plugin will replace the previous one and take effect.

## Procedure

1. Log on to the API Gateway console .
2. In the left-side navigation pane, choose **Publish APIs** -> **Plugin**.
3. On the Plugins list page, find the target plugin and click **Bind API** in the Operation column.
4. Select the publish environment and the group of the APIs to which you want to bind a plugin.
5. To bind a plugin to one API, find the target API and click +Add in the Operation column to add the API to the right pane. To bind a plugin to multiple APIs, select the target APIs and click Add Selected in the lower-left corner to add these APIs to the right pane. Then, click **OK**.



# 2.1.5.4.4. Delete a plugin

You can delete existing plugins.

## Procedure

1. Log on to the API Gateway console .
2. In the left-side navigation pane, choose Publish APIs **Plugin**.
3. On the Plugins list page, find the target plugin and click **Delete** in the Operation column.
4. In the Confirm Deletion message, click **OK**.

# 2.1.5.4.5. Unbind a plugin

You can unbind plugins from the APIs to which they are bound.

## Procedure

1. Log on to the API Gateway console .
2. In the left-side navigation pane, choose Publish APIs **Plugin**.
3. On the Plugins list page, click the name of the target plugin to go to the **Create Plugin** page.
4. Click Bound API List. The bound APIs are displayed. Find the target APIs one at a time and click **Unbind** in the Operation column.
5. In the Confirm Unbind message, click **OK**.

# 2.1.6. Manage monitoring

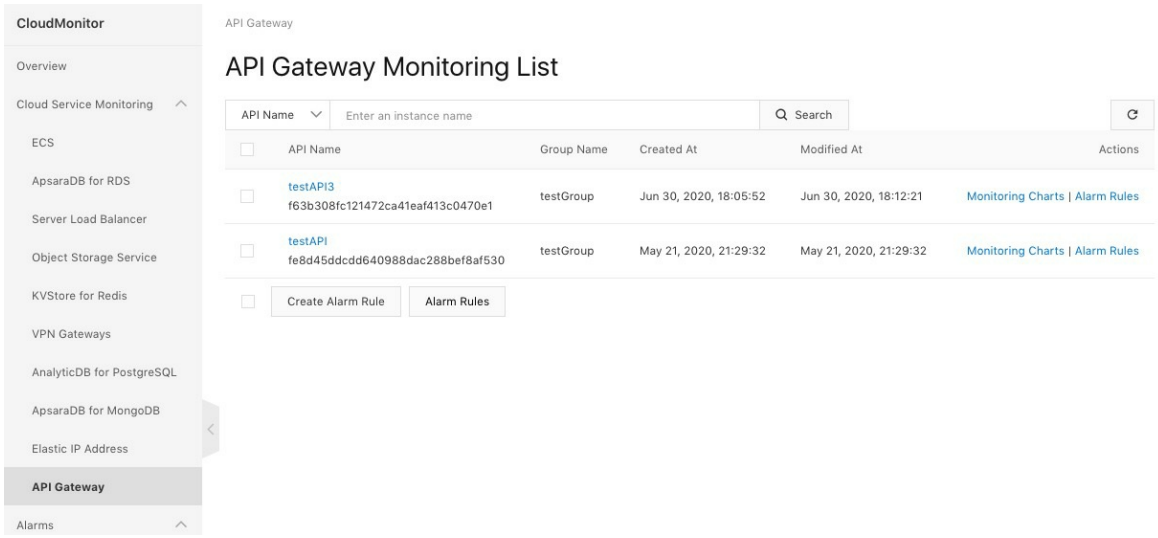
# 2.1.6.1. Use CloudMonitor to view monitoring information and configure alert rules

API Gateway works with CloudMonitor to provide visualized real-time monitoring and alerting features. You can use these features to obtain statistical data about your APIs in multiple dimensions, such as the number of API calls, traffic, backend response time, and error distribution. You can view data in different units of time.

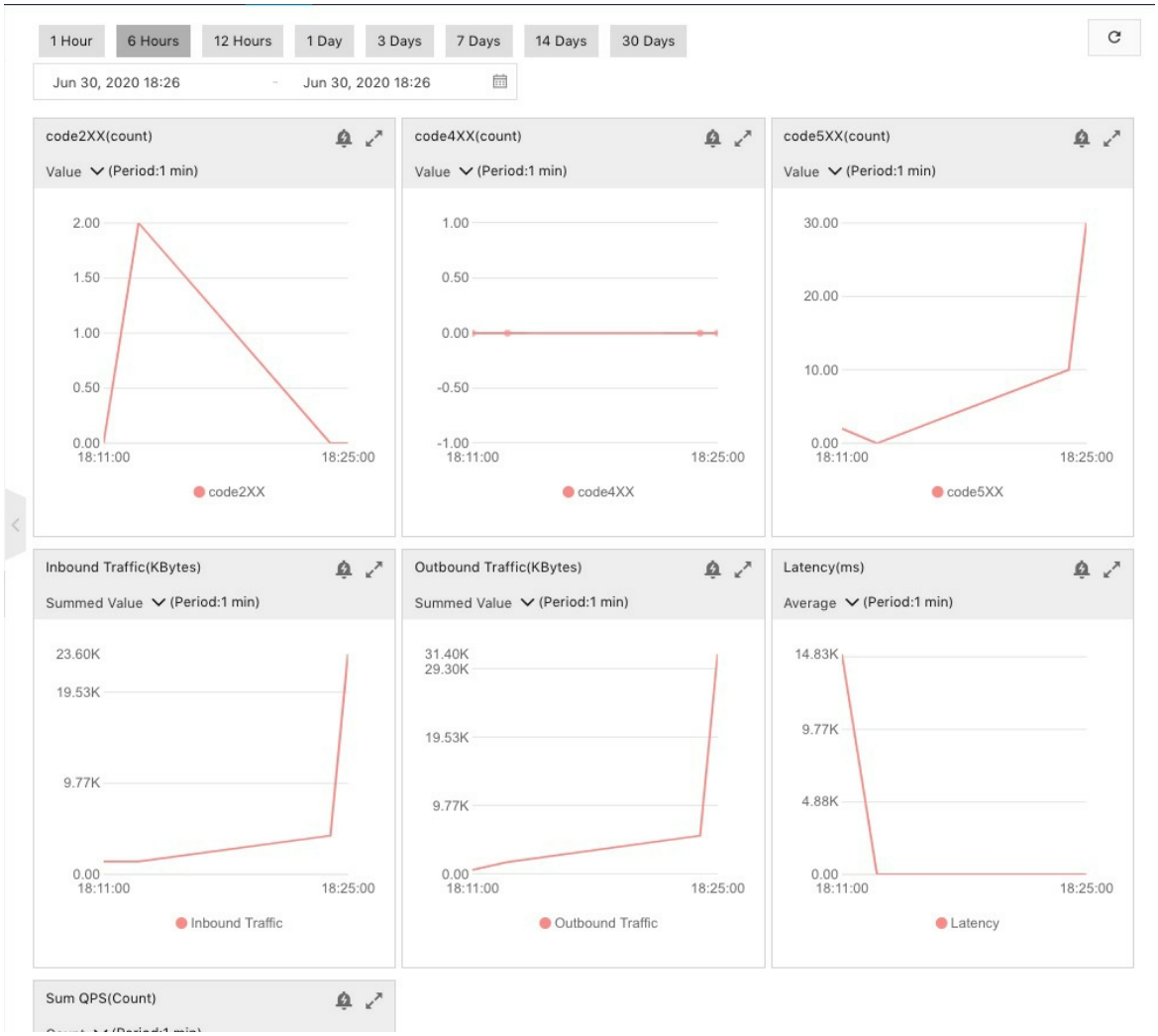
## View monitoring information of API calls

Perform the following steps to view the data of API calls within your Apsara Stack tenant account in the CloudMonitor console.

1. In the top navigation bar of the Apsara Uni-manager Management Console, choose **Products > Monitoring and O&M > Cloud Monitor**.
2. In the CloudMonitor console, choose Cloud Service Monitoring > **API Gateway** in the left-side navigation pane.




3. On the API Gateway Monitoring List page, find your API and click **Monitoring Charts** in the Actions column.




Descriptions of monitoring charts:

- code2XX(count)

Shows the number of requests with a 2XX HTTP status code returned. A 2XX HTTP status code, such as 200, indicates that the request succeeded at the backend.

- code4XX(count)

  Shows the number of requests with a 4XX HTTP status code returned. A 4XX HTTP status code, such as 404, indicates a client error.

- code5XX(count)

  Shows the number of requests with a 5XX HTTP status code returned. A 5XX HTTP status code, such as 500, indicates a server error.

- Inbound Traffic(KBytes)

  Shows the size of API requests received.

- Outbound Traffic(KBytes)

  Shows the size of API responses sent.

- Latency(ms)

  Shows the response time of your backend service. The latency in API Gateway ranges from 3 ms to 5 ms, which is excluded from the response time.

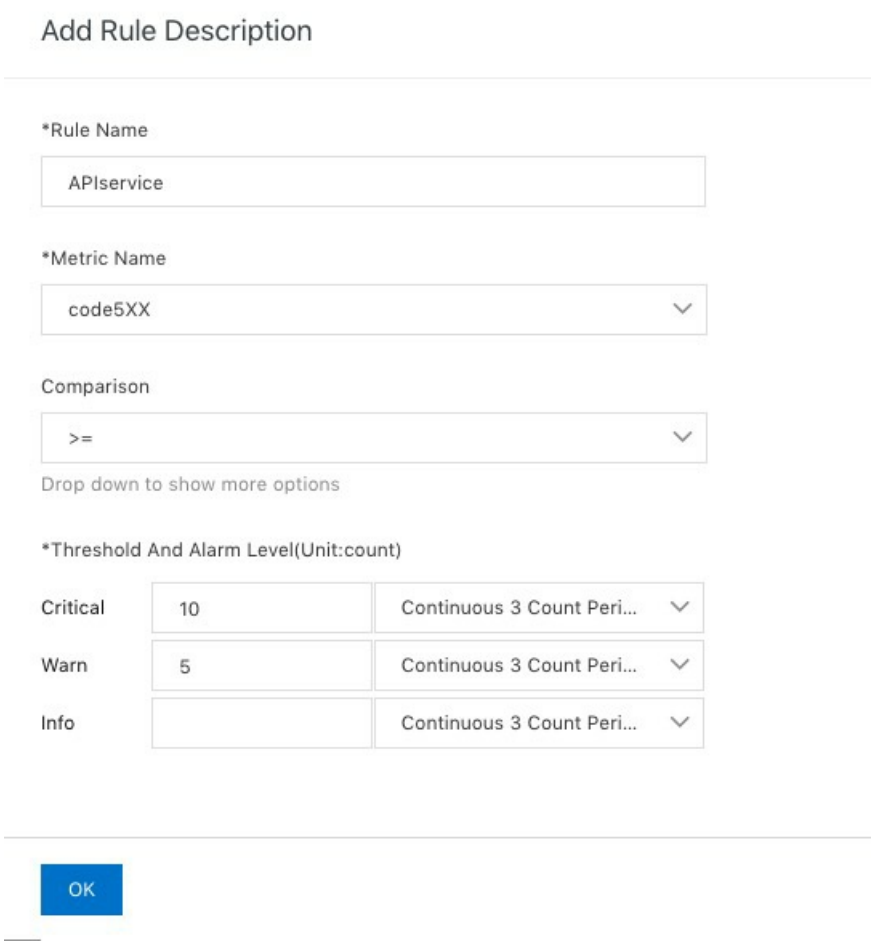
- Sum QPS(Count)

  Shows the total number of API requests.

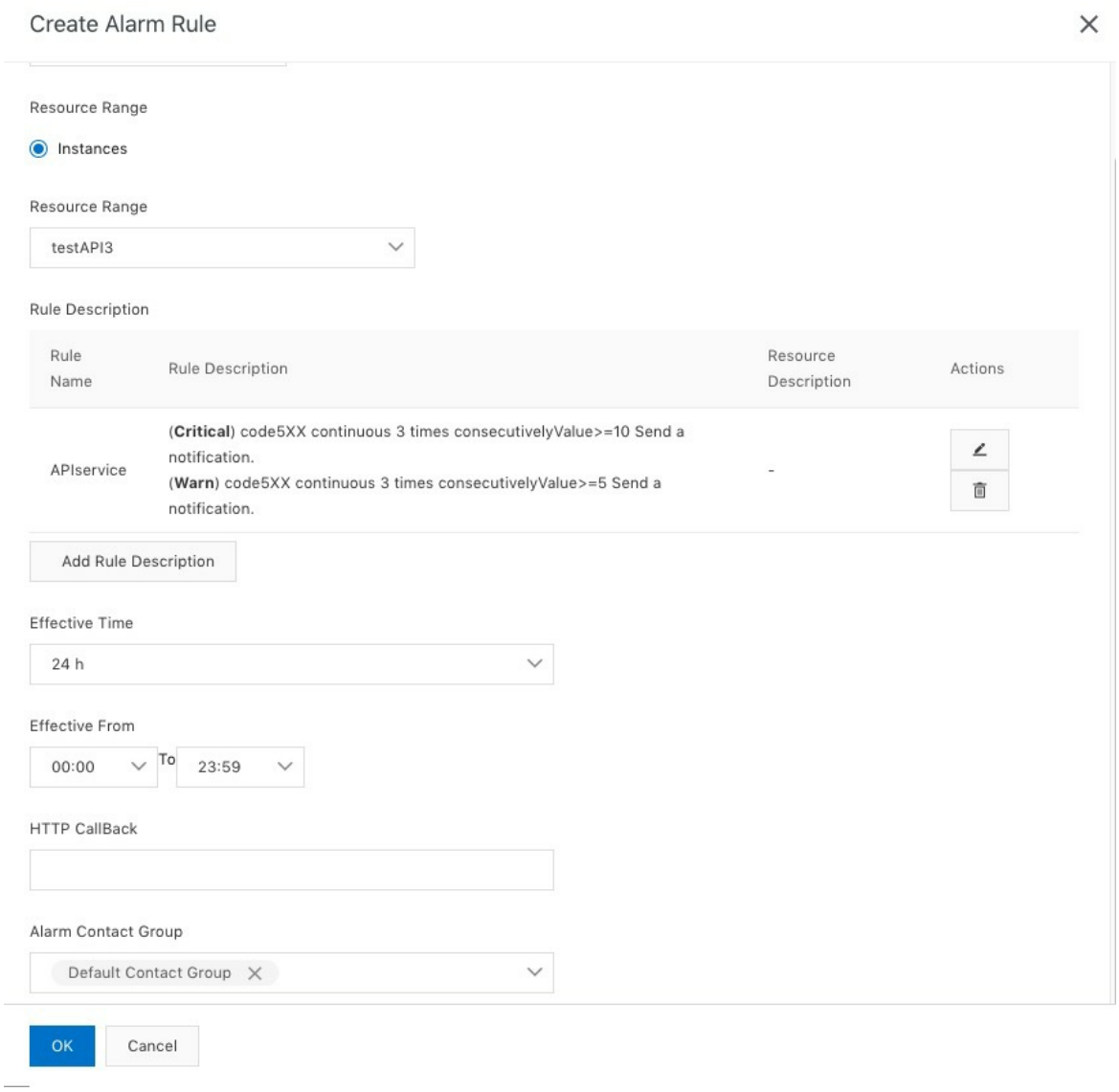
## Configure API alert rules

You can configure API alert rules in the CloudMonitor console to achieve real-time API alerting.

