

# Distributed Message Service

## Developer Guide

Issue 06  
Date 2019-01-04



**Copyright © Huawei Technologies Co., Ltd. 2020. All rights reserved.**

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Huawei Technologies Co., Ltd.

## **Trademarks and Permissions**



HUAWEI and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

## **Notice**

The purchased products, services and features are stipulated by the contract made between Huawei and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

---

# Contents

---

<b>1 DMS Kafka Developer Guide.....</b>	<b>1</b>
1.1 Overview.....	1
1.2 Java SDK.....	1
1.2.1 Preparing the Environment.....	1
1.2.2 Creating a Project.....	6
1.2.3 Configuring Parameters.....	8
1.2.4 Running the Sample Project.....	10
1.2.5 Compiling the Sample Project Code.....	11
1.2.6 Code of the Sample Project.....	12
1.2.7 Using the Enhanced Java SDK.....	13
1.2.8 FAQs.....	14
1.3 Python SDK.....	14
1.4 Lua SDK.....	16
1.5 C SDK.....	18
1.6 Go SDK.....	21
1.7 Recommended Parameter Settings for Kafka Clients.....	24
<b>2 Standard Queue Developer Guide.....</b>	<b>28</b>
2.1 Overview.....	28
2.2 Preparing the Environment.....	30
2.3 Creating a Project.....	36
2.4 Configuring Parameters.....	37
2.5 Running the Sample Project.....	39
2.6 Compiling the Sample Project Code.....	40
2.7 Code of the Sample Project.....	40
<b>3 FAQs.....</b>	<b>42</b>
3.1 RESTful APIs.....	42
<b>4 Change History.....</b>	<b>43</b>

# 1 DMS Kafka Developer Guide

---

## 1.1 Overview

---

**NOTICE**

This document describes how to connect to Distributed Message Service (DMS) Kafka (non-premium) through clients.

---

### DMS Kafka API

DMS Kafka supports open-source Kafka application programming interfaces (APIs). Third-party applications can implement open-source Kafka service capabilities by directly using a Kafka client to call DMS.

### Usage Restrictions

Generally, DMS Kafka can process thousands of messages per second. If more messages need to be processed per second, submit a service ticket or contact the customer service.

The recommended Kafka client version is 0.10.2.1 or higher.

If the Kafka SDK is used to produce messages, the maximum size of a single message is 10 MB. If the DMS console is used to produce messages, the maximum size of a single message is 512 KB.

## 1.2 Java SDK

### 1.2.1 Preparing the Environment

#### Helpful Links

- [DMS Kafka SDK](#)

- [DMS Kafka Enhanced SDK](#)
- [Sample Project](#)

 **NOTE**

To create a new project, use the downloaded SDK. To write code based on the sample project, use the SDK included in the project.

## Preparing Tools

Eclipse: Download Eclipse 3.6.0 or later from the [Eclipse official website](#).

JDK: Download Java Development Kit 1.8.111 or later from the [Oracle official website](#).

Apache Maven: Download Apache Maven 3.0.3 or later from the [Maven official website](#).

## Obtaining a Topic ID and Consumer Group ID

Before accessing DMS using the SDK, create a Kafka queue and consumer group on the DMS console, and obtain the topic ID and consumer group ID.

- Step 1** Log in to the management console.
- Step 2** Choose **Service List > Application > Distributed Message Service** to launch the DMS console.
- Step 3** In the navigation pane, choose **Queue Manager**.
- Step 4** On the **Queue Manager** page, click **Create Queue**.
- Step 5** Specify queue parameters.

**Table 1-1** Parameter description

Parameter	Description
Name	Name of the queue you want to create. The name must be unique.  When you create a queue, a default queue name is generated, which you can change if required. A queue name consists of 1 to 64 characters. Only letters, digits, underscores (_), and hyphens (-) are allowed.  The queue name cannot be modified after creation of the queue.
Type	Select <b>Kafka queue</b> .