1. On the API Gateway Monitoring List page, find your API.
2. Click **Alert Rules** in the Actions column.
3. On the Alert Rules page, click **Create Alert Rule** in the upper-right corner. In the Modify Alert Rule panel, set Product to API Gateway and Resource Range to the required API.
4. Click **Add Rule Description** and specify Rule Name, Metric Name, Comparison, and Threshold And Alert Level. Then, click **OK**.




5. Specify **Alert Contact Group** and click OK.




> **Note**
>
> To monitor the service status of APIs, we recommend that you monitor code5XX(count).

## 2.1.6.2. View statistical information on the global monitoring page in API Gateway

You can view monitoring information about API calls in the CloudMonitor console. The API Gateway console also provides an overview page for you to view the statistics of API calls. You can view the statistical information about API calls on the global monitoring page.

> **Notice**
>
> On the global monitoring page, you can view only the information about API calls in a specific API group or the calling information about a specific API within your account. Even user root cannot view all the data on the global monitoring page.

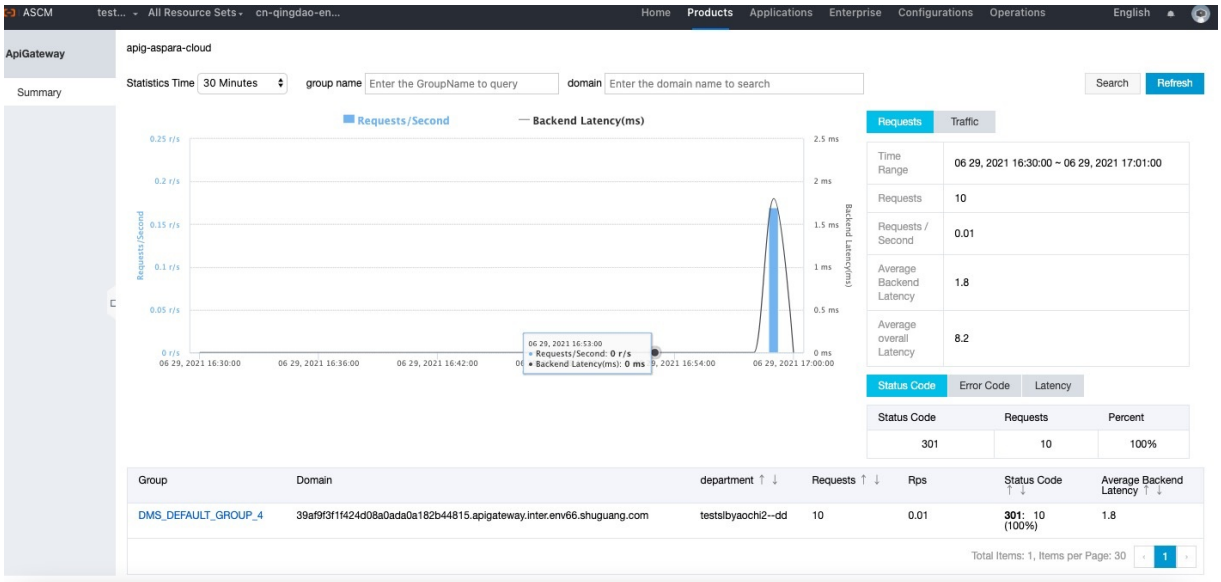
## View overall monitoring information

1. Log on to the API Gateway console. In the left-side navigation pane, click **Instances**. On the Instance list page, find the desired instance and click **Global Monitoring** in the upper-right corner. The summary page appears.
2. Specify Statistics Time and specify group name or domain to view specific information about API calls within your account. By default, the Statistics Time parameter is set to 30 Minutes.

You can view the following metrics:

- Requests: On this tab, you can view the following metrics within your account: Requests, Requests / Second, Average Backend Latency, and Average overall Latency.
- Traffic: On this tab, you can view the following metrics within your account: Request Traffic, Response Traffic, Average Request Size, and Average Response Size.
- Status Code: On this tab, you can view the returned status codes within your account, and the number and percentage of requests with each returned status code.
- Error Code: On this tab, you can view the returned error codes within your account, and the number and percentage of requests with each returned error code.
- Latency: On this tab, you can view the number and percentage of requests within a specific latency range.

In addition, you can view the domain name bound to each API group, and can sort API groups by department, number of requests, requests per second (RPS), status code, or average backend latency.



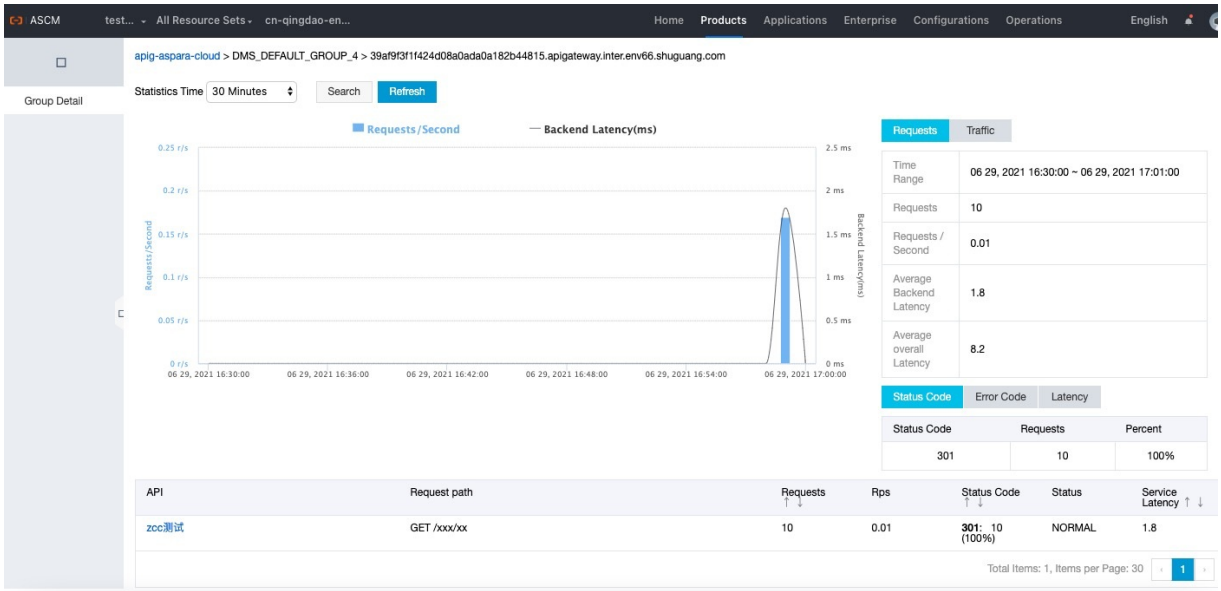

## View monitoring information of a specific API group

On the Summary page, click the name of the desired API group. On the Group Detail page, specify Statistics Time to view specific information about API calls in this API group. By default, the Statistics Time parameter is set to 30 Minutes.

You can view the following metrics:

- Requests: On this tab, you can view the following metrics within your API group: Requests, Requests / Second, Average Backend Latency, and Average overall Latency.
- Traffic: On this tab, you can view the following metrics within your API group: Request Traffic, Response Traffic, Average Request Size, and Average Response Size.
- Status Code: On this tab, you can view the returned status codes within a specified period of time in your API group, and the number and percentage of requests with each returned status code.
- Error Code: On this tab, you can view the returned error codes within a specified period of time in your API group, and the number and percentage of requests with each returned error code.
- Latency: On this tab, you can view the number and percentage of requests within a specific latency range in your API group.

In addition, you can view the request paths of all APIs in your API group, and can sort APIs by number of requests, RPS, status code, or latency.



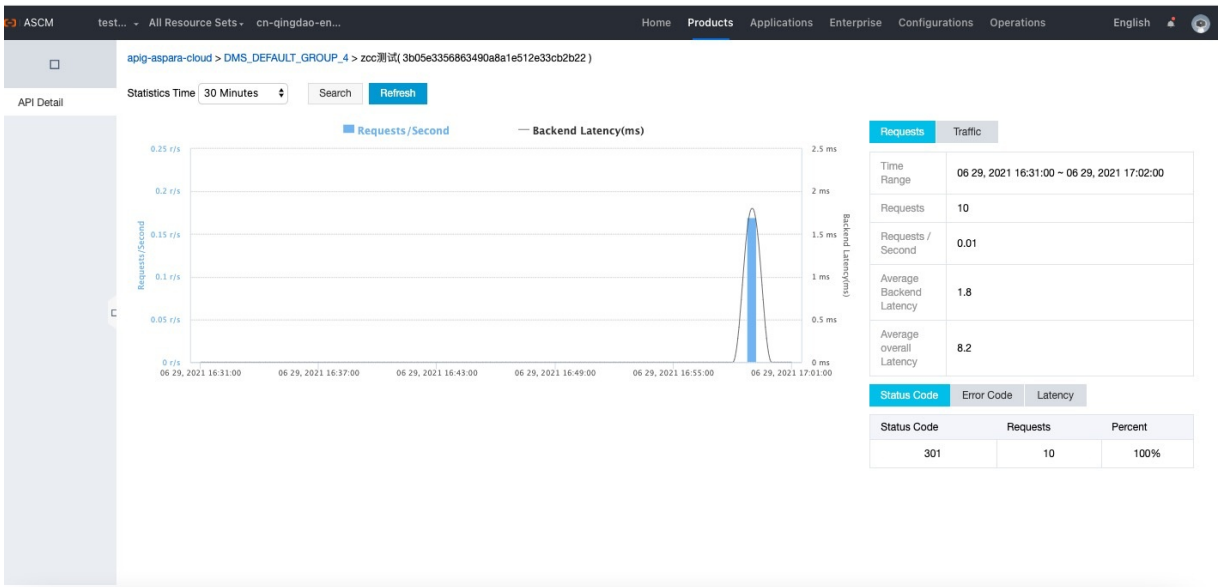

## View monitoring information of a specific API

On the Group Detail page, click the name of the desired API. On the API Detail page, specify Statistics Time to view the specific calling information about the API. By default, the Statistics Time parameter is set to 30 Minutes.

You can view the following metrics:

- Requests: On this tab, you can view the following metrics: Requests, Requests / Second, Average Backend Latency, and Average overall Latency.
- Traffic: On this tab, you can view the following metrics: Request Traffic, Response Traffic, Average Request Size, and Average Response Size.
- Status Code: On this tab, you can view the returned status codes within a specified period of time, and the number and percentage of requests with each returned status code.

- Error Code: On this tab, you can view the returned error codes within a specified period of time, and the number and percentage of requests with each returned error code.
- Latency: On this tab, you can view the number and percentage of requests within a specific latency range.




# 2.1.6.3. Configure an account for global monitoring

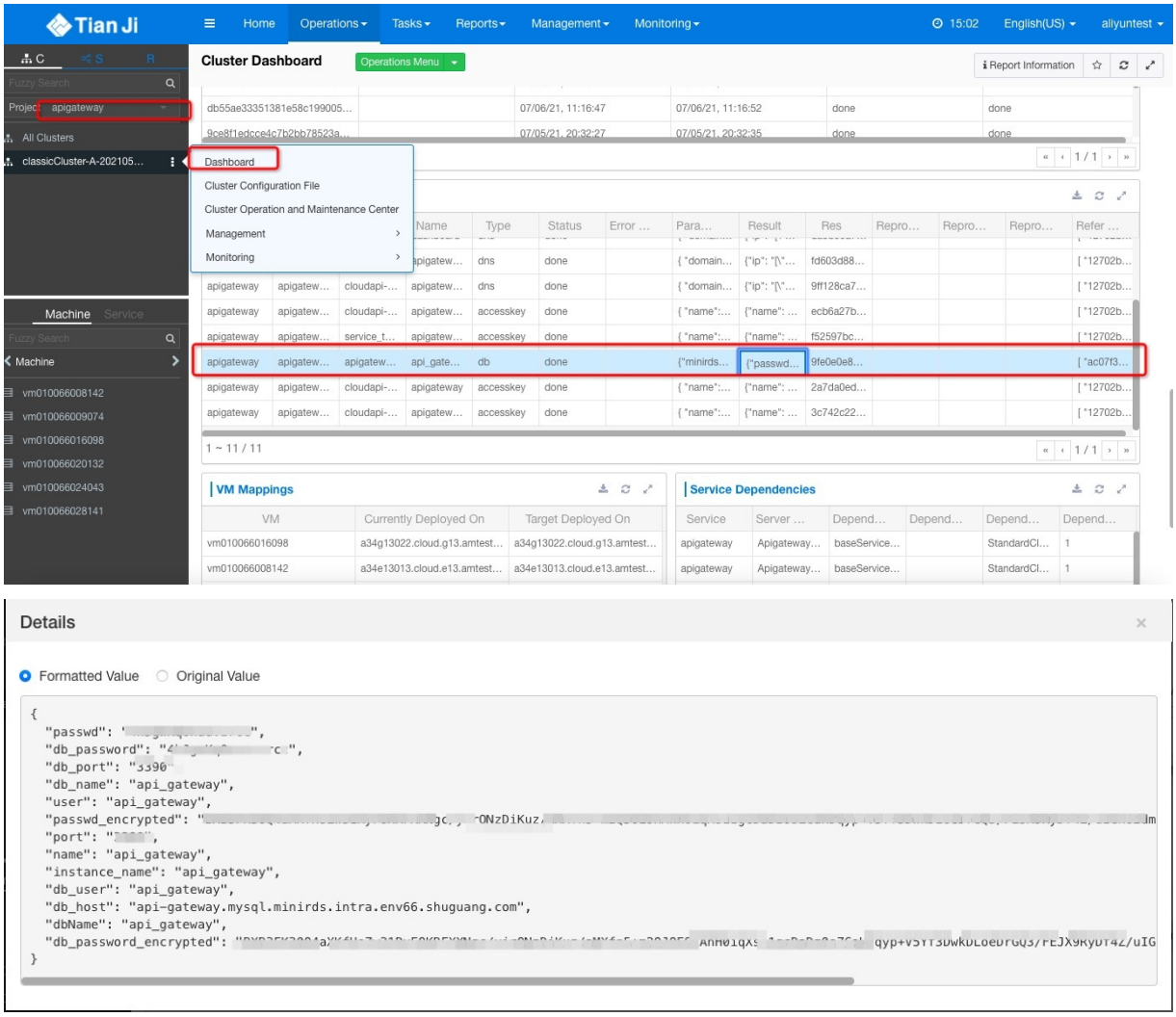
API Gateway provides the global monitoring feature. You can configure a global administrator account that can be used to view the API calls in all departments.

## Configure global monitoring

To configure global monitoring, you must configure a global administrator account first. To configure a global administrator account, you must add the PrimaryKey value of the Level-1 department to which the global administrator account belongs to the desired API Gateway database.

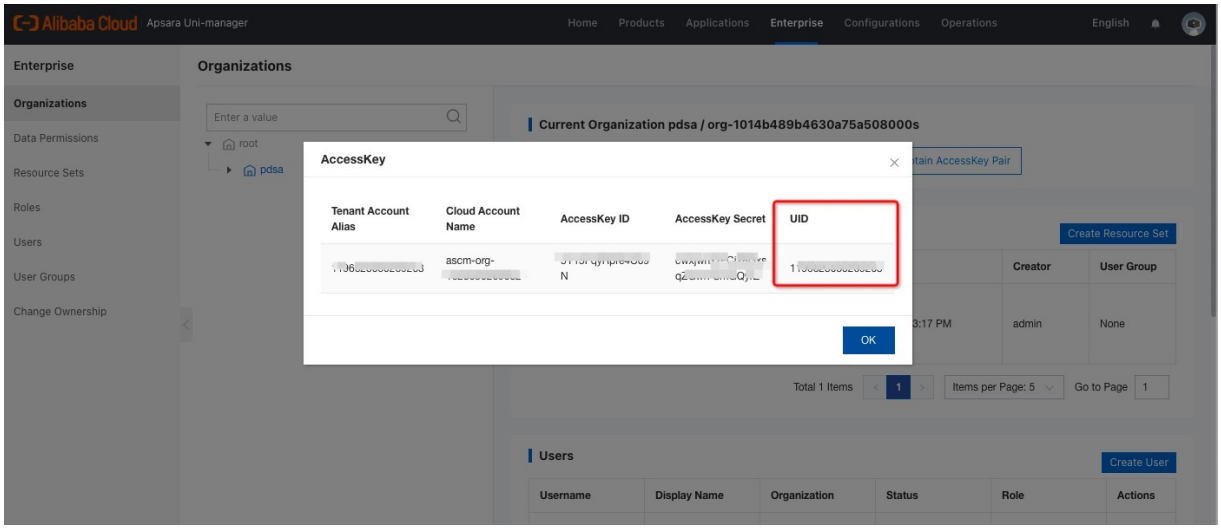
Step 1: Obtain the username and password that are used to access the desired database.

Log on to the Apsara Infrastructure Management Framework console. In the left-side navigation pane, choose Tools > Operation Tools > Machine Tools. The Operation Tools page appears. On the C tab in the left-side navigation pane, select apigateway from the Project drop-down list. Move the pointer over the More icon next to one of the listed clusters and select Dashboard from the shortcut menu. In the Cluster Resource section, find the cluster for which the Type parameter is set to db. Then, right-click the value in the Result column and select Show More from the shortcut menu to view the details of the desired database. The db_user parameter specifies the username that is used to access the database. The db_host parameter specifies the address of the database. The db_password parameter specifies the password that is used to access the database. The db_port parameter specifies the port of the database. The db_name parameter specifies the name of the database.




Step 2: Obtain the PrimaryKey value of the Level-1 department to which the global administrator account belongs.

Log on to the Apsara Uni-manager Management Console. In the top navigation bar, click Enterprise. Select the Level-1 department to which the global administrator account belongs in the left-side navigation pane of the Organizations page and click Obtain AccessKey Pair. In the AccessKey message, view the PrimaryKey value.



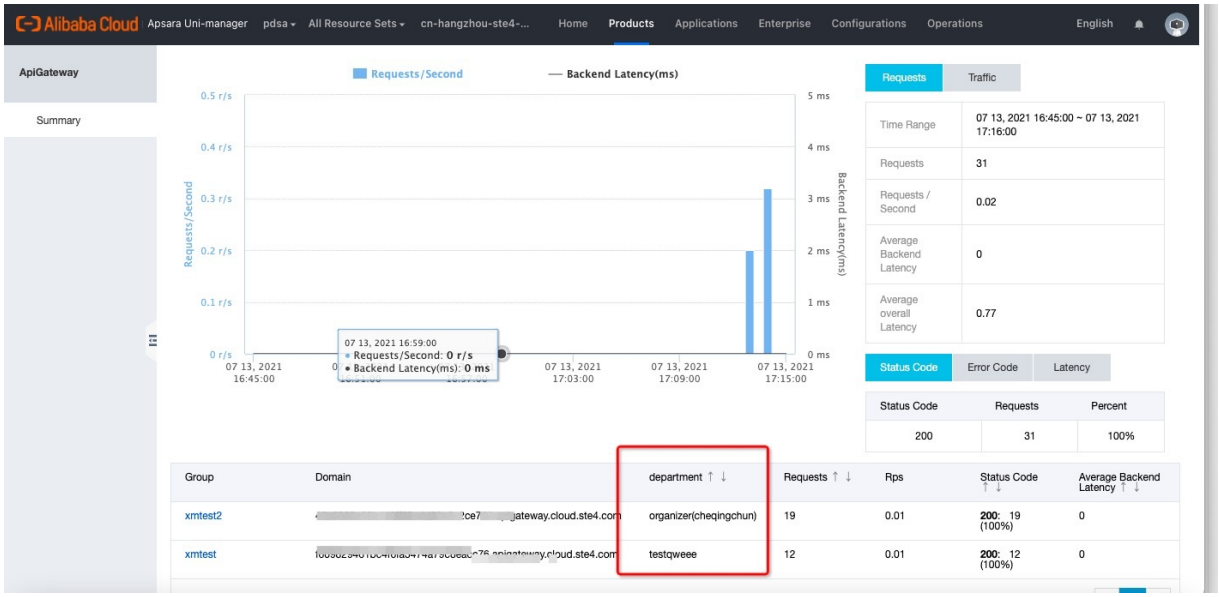

Step 3: Log on to the database to configure the global administrator account.

In the Apsara Infrastructure Management Framework console, log on to the host for which the serverRole parameter is set to apigateway.ApigatewayOpenAPIForClassic. Then, run the following commands to access the desired database and configure the global administrator account based on the information obtained in Step 1 and Step 2.

```
mysql -u api_gateway  -hapi-gateway.mysql.minirds.intra.env66.shuguang.com  -pemSJzawsl70yb
hll  -P 3157        # Log on to the database environment.
use  api_gateway;    # Select the api_gateway database.
INSERT INTO `ca_stastic_config`( `gmt_create` , `gmt_modified` , `config_key` , `config_val
ue` , `is_removed` ) VALUES(NOW(),NOW(),'AdminUidConfig','["xxx"]', 0);          # Replace x
xx with the PrimaryKey value that serves as the uid value to configure the global administr
ator account. You can configure multiple uid values.
UPDATE `ca_stastic_config` SET `config_value` = '["xxx"]' WHERE `config_key` = 'AdminUidCon
fig';               # Update configurations.
```

Step 4: Use the global administrator account to view monitoring details.

Log on to the API Gateway console. Click Instances in the left-side navigation pane. On the Instance list page, find the desired instance and click Global Monitoring. On the Summary page, you can filter data by API group or domain name based on the specified period of time.




**Note: By default, only the monitoring data of API groups in the Level-1 department can be displayed. Only the configured global administrator account can be used to view monitoring data in all departments.**

# 2.1.7. Advanced usage

## 2.1.7.1. Customize business parameters for logs

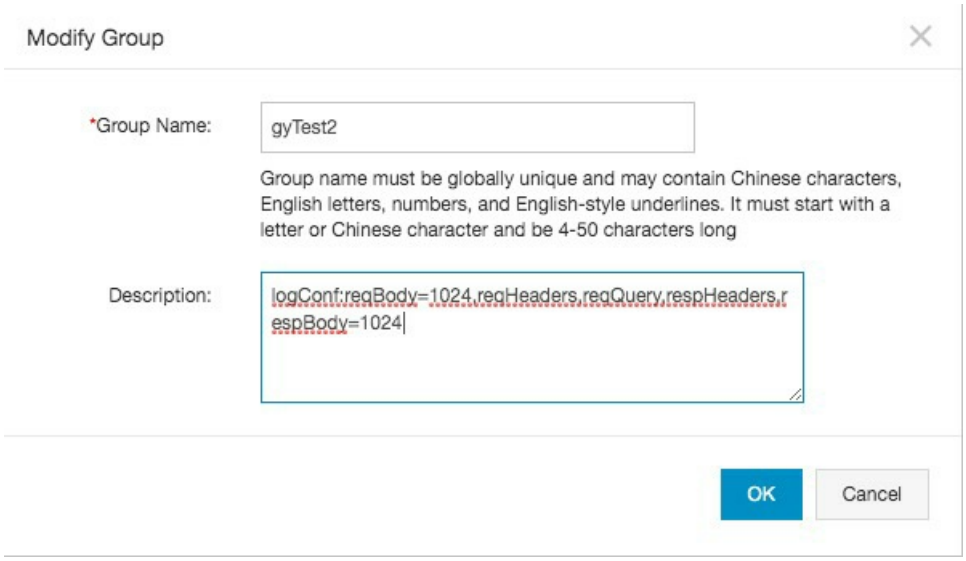
API Gateway provides the Hack mode, which allows you to record the request and response parameters of API calls in logs.

### Procedure

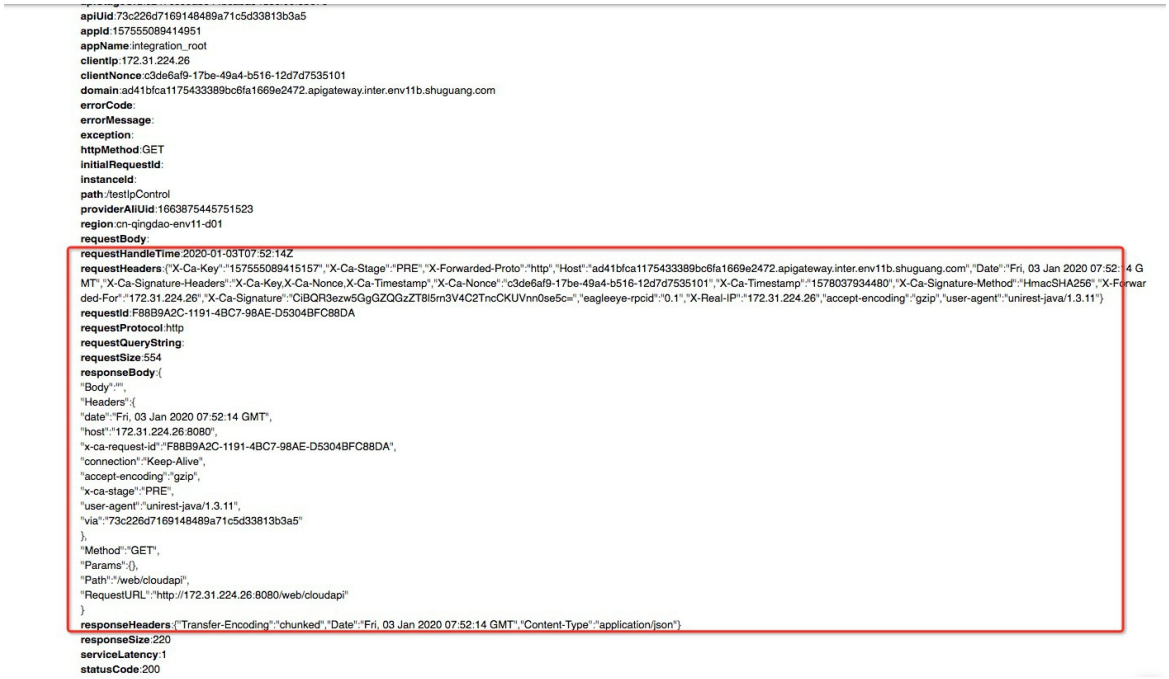
1. Log on to the API Gateway console
2. In the left-side navigation pane, choose **Publish APIs > API Groups**.
3. On the Group List page, find your API group and click its name to go to the Group Details page.
4. Click **Modify Group Message** in the upper-right corner.
5. In the **Description** field of the Modify Group dialog box, add the following content:

```
logConf:reqBody=1024,reqHeaders,reqQuery,respHeaders,respBody=1024
```

> **Note**
>
> - The content must be in a separate line. Otherwise, the configuration fails.
> - You can also modify the content based on your requirements. For example, you can remove respHeaders to omit the response header and its content in the logs.
> - reqBody=1024 indicates that the log records a maximum of 1,024 characters from the request body. Extra characters are discarded.




6. Log on to the Log Service console and check whether the description change has been recorded.




# 2.1.7.2. Configure Log Service logs for API Gateway

# 2.1.7.2.1. Initialize the default Log Service configuration of API Gateway

By default, API call logs of API Gateway are synchronized to Log Service in Apsara Infrastructure Management Framework. However, your account can be activated only after you log on to the Log Service console from Apsara Infrastructure Management Framework.