Parameter	Description
Mode	Select either <b>High throughput</b> or <b>High reliability</b> . Default value: <b>High throughput</b> . <b>High throughput:</b> All message replicas are flushed to disk asynchronously. Select this mode when high message delivery performance is required. <b>High reliability:</b> All message replicas are flushed to disk synchronously. Select this mode when high message delivery reliability is required.
Message Retention Period (h)	This parameter is available only for Kafka queues. The number of hours for which messages will be preserved in a Kafka queue. Messages older than that period will be deleted. Deleted messages are not retrievable to consumer groups. Value range: integers from 1 to 72 Default value: <b>72</b>
Description (optional)	The description consists of a maximum of 160 characters and cannot contain angle brackets (< and >).

**Figure 1-1** Creating a Kafka queue

### Create Queue

\* Name  ?

\* Type  Standard  Kafka

Mode  High throughput  High reliability

\* Message Retention Period (h) ?  72  hours

Description

**Step 6** Click **OK**.

**Step 7** Click the name of the queue. On the displayed queue details page, obtain the Kafka topic ID.

**Figure 1-2** Obtaining the Kafka topic ID  
**Basic Information**

Queue Name: queue-1311728893  
 Queue ID: fbb5f52b-a60f-4aba-9bc5-438a4341d5c2  
 Kafka Topic: k-1bed00d7fc8145dbaa782cbdcf7cdec1-fbb5f52b-a60f-4aba-9bc5-438a4341d5c2  
 Queue Type: Kafka

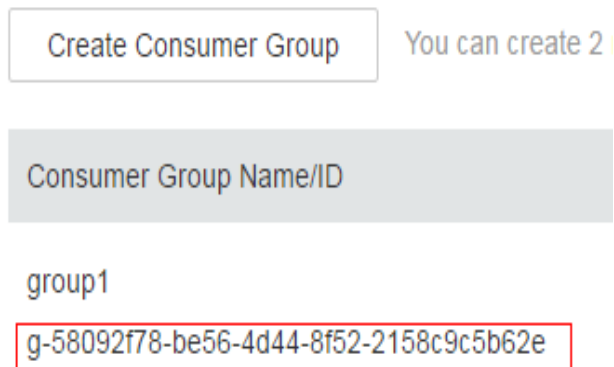
**Step 8** Click **Create Consumer Group**. The **Create Consumer Group** dialog box is displayed.

**Step 9** Enter a consumer group name.

A default queue name is generated, which you can change if required. A consumer group name consists of 1 to 32 characters. Only letters, digits, underscores (\_), and hyphens (-) are allowed. Consumer group names must be unique within the same queue.

**Step 10** Click **OK**. Obtain the ID of the consumer group in the consumer group list.

**Figure 1-3** Obtaining the consumer group ID



----End

## Obtaining a Project ID

When calling APIs, you need to specify **project\_id** in API requests. Obtain a project ID by performing the following procedure.

**Step 1** Sign up and log in to the management console.

**Step 2** Click the username and choose **My Credentials** from the drop-down list.

**Step 3** On the **My Credentials** page, view project IDs in the project list.

**Figure 1-4** Obtaining a project ID

<u>Project List</u>	Access Keys	
Region	Project	Project ID
	cn-east-2	d025c1ff85a24e1d86e15de3bd308a27
	cn-north-1	1bed00d7fc8145dbaa782cbdcf7cdec1
	cn-northeast-1	8c3d5005e3494e139c1f21c4854e79de
	cn-south-1	bd67310717144d19a60a3fae5130c36f

----End

## Obtaining an AK/SK

- Step 1** Sign up and log in to the management console.
- Step 2** Click the username and choose **My Credentials** from the drop-down list.
- Step 3** On the **My Credentials** page, click the **Access Keys** tab.
- Step 4** Click **Create Access Keys**.
- Step 5** Enter the password for login.
- Step 6** Enter the verification code received by email or SMS message.
- Step 7** Click **OK**.

### NOTE

Keep the key secure and do not disclose it to any unauthorized people.