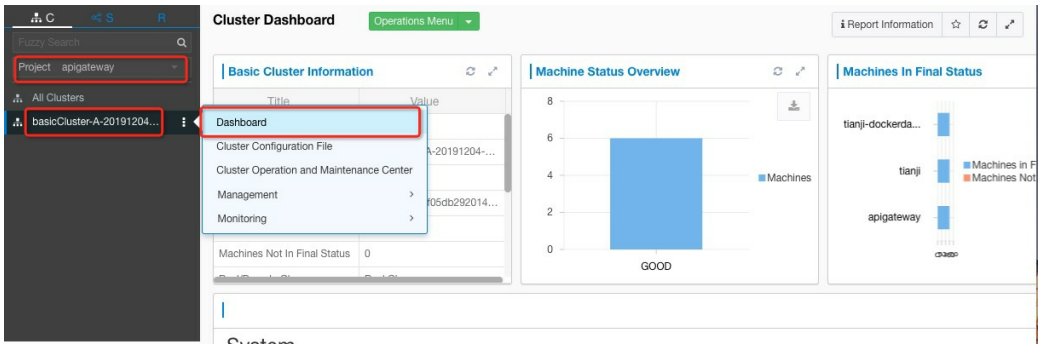
## Procedure

1. Log on to the Apsara Infrastructure Management Framework console. In the left-side navigation

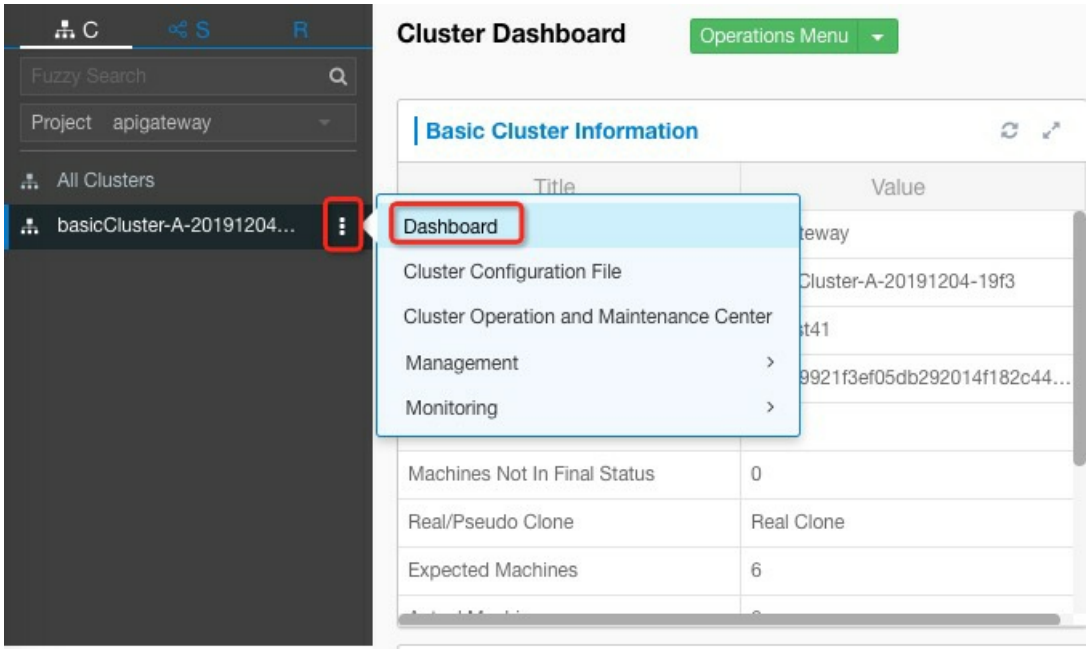
pane, choose Tools > Operation Tools > Machine Tools. The Operation Tools page appears. You can also click Go on the Machine Tools page to go to the Operation Tools page again.

2. Click the **C** tab in the left-side navigation pane. Then, select *apigateway* from the **Project** drop-down list.

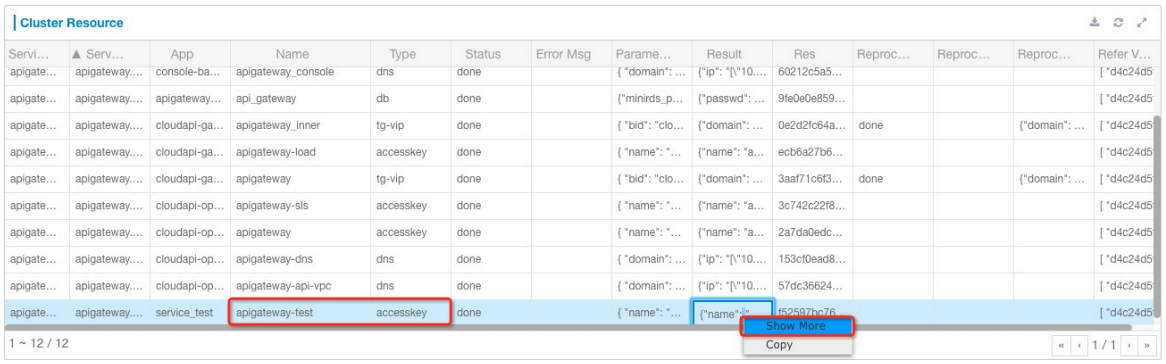
   Cluster O&M page




3. Move the pointer over the icon next to one of the filtered clusters, and select **Dashboard** from the shortcut menu.



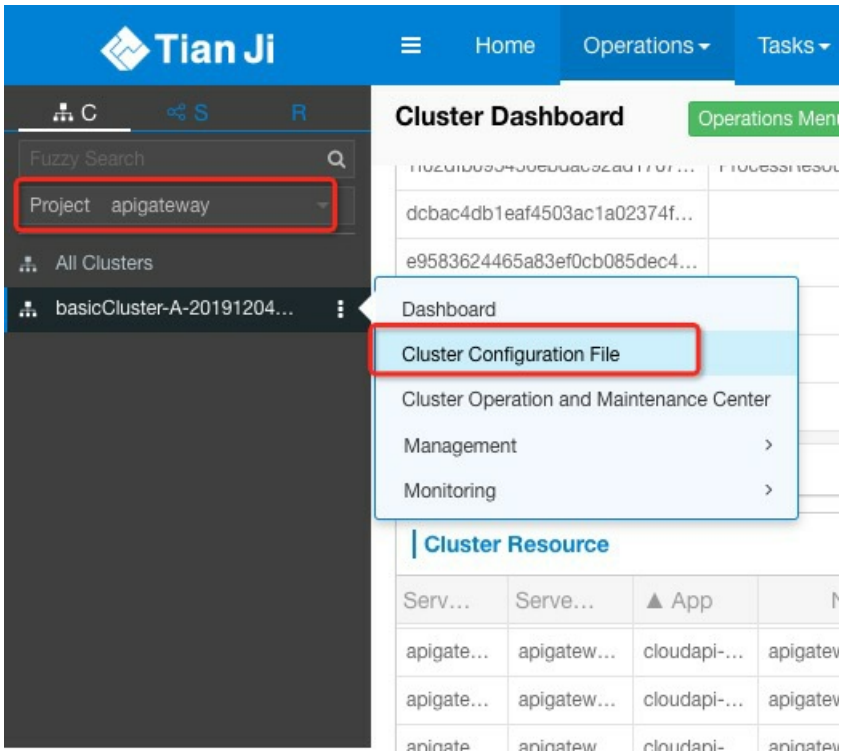

4. In the Cluster Resource section of the page that appears, find the service with Name set to apigateway-sls and Type set to accesskey. Right-click the value in the Result column and select **Show More** from the shortcut menu to view the values of accesskey-id and accesskey-secret.



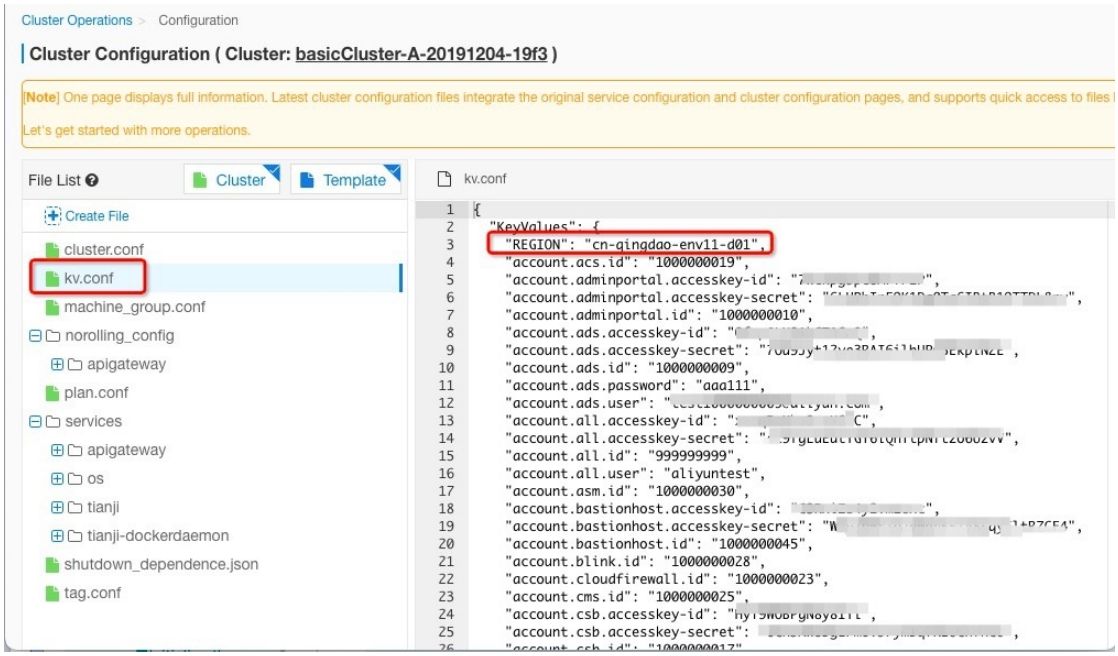

5. Use the accesskey-id and accesskey-secret values that you obtained to log on to the Log Service console from Apsara Infrastructure Management Framework.

   The URL for the Log Service console in Apsara Infrastructure Management Framework is in the following format: http://portal.${region}.sls.${internet-domain}. You can obtain the values of region and internet-domain from the kv.conf file in the Apsara Infrastructure Management Framework console.

   i. Move the pointer over the More icon next to the apigateway cluster and select **Cluster Configuration File** from the shortcut menu.



ii. Click the **kv.conf** file to view the values of region and internet-domain.




**Note** After you log on to the Log Service console, Log Service is automatically configured for API Gateway. This operation takes several minutes.

# 2.1.7.2.2. Configure API Gateway to ship logs to your Log Service project

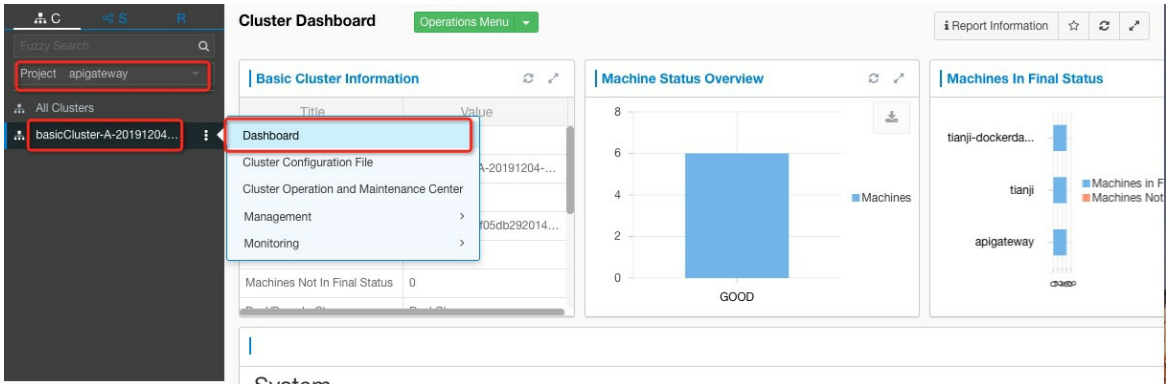
If you do not want to use Log Service and a fixed account in Apsara Infrastructure Management Framework, you can create a Log Service project. Then, you can configure API Gateway to ship logs to your Log Service project.

## Context

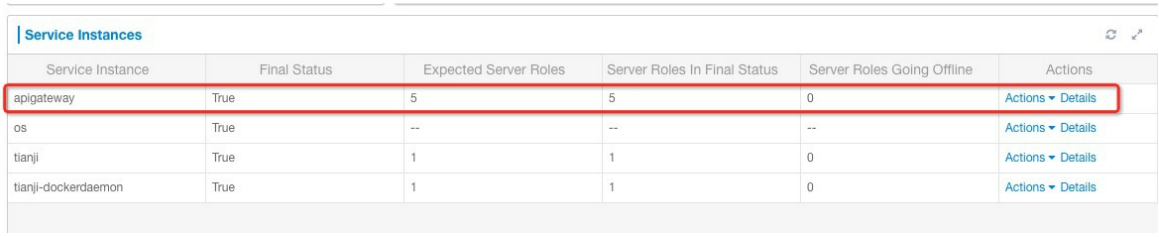
You must perform the following steps to create a Log Service project and configure Logstores and machine groups:

## 1. Log on to the machine where API Gateway resides

1. Log on to the Apsara Infrastructure Management Framework console. In the left-side navigation pane, choose Tools > Operation Tools > Machine Tools. The Operation Tools page appears. Click the C tab in the left-side navigation pane. Then, select apigateway from the Project drop-down list. Move the pointer over the More icon next to one of the filtered clusters, and select Dashboard from the shortcut menu.




2. In the **Service Instances** section of the page that appears, find the row where the value of Service Instance is apigateway.



Service Instances

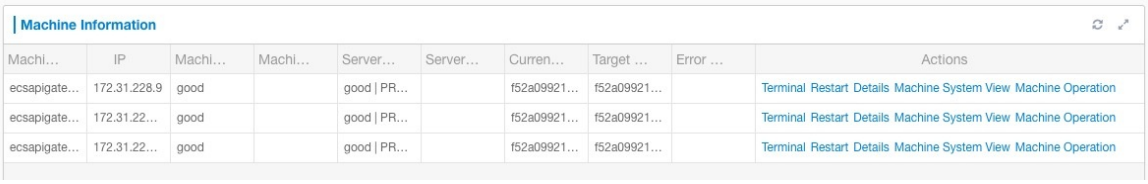
| Service Instance | Final Status | Expected Server Roles | Server Roles In Final Status | Server Roles Going Offline | Actions |
|---|---|---|---|---|---|
| apigateway | True | 5 | 5 | 0 | Actions Details |
| os | True | -- | -- | -- | Actions Details |
| tianji | True | 1 | 1 | 0 | Actions Details |
| tianji-dockerdaemon | True | 1 | 1 | 0 | Actions Details |

3. Click **Details** in the Actions column. In the **Server Role List** section of the page that appears, find the row where the value of Server Role is ApigatewayLite#.

Server Role List

| Server Role | Current Status | Expected Machines | Machines In Final ... | Machines Going ... | Rolling Task Status | Time Used | Actions |
|---|---|---|---|---|---|---|---|
| ApigatewayConsole# | In Final Status | 2 | 2 | 0 | no rolling | | Details |
| ApigatewayDB# | In Final Status | 1 | 1 | 0 | no rolling | | Details |
| ApigatewayLite# | In Final Status | 3 | 3 | 0 | no rolling | | Details |
| ApigatewayOpenAPI# | In Final Status | 2 | 2 | 0 | no rolling | | Details |
| ServiceTest# | In Final Status | 1 | 1 | 0 | no rolling | | Details |

4. Click **Details** in the Actions column. Find the required machine in the **Machine Information** section and click Terminal in the Actions column to log on to the machine.

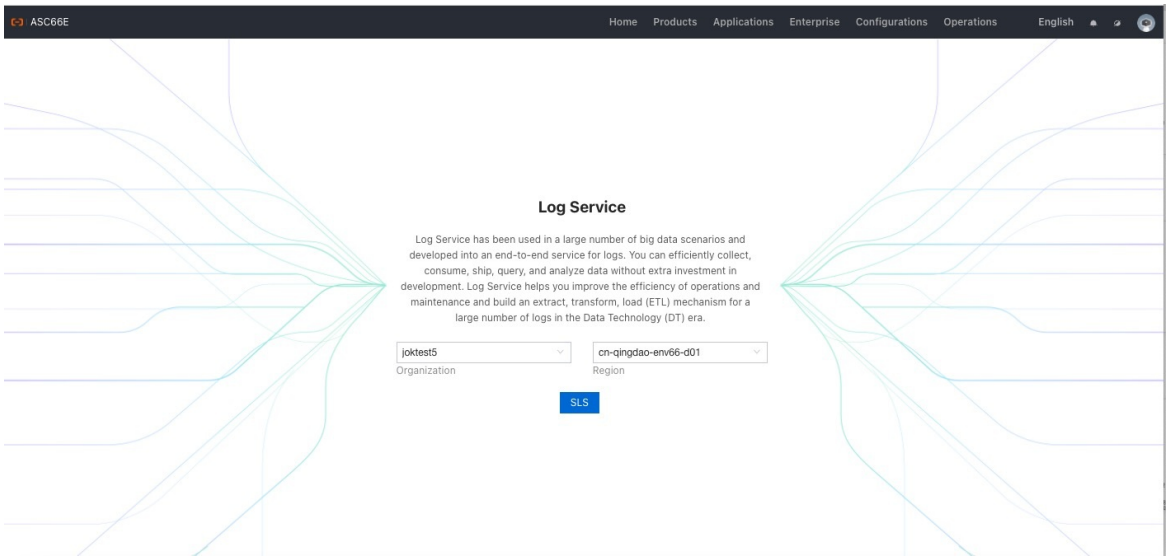


Machine Information

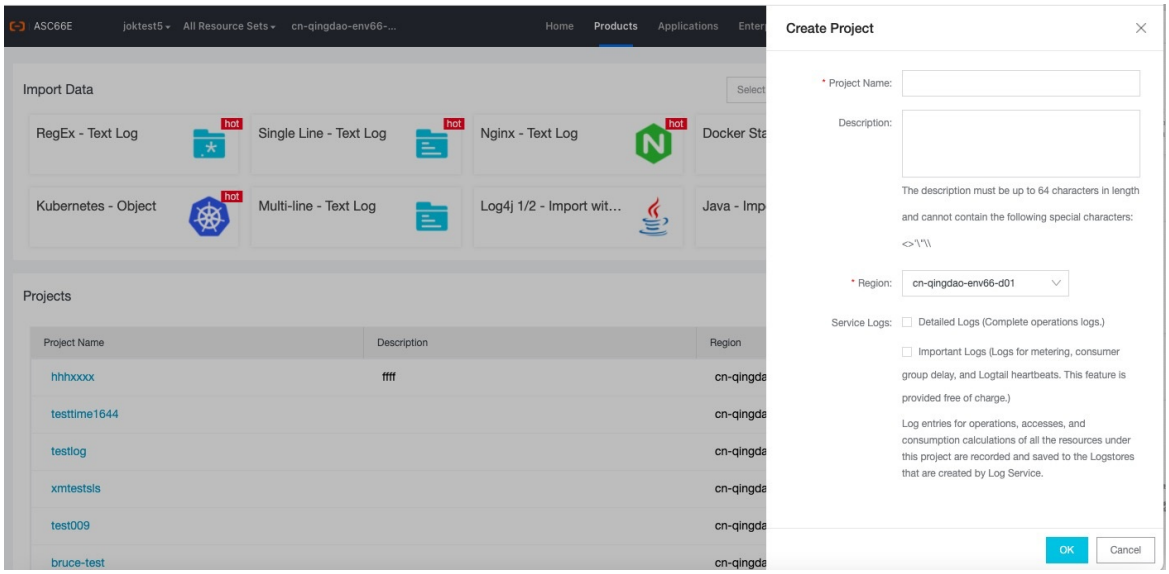
| Machi... | IP | Machi... | Machi... | Server... | Server... | Curren... | Target ... | Error ... | Actions |
|---|---|---|---|---|---|---|---|---|---|
| ecsapigate... | 172.31.228.9 | good | | good \| PR... | | f52a09921... | f52a09921... | | Terminal Restart Details Machine System View Machine Operation |
| ecsapigate... | 172.31.22... | good | | good \| PR... | | f52a09921... | f52a09921... | | Terminal Restart Details Machine System View Machine Operation |
| ecsapigate... | 172.31.22... | good | | good \| PR... | | f52a09921... | f52a09921... | | Terminal Restart Details Machine System View Machine Operation |

## 2. Configure logs in the Log Service console

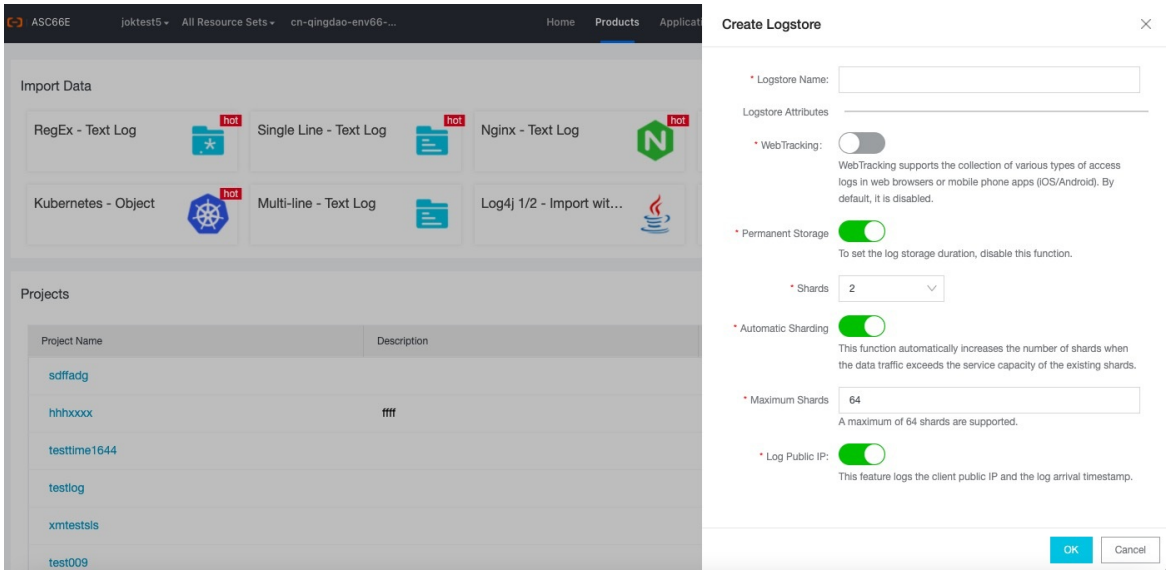
1. Log on to the Apsara Uni-manager Management Console. In the top navigation bar, choose Products —— Application Services —— Log Service. On the page that appears, specify **Region** and **Organization** and click **SLS**.



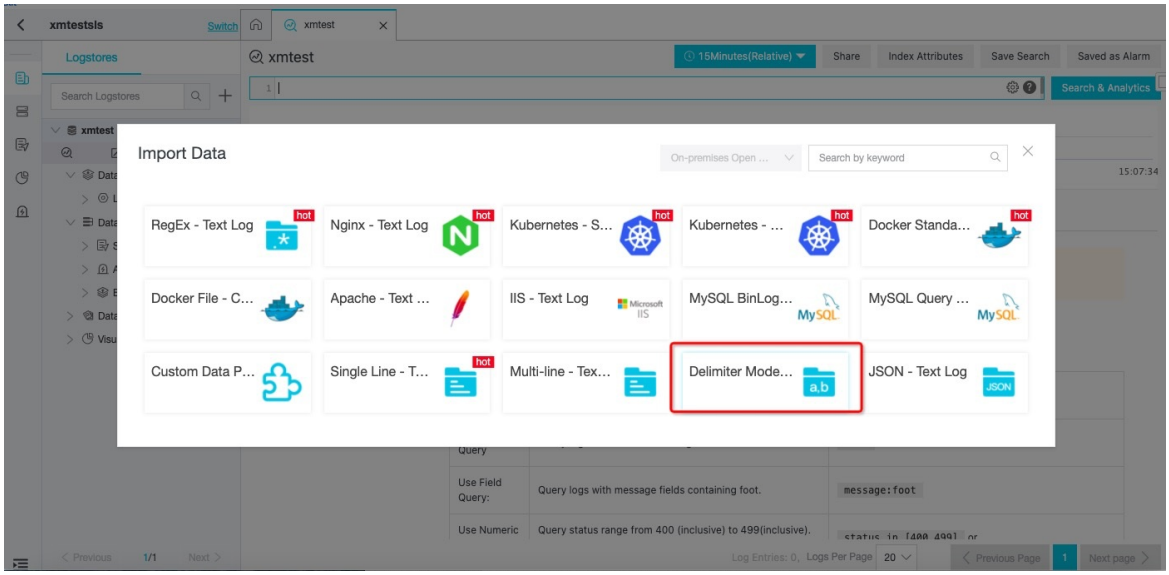

2. Click Create Project. In the Create Project panel, configure parameters and click OK.



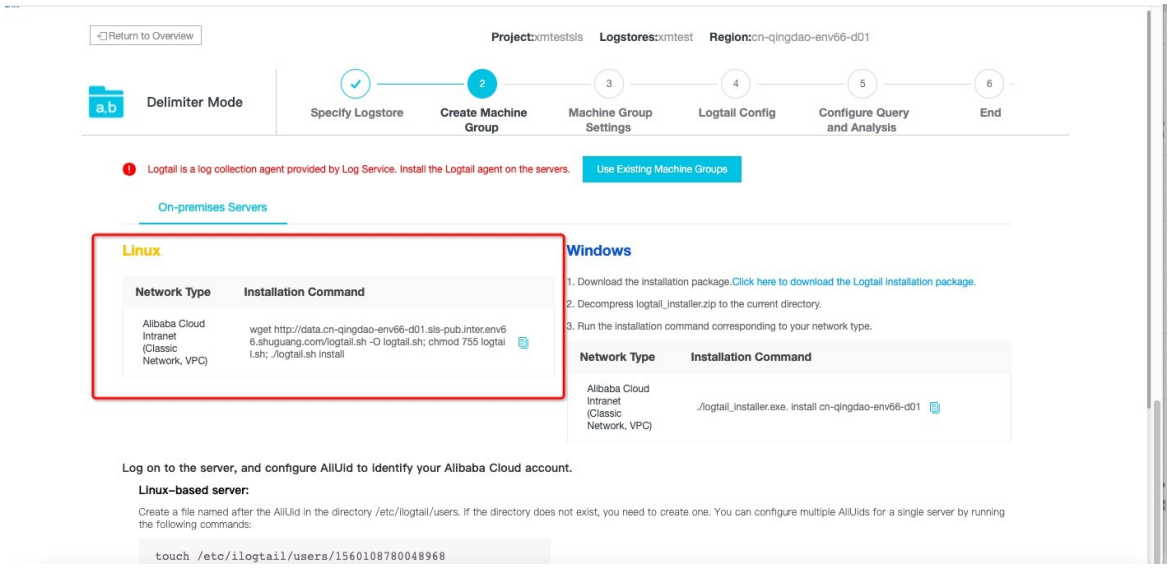

3. In the Created message, click OK. In the Create Logstore panel, configure parameters and click OK to create a Logstore.



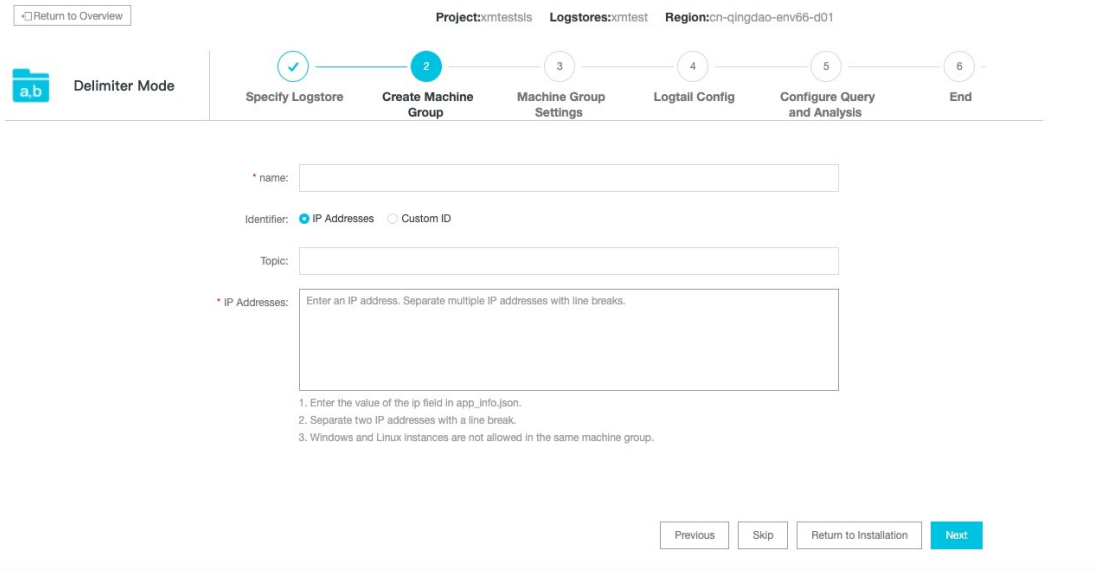

4. In the Created message, click OK. On the Import Data page, click **Delimiter Mode - Text Log**.



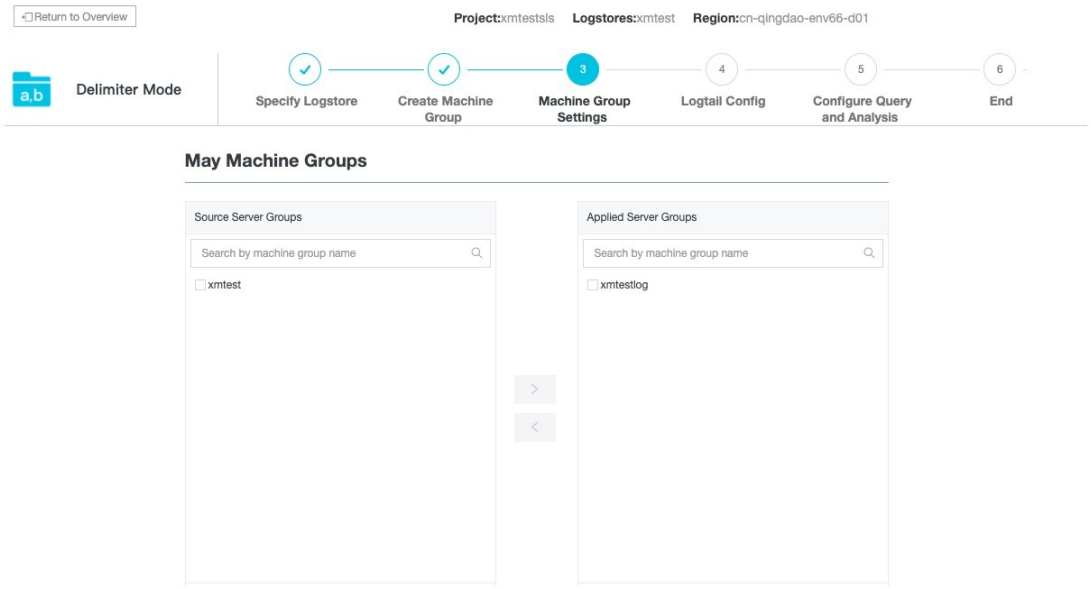

5. Access the container whose service role starts with ApigatewayLite in the Apsara Infrastructure Management Framework console. Then, run the installation command that is displayed in the Create Machine step on the TerminalService page. The operations are similar to those in the first section. If the "install logtail success" message appears, the installation succeeds.




6. After the installation is complete, click Complete Installation and configure parameters in the Create Machine step. Specify name. Set IP Addresses to the IP addresses of all the Docker containers whose service role starts with ApigatewayLite.




7. Click Next to go to the Machine Group step. Configure machine groups and click Next. A heartbeat check automatically starts. If the check succeeds, you are redirected to the Logtail Config step.

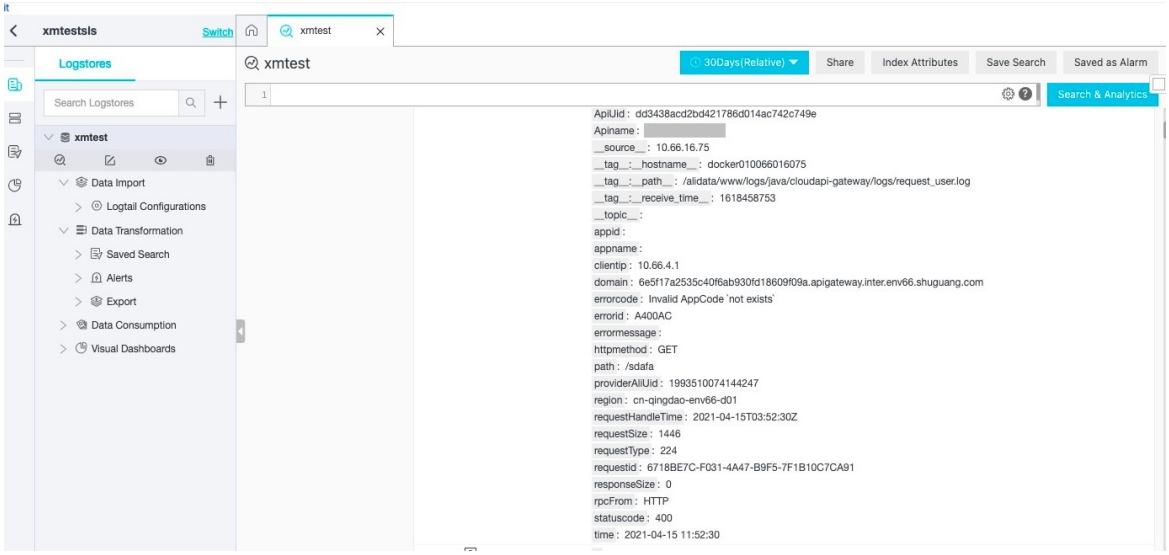



8. In the Logtail Config step, configure the parameters that are described in the following table and click **Next**.

| Parameter | Value |
| --- | --- |
| Config Name | gateway_request_log |
| Log Path | /alidata/www/logs/java/cloudapi-gateway/logs/request_user.log |
| Mode | Delimiter Mode |
| Log Sample | 2019-08-15 14:25:05\|CC7C526B-C915-44F1-93A2-F44DBBE35177\|b2909c9fa66146f19baf2bd8f4709ab8\|integration_root\|e4032f87ace14cc6965cf5352a9637a6\|RELEASE\|0619f7763b004fc3a16a41dd712ce8d7\|biz1_anonymous\|10.4.21.241\|\|\|b2909c9fa66146f19baf2bd8f4709ab8.apigateway.env4b.shuguang.com\|POST\|/biz1/anonymous\|403\|A403JT:Invalid JWT: deserialized JWT failed\|\|1453964555641148\|cn-qingdao-env4b-d01\|2019-08-15T06:25:05Z\|614\|0\|0\|A403JT\|http\|\|\|cf802da1-54d0-49b8-b77c-2b20b172d90a\|\|{"X-JWT-Token":"bad jwt token"}\|a=%21111\|\|\|\|\|\| |

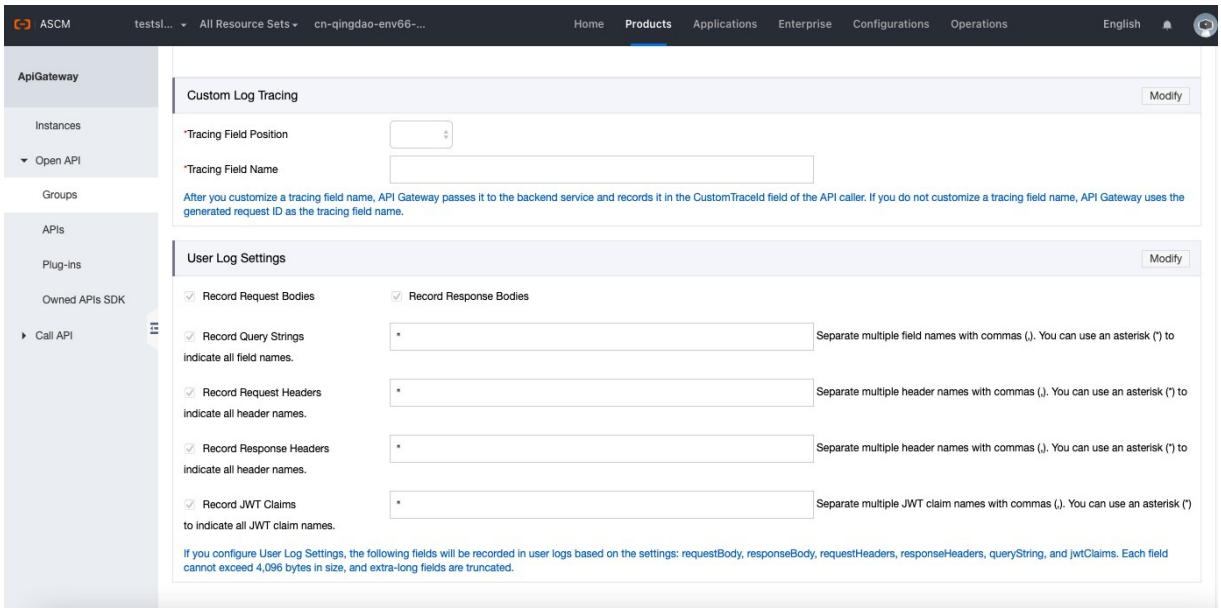
| Parameter | Value |
| --- | --- |
| Delimiter | Vertical Line |

9. Use the default settings until the entire configuration process is complete.
10. After the configuration is complete, make some API calls in the API Gateway console and check whether the configuration takes effect in the Log Service console.



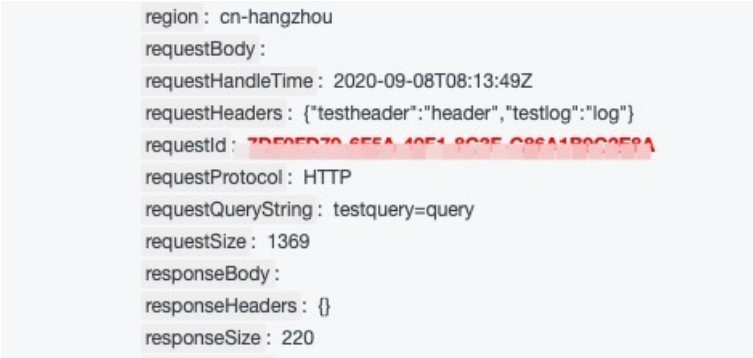

# 2.1.7.2.3. Configure the logging of HTTP requests and responses

If you want API Gateway to record the HTTP requests it receives and the HTTP responses it returns, you can perform the operations described in this topic.



- Record Request Headers: Separate the names of the request headers that you want to record with commas (,). You can use an asterisk (*) to indicate all header names.
- Record Response Headers: Separate the names of the response headers that you want to record with commas (,). You can use an asterisk (*) to indicate all header names.
- Record Query Strings: Separate the names of the fields that you want to record with commas (,). You can use an asterisk (*) to indicate all field names.
- Record JWT Claims: Separate the names of the JWT claims that you want to record with commas (,). You can use an asterisk (*) to indicate all JWT claim names.

Then, you can view the related information in logs. The following figure shows a log.




After the preceding log settings are configured, the system records the following fields in logs: requestBody, responseBody, requestHeaders, responseHeaders, queryString, and jwtClaims. The size of each field must be no more than 4,096 bytes. If the size of a field exceeds this limit, the system truncates the field before it is recorded.

# 2.1.7.3. Cross-user VPC authorization

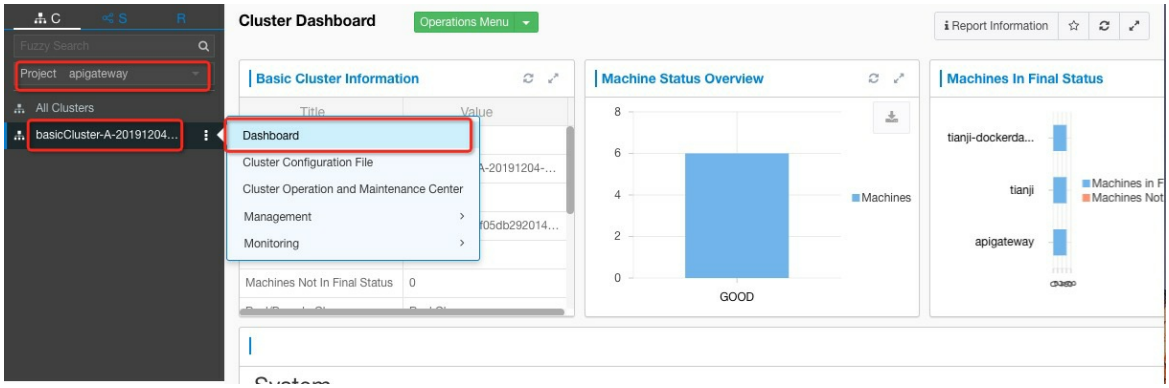
If your backend service resides in a VPC, you must configure the backend service address through VPC authorization. By default, a VPC owner must be the same user as an API owner. Starting from Apsara Stack V3.8.1, API Gateway provides two internal APIs that allow VPC owners to authorize their VPCs to other users.

# 2.1.7.3.1. User authorization across VPCs

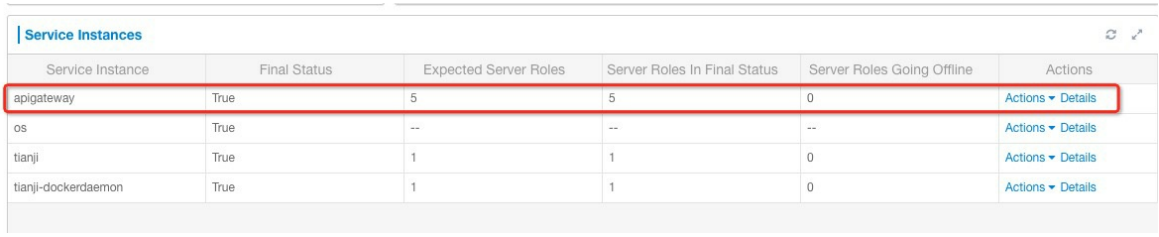
APIs can be used across multiple VPCs. For security reasons, VPC owners must call APIs to explicitly authorize access to VPCs before API providers can use the VPCs. The OpenAPI component of API Gateway has a built-in Aliyun CLI tool. You can use this tool to authorize access to VPCs. The following steps describe how to call the API:

## Procedure

1. Log on to the Apsara Infrastructure Management Framework console. In the left-side navigation pane, click the C tab and select apigateway from the Project drop-down list. Place the pointer over the More icon next to one of the filtered clusters and choose Dashboard from the shortcut menu.




2. On the dashboard page, go to **Service Instance List**, and find apigateway in the Service Instance column.

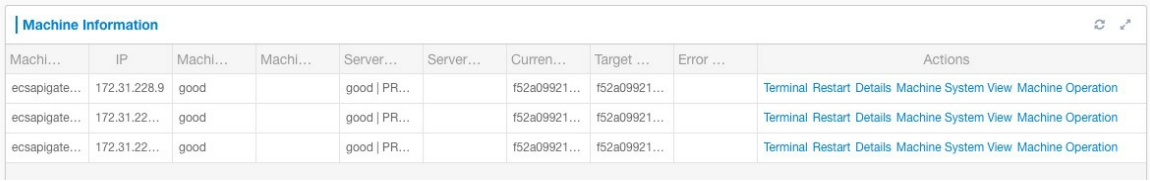


| Service Instance | Final Status | Expected Server Roles | Server Roles In Final Status | Server Roles Going Offline | Actions |
|---|---|---|---|---|---|
| apigateway | True | 5 | 5 | 0 | Actions Details |
| os | True | -- | -- | -- | Actions Details |
| tianji | True | 1 | 1 | 0 | Actions Details |
| tianji-dockerdaemon | True | 1 | 1 | 0 | Actions Details |

3. Click **Details** in the Actions column corresponding to the apigateway service instance. In the **Server Role List** section, find ApigatewayLite# in the Server Role column.

| Server Role | Current Status | Expected Machines | Machines In Final ... | Machines Going ... | Rolling Task Status | Time Used | Actions |
|---|---|---|---|---|---|---|---|
| ApigatewayConsole# | In Final Status | 2 | 2 | 0 | no rolling | | Details |
| ApigatewayDB# | In Final Status | 1 | 1 | 0 | no rolling | | Details |
| ApigatewayLite# | In Final Status | 3 | 3 | 0 | no rolling | | Details |
| ApigatewayOpenAPI# | In Final Status | 2 | 2 | 0 | no rolling | | Details |
| ServiceTest# | In Final Status | 1 | 1 | 0 | no rolling | | Details |

4. Click **Details** in the Actions column corresponding to the ApigatewayLite# server role. Then, find the **Machine Information** section.



| Machi... | IP | Machi... | Machi... | Server... | Server... | Curren... | Target ... | Error ... | Actions |
|---|---|---|---|---|---|---|---|---|---|
| ecsapigate... | 172.31.228.9 | good | | good \| PR... | | f52a09921... | f52a09921... | | Terminal Restart Details Machine System View Machine Operation |
| ecsapigate... | 172.31.22... | good | | good \| PR... | | f52a09921... | f52a09921... | | Terminal Restart Details Machine System View Machine Operation |
| ecsapigate... | 172.31.22... | good | | good \| PR... | | f52a09921... | f52a09921... | | Terminal Restart Details Machine System View Machine Operation |

5. Click Terminal in the Actions column corresponding to a machine in the server role to access the container.
6. Configure the AccessKey pair used to call the CLI tool. Run the following commands:

```
aliyun configure -- profile vpctest //Add the AccessKey pair configuration. vpctest is
the profile name, which can be customized. After you press Enter, configure AccessKeyId
and AccessSecret as prompted.
aliyun configure list //View the config configuration to check whether the preceding pr
ofile has been added.
```

7. Run the following command to perform authorization:

```
aliyun cloudapi AuthorizeVpc --VpcId vpc-tb5mfcwx3s4zqctzw**** --TargetUserId 147546214
349****
--endpoint apigateway.cn-qingdao-env11-d01.inter.env11b.shuguang.com --force
--profile vpctest
```

> **Note**
>
> - Log on to the Apsara Infrastructure Management Framework console. In the top navigation bar, choose **Reports** > **System Reports**. On the System Reports page, click Registration Vars of Services. On the Registration Vars of Services report, right-click the value in the Service Registration column corresponding to the apigateway service and choose Show More from the shortcut menu. The value of the apigateway.openapi.endpoint variable must be used as the endpoint in the preceding command. The profile value also needs to be replaced based on actual needs.
> - This command authorizes the user whose ID is 1475462143497330 to use the VPC with the ID vpc-tb5mfcwx3s4zqctzw19w2. You can replace the parameter values as needed.

Success Operation Sample

```
[root@docker010011102023 ~]
#aliyun cloudapi AuthorizeVpc --VpcId [redacted] --TargetUserId 1475462143497330 --endpoint apigateway.cn-qingdao-env
11-d01.inter.env11b.shuguang.com --force --profile vpctest
{"RequestId":"8E64BC51-60D7-4D85-B5F0-3E3F12C4B6D2"}
```

# 2.1.7.3.2. Configure APIs

After an app is authorized to call an API, the API owner must configure the API and define the API backend service. This is because you cannot select the VPC ID of other users in the Apsara Uni-manager Management Console.

## Context

When you configure an API, take note of the following points:

- Backend Service Type cannot be set to VPC.
- The backend service address must be in the http(s)://{Backend service IP address}.{vpcId}.gateway.vpc:{port} format. The content in {} can be substituted as required. Example:

```
http://192.168.XX.XX.vpc-tb5mfcwx3s4zqctzw****.gateway.vpc:8080
```

## Procedure

1. Log on to the API Gateway console
2. In the left-side navigation pane, choose **Publish APIs** > **APIs**.
3. On the API List page, find your API and perform the following operations:
   - Click the name of the API to go to the **API Definition** page. You can view information about the API.
   - Click **Edit** in the upper-right corner, modify configurations as required, and then click **Save**.

The procedure of creating an API is similar to that of modifying an API. For more information about how to create an API, see Create an API. If you want to cancel the modifications before the modifications are submitted, click Cancel Edit in the upper-right corner of the edit page.

# 2.1.7.4. Call an API over HTTPS

## Context

API Gateway locates a unique API group by domain name, and locates a unique API in the API group by using Path and HTTPMethod. API Gateway assigns a second-level domain for each API group. You can use the domain name to call APIs that belong to the API group. The second-level domain supports only access over HTTP. You can also use a custom domain name to call APIs. This topic describes how to call APIs by using a second-level domain or by using a custom domain name.

## Use a second-level domain to call APIs over HTTPS

To use a second-level domain to call APIs over HTTPS, you must perform the following steps to configure a wildcard domain name certificate in Apsara Stack. You must prepare the certificate yourself.

1. Prepare configuration files for a second-level domain.

   Modify configurations in the following code: Replace *.wildcard.com with your wildcard domain name *.apigateway.${internet-domain}. You can obtain the value of the ${internet-domain} variable in the kv.conf configuration file for Apsara Infrastructure Management Framework. For example, if the domain name that you use to provide external services is abc.alibaba.com, replace *.wildcard.com with *.alibaba.com. Save the modified code to a wildcard.conf file. Set the name of the public key file in the certificate to wildcard.crt. Set the name of the private key file in the certificate to wildcard.key.

   ```
   server {
       listen              443 http2 ssl;
       server_name         *.wildcard.com;
       limit_req           zone=perserver_req burst=100;
       ssl_protocols       TLSv1 TLSv1.1 TLSv1.2;
       ssl_ciphers         ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-SHA:AES128-GCM-SHA
   256:AES128-SHA256:AES128-SHA:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-
   RSA-AES256-SHA:AES256-GCM-SHA384:AES256-SHA256:AES256-SHA:ECDHE-RSA-AES128-SHA256:!aNUL
   L:!eNULL:!RC4:!EXPORT:!DES:!3DES:!MD5:!DSS:!PKS;
       ssl_certificate     /home/admin/cai/certs/wildcard.crt;
       ssl_certificate_key /home/admin/cai/certs/wildcard.key;
       ssl_session_cache   shared:SSL:10m;
       ssl_session_timeout 10m;
       location / {
           proxy_pass http://gateway_upstream;
       }
   }
   ```

2. Place the files at specified locations.

   Copy and paste the wildcard.conf file to the /alidata/sites/conf directory of the host for apigateway.ApiGatewayLite#. Copy and paste the wildcard.crt and wildcard.key files to the /alidata/sites/certs directory of the host for apigateway.ApiGatewayLite#.

3. Activate the certificate.

   Run `/home/admin/cai/bin/nginxctl reload` in the apigateway.ApiGatewayLite# SR container.

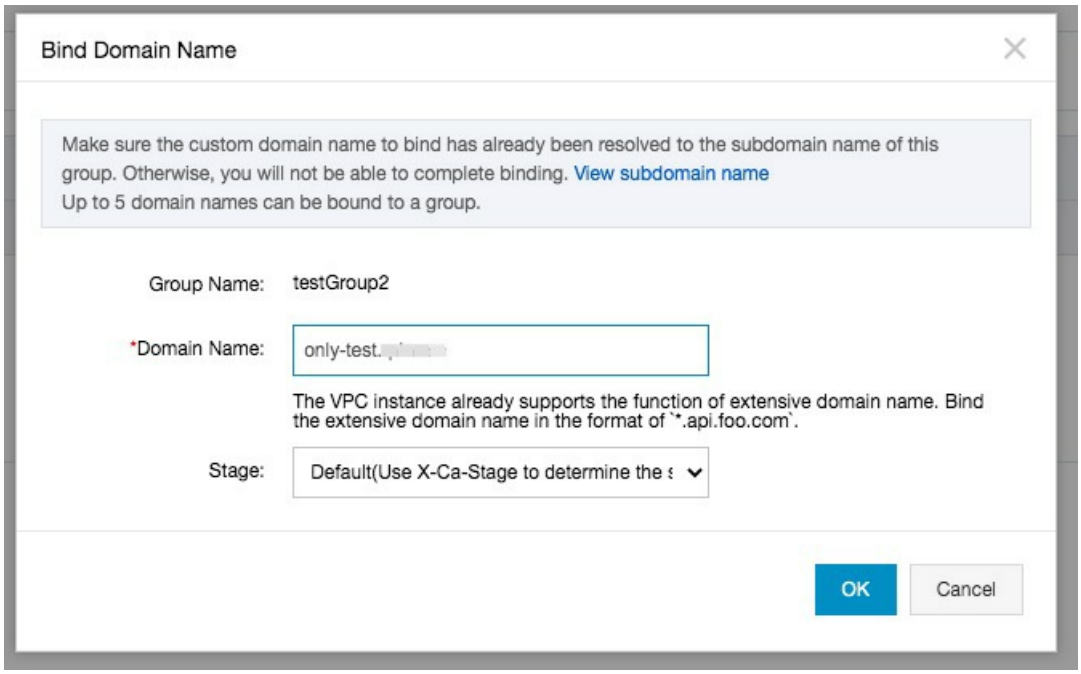
> **Note** You must perform Step 2 and Step 3 on all machines under the apigateway.ApiGatewayLite# SR container.

## Use a custom domain name to call APIs over HTTPS

Perform the following steps to use a custom domain name to call APIs over HTTPS:

1. Bind a custom domain name to a specific API group.
   i. Log on to the API Gateway console. In the left-side navigation pane, choose Publish APIs > API Groups. On the Group List page, click the name of the desired group to go to the Group Details page.
   ii. In the Custom Domain Name section, click **Bind Domain**.


   iii. In the Bind Domain Name dialog box, specify Domain Name and click **OK**.




2. Resolve the domain name.

   To access API Gateway by using this domain name, you must resolve the domain name in Apsara Stack to the virtual IP address of API Gateway.

   Run the ping [Second-level domain] command to obtain the virtual IP address of API Gateway.

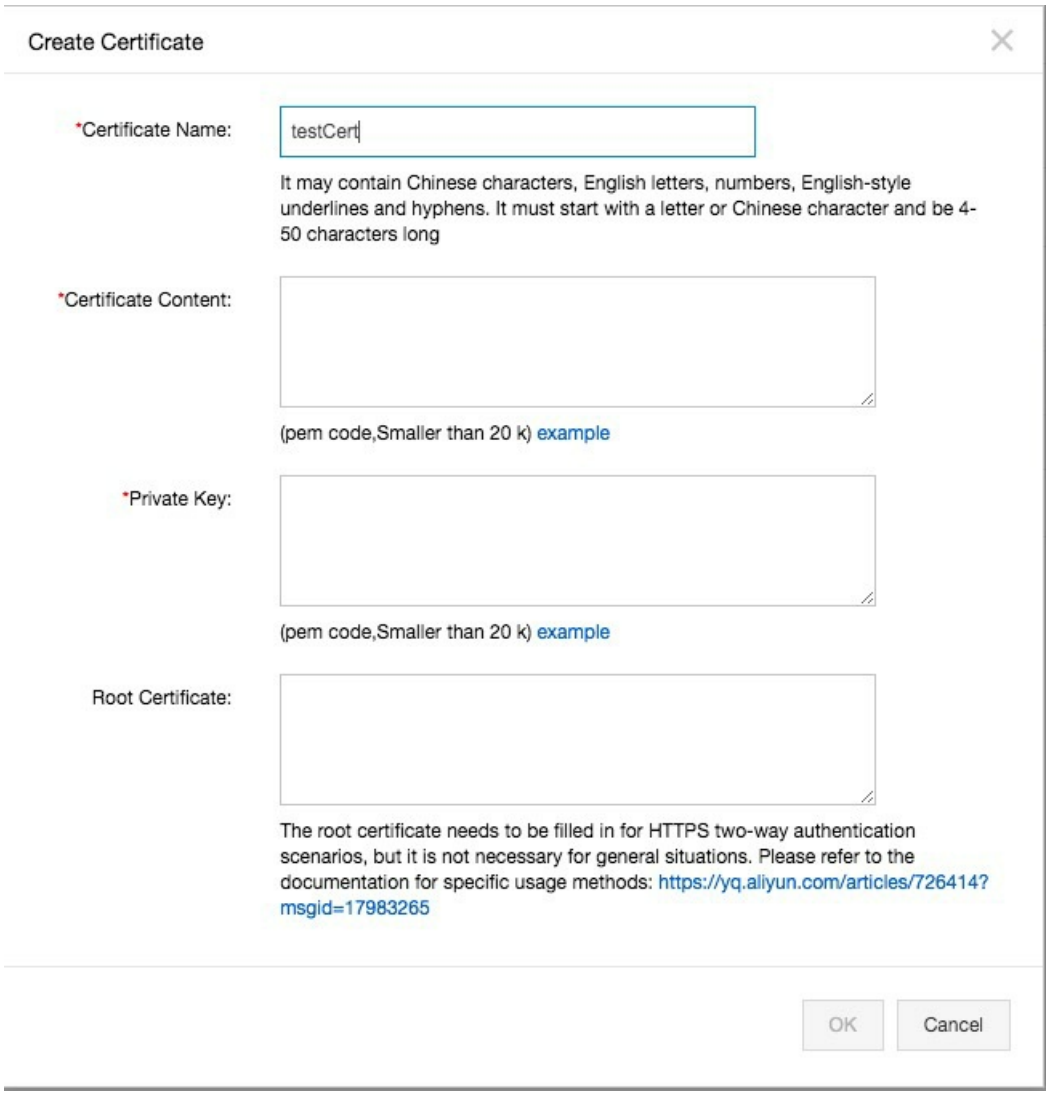
3. Upload a domain name certificate.

   After a domain name is bound to an API group, you can use the domain name to call all the APIs that belong to this API group over HTTP. If you want to call APIs over HTTPS, you must upload an SSL certificate for the domain name. You must prepare the certificate yourself and upload the

certificate to API Gateway.

i. In the **Custom Domain Name** section of the **Group Details** page, find the desired domain name and click **Create Certificate** in the SSL Certificate column.


ii. In the Create Certificate dialog box, specify **Certificate Name**, **Certificate Content**, and **Private Key**. Then, click OK.




**Note** If the certificate is a self-signed certificate, ignore certificate verification when you call APIs.

# 2.1.8. FAQ

## 2.1.8.1. How do I obtain error information?

API Gateway returns a response to the client after it receives a request.

You must check the response headers that start with X-Ca. Take note of the following points:

```
// The unique ID of the request. When API Gateway receives a request, it generates a reques
t ID and returns the request ID to the client in the X-Ca-Request-Id header. We recommend t
hat you record the request ID in both the client and your backend service for troubleshooti
ng and tracking.
X-Ca-Request-Id: 7AD052CB-EE8B-4DFD-BBAF-EFB340E0A5AF
// The error message returned by API Gateway. If a request fails, API Gateway returns the e
rror message to the client in the X-Ca-Error-Message header.
X-Ca-Error-Message: Invalid Parameter Required `field1`
// The error code of a system error in API Gateway. If a request is blocked by API Gateway
due to an error, API Gateway returns the corresponding error code in the X-Ca-Error-Code he
ader. Instances of the classic network type do not have this header.
X-Ca-Error-Code: I400MP
```

The `X-Ca-Error-Code` and `X-Ca-Error-Message` `headers` help you identify the error cause. The `X-Ca-Request-Id` header helps you query request logs in Log Service. You can also provide the request ID included in the X-Ca-Request-Id header for technical support personnel to check log information and resolve issues.

For more information about `X-Ca-Error-Code`, see Error codes.

# 2.1.8.2. Error codes

- If the client receives a response in which the `X-Ca-Error-Code` header is not empty, the header is returned by API Gateway. An error code is six characters in length. For more information, see the following table. `X-Ca-Error-Message` indicates detailed information about an error message.
- If the `X-Ca-Error-Code` header is empty, the HTTP error code is generated by your backend service. API Gateway transparently transmits the error message from your backend service.

| Error code | HTTP status code | Error message | Description |
| --- | --- | --- | --- |
| I400HD | 400 | Invalid Header `${HeaderName}` ${Reason} | The error message returned because the HTTP request header is invalid. |
| I400MH | 400 | Header `${HeaderName}` is Required | The error message returned because the HTTP request header is missing. |

| Error code | HTTP status code | Error message | Description |
| --- | --- | --- | --- |
| I400BD | 400 | Invalid Body: ${Reason} | The error message returned because the HTTP request body is invalid. |
| I400PA | 400 | Invalid Request Path `${Reason}` | The error message returned because the HTTP request path is invalid. |
| I405UM | 405 | Unsupported Method `${Reason}` | The error message returned because the HTTP request method is not supported. |
| I400RU | 400 | Invalid Request Uri `${Reason}` | The error message returned because the HTTP request URL is invalid. |
| I403PT | 403 | Invalid protocol ${Protocol} unsupported | The error message returned because a protocol that is not supported in API configurations is used. Check the API configurations. |
| I413RL | 413 | Request body too Large | The error message returned because the request body is too large. |
| I413UL | 413 | Request URL too Large | The error message returned because the request URL is too long. |
| I400CT | 400 | Invalid Content-Type: `${Reason}` | The error message returned because the Content-Type setting is invalid. |

| Error code | HTTP status code | Error message | Description |
| --- | --- | --- | --- |
| I404DO | 404 | Invalid Domain `${DomainName}` | The error message returned because the domain name is unknown. |
| I410GG | 410 | Group's instance invalid | The error message returned because an invalid instance is requested. The group may not belong to the current instance. |
| I400SG | 400 | Invalid Stage | The error message returned because an unknown environment is requested. |
| I404NF | 404 | API not found ${Reason} | The error message returned because the corresponding API is not found based on the Path and Method settings of the request. |
| X400PM | 400 | Invalid plugin meta ${PluginName} ${Reason} | The error message returned because the metadata of the plugin is invalid. Submit a ticket to contact customer service. |
| X500ED | 500 | Expired api definition | The error message returned because the specified API metadata definition is invalid. Submit a ticket to contact customer service. |

| Error code | HTTP status code | Error message | Description |
| --- | --- | --- | --- |
| X500AM | 500 | Invalid Api Meta, try deploy again or contact us via ticket | The error message returned because the specified API metadata definition is invalid. Submit a ticket to contact customer service. |
| X403DG | 403 | Bad Domain or Group: ${Reason} | The error message returned because the grouped data is invalid. Submit a ticket to contact customer service. |
| B451DO | 451 | Unavailable Domain for Legal Reasons | The error message returned because the domain name does not comply with the requirements of relevant laws and regulations. |
| B451GO | 451 | Unavailable Group for Legal Reasons | The error message returned because the group does not comply with the requirements of relevant laws and regulations. |
| B403OD | 403 | Provider Account Overdue | The error message returned because the API provider has overdue payments. |
| A400AC | 400 | Invalid AppCode ${Reason} | The error message returned because the corresponding AppCode is not found when you perform an authorization in AppCode mode. |

| Error code | HTTP status code | Error message | Description |
| --- | --- | --- | --- |
| A400IK | 400 | Invalid AppKey | The error message returned because the corresponding AppKey setting is not found when you perform an authorization by using a key-secret pair. |
| A403IS | 403 | Invalid Signature, Server StringToSign:`${StringToSign}` | The error message returned because the signature is invalid. For more information, see Request signatures. |
| A403EP | 403 | App authorization expired | The error message returned because the authorization expired. |
| A403PR | 403 | Plugin Authorization Needed | The error message returned because plugin authorization is not performed. |
| A400MA | 400 | Need authorization, `X-Ca-Key` or `Authorization: APPCODE ...` is required | The error message returned because authorization is not performed in AppCode mode or by using a key-secret pair. |
| I400I5 | 400 | Invalid Content-MD5 ${Reason} | The error message returned because Content-MD5 is invalid. |
| I400NC | 400 | X-Ca-Nonce is required | The error message returned because the X-Ca-Nonce header is not provided after you select Force Nonce Check (Anti Replay by X-Ca-Nonce). |

| Error code | HTTP status code | Error message | Description |
| --- | --- | --- | --- |
| S403NU | 403 | Nonce Used | The error message returned because a replay attack is detected. The X-Ca-Nonce header in the request is repeated. |
| S403TE | 403 | X-Ca-Timestamp is expired | The error message returned because the timestamp specified by the X-Ca-Timestamp header expired. |
| I400MP | 400 | Parameter `${ParameterName}` is required | The error message returned because the required parameter is not specified in the API configuration. |
| I400IP | 400 | Invalid parameter `${ParameterName}` ${Reason} | The error message returned because the value of the parameter that is specified in the API configuration is invalid. |
| I400JR | 400 | JWT required | The error message returned because no JWT-related parameters are found. |
| S403JI | 403 | Claim `jti` is required when `preventJtiReplay:true` | The error message returned because no valid jti claim is included in the request when preventJtiReplay is set to true in a JWT authentication plugin. |

| Error code | HTTP status code | Error message | Description |
| --- | --- | --- | --- |
| S403JU | 403 | Claim `jti` in JWT is used | The error message returned because the jti claim that is included in the request is used when preventJtiReplay is set to true in a JWT authentication plugin. |
| I400JD | 400 | JWT Deserialize Failed: `${Token}` | The error message returned because the JWT that is read from the request failed to be parsed. |
| A403JT | 403 | Invalid JWT: ${Reason} | The error message returned because the JWT that is included in the request is invalid. |
| A403JK | 403 | No matching JWK, `${kid}` not found | The error message returned because no JWK matches kid configured in the JWT included in the request. |
| A403JE | 403 | JWT is expired at `${Date}` | The error message returned because the JWT that is read from the request expired. |
| I400JP | 400 | Invalid JWT plugin config: ${JWT} | The error message returned because the JWT authentication plugin is incorrectly configured. |
| A403OL | 403 | OAuth2 Login failed: ${Reason} | |
| A403OU | 403 | OAuth2 Get User Info failed: ${Reason} | |

| Error code | HTTP status code | Error message | Description |
| --- | --- | --- | --- |
| A401OT | 401 | Invalid OAuth2 Access Token | |
| A401OM | 401 | OAuth2 Access Token is required | |
| T429ID | 429 | Throttled by INNER DOMAIN Flow Control, ${Domain} is a test domain, only 1000 requests per day | The error message returned because the number of requests initiated has exceeded the upper limit allowed for a default second-level domain. To increase the quota, use your own domain name. |
| T429IN | 429 | Throttled by INSTANCE Flow Control | The error message returned because throttling is performed for the current instance. |
| T429GR | 429 | Throttled by GROUP Flow Control | The error message returned because throttling is performed for the current group. |
| T429PA | 429 | Throttled by API Flow Control | The error message returned because the default API-level throttling policy defined in the throttling plugin is used. |
| T429PR | 429 | Throttled by PLUGIN Flow Control | The error message returned because the special throttling policy defined in the throttling plugin is used. |
| T429UP | 429 | Throttled by Usage Plan Flow Control | The error message returned because throttling is performed for the usage plan. |

| Error code | HTTP status code | Error message | Description |
| --- | --- | --- | --- |
| T429SR | 429 | Throttled by SERVER Flow Control | |
| T429MR | 429 | Too Many Requests, throttle by `${Description}` | |
| A403IP | 403 | Access denied by IP Control Policy | The error message returned because access is denied by the IP address-based access control plugin. |
| A403IN | 403 | Access from internet is disabled ${Reason} | The error message returned because you are not allowed to call APIs or access API groups over the Internet. |
| A403VN | 403 | Access from invalid VPC is disabled | The error message returned because access over a VPC is denied. |
| A403AC | 403 | Access Control Forbidden by ${RuleName} | The error message returned because access is denied by the access control plugin. |
| A403CO | 403 | Cross origin resource forbidden ${Domain} | The error message returned because access is denied by the CORS plugin. |

| Error code | HTTP status code | Error message | Description |
| --- | --- | --- | --- |
| I404CO | 404 | Cross origin resource not found ${Method} - ${Path} | The error message returned because the API definition is not found based on the Path and Method settings of the request that is pre-checked by the CORS plugin. |
| I404CH | 404 | Content not cached, with `Cache-Control:only-if-cached` | |
| I404NR | 404 | ${Resource} not found | |
| I404SR | 404 | Stage route missing: ${Reason} | |
| B403MO | 403 | Api Market Subscription overdue | The error message returned because the API provider has overdue payments. |
| B403MQ | 403 | Api Market Subscription quota exhausted | The error message returned because the API quota you purchased in Alibaba Cloud Marketplace has been exhausted. |
| B403ME | 403 | Api Market Subscription expired | The error message returned because the API subscription relationship expired. |
| B403MI | 403 | Api Market Subscription invalid | The error message returned because the API marketplace subscription relationship is invalid. |

| Error code | HTTP status code | Error message | Description |
| --- | --- | --- | --- |
| D504RE | 504 | Backend domain `${Domain}` resolve failed | The error message returned because the domain name failed to be resolved at the backend. |
| D504IL | 504 | Backend domain `${Domain}` resolve to illegal address `${Address}` | The error message returned because the domain name resolution results are invalid at the backend. |
| D504CO | 504 | Backend service connect failed `${Reason}` | The error message returned because the backend connection failed. Check the security group configurations, the startup status of the backend server, and firewall configurations. |
| D504CS | 504 | Backend http ssl connect failed `${Reason}` | The error message returned because the backend connection over HTTPS failed. Check whether the backend protocol matches the port. |
| D504TO | 504 | Backend service request timeout | The error message returned because the backend request timed out. |
| X504VE | 504 | Backend service vpc mapped failed | The error message returned because the VPC mapping at the backend is invalid. Submit a ticket to contact customer service. |

| Error code | HTTP status code | Error message | Description |
| --- | --- | --- | --- |
| D503BB | 503 | Backend circuit breaker busy | The error message returned because the API is protected by its circuit breaker. |
| D503CB | 503 | Backend circuit breaker open, ${Reason} | The error message returned because the circuit breaker is open for the API. Check the backend performance of the API. |
| I508LD | 508 | Loop Detected | The error message returned because loopback call is detected. |
| I404DD | 404 | Device id ${DeviceId} not found | The error message returned because the device ID is not found when you call APIs over WebSocket. |
| A403FC | 403 | Function Compute AssumeRole failed ${RequestId}:${Reason} | The error message returned because an authorization error occurs when Function Compute serves as the backend service. |
| D502FC | 502 | Function Compute response invalid: ${Reason} | The error message returned because responses from the backend service of the Function Compute type are invalid. |
| X500ER | 500 | Service Internal Error | The error message returned because an internal server error occurred. Submit a ticket to contact customer service. |

| Error code | HTTP status code | Error message | Description |
| --- | --- | --- | --- |
| X503BZ | 503 | Service Busy | The error message returned because the service is busy in API Gateway. Try again later or submit a ticket to contact customer service. |
| X504TO | 504 | Service timeout | The error message returned because the service processing timed out in API Gateway. |

*Some error codes may change with version updates or the addition of new features.*