- Step 8** Download the **credentials.csv** file containing your AK and SK to a local computer.

----End

## Obtaining Region and Endpoint Information

Obtain the region and endpoint from [Regions and Endpoints](#).

## Creating an ECS

### NOTE

An Elastic Cloud Server (ECS) must be available to run the sample project.

Log in to the ECS console, create a Linux ECS, and bind an elastic IP address (EIP) to it. Record the EIP, username, and password. If you already created an ECS, skip this step.

 NOTE

EIPs are used to log in to ECSs and upload files.

## Summary of Environment Information

**Table 1-2** Required environment information

Category	Information	Example
ECS	EIP	192.168.141.228
	Username	huaweicloud
	Password	*****
DMS	Queue name	my-kafka-queue
	Queue ID	4df89da6-ede4-4072-93e0-28dc6e866299
	Queue type	Kafka queue
	Topic ID	k- bd67aaead60940d688b872c31bdc653b-4df8 9da6-ede4-4072-93e0-28dc6e866299
	Consumer group name	my-consumer-group
	Consumer group ID	g-7ec0caac-01fb-4f91-a4f2-0a9dd48f8af7
AK/SK	Access key ID	*****
	Secret access key (SK)	*****
Project	Region	CN North-Beijing1
	Project name	cn-north-1
	Project ID	bd67aaead60940d688b872c31bdc653b
Region and endpoint	Name	CN North-Beijing1
	Endpoint	dms-kafka.cn-north-1.myhuaweicloud.com: 37000
DNS	DNS server IP Address	-

### 1.2.2 Creating a Project

This section uses the Maven project **kafkademo** as an example to describe how to create a project.

## Procedure

**Step 1** Download the demo package.

1. Log in to the DMS console.
2. In the navigation pane, choose **Using APIs**.
3. Choose **Kafka APIs**.
4. Click **Download Sample Code** to download DmsKafkaDemo.zip.

**Step 2** Click **Download SDK** to download the DMS Kafka SASL package.

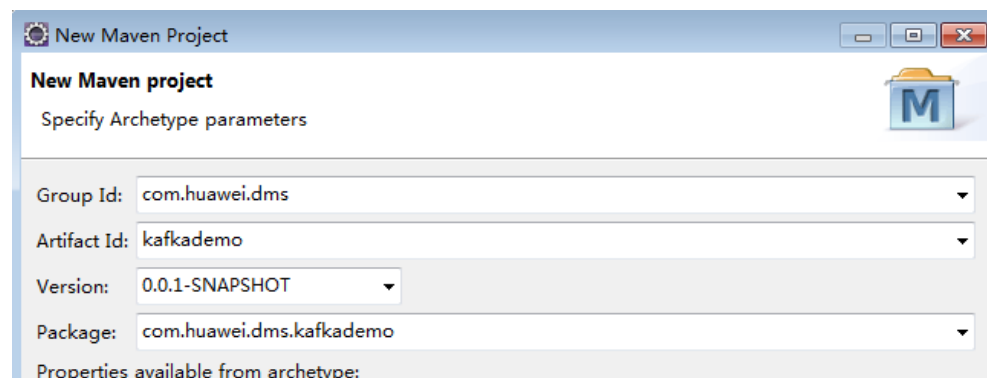
Decompress the following directories from the package:

- **client.truststore.jks**: client certificate
- **dms.kafka.sasl.client-1.0.0.jar**: DMS Kafka SASL package
- **dms\_kafka\_client\_jaas.conf**: client configuration file

You can also decompress the SDK package from \DmsKafkaDemo\dist\libs\dms.kafka.sasl.client-1.0.0.jar.

**Step 3** On Eclipse (the recommended version is 4.6 or later), create a Maven project. The project name **kafkademo** is used as an example.

**Figure 1-5** Creating a Maven project



**Step 4** Click **Finish**.

**Step 5** Import the DMS Kafka SASL package.

1. Right-click the new project **kafkademo**, and create a **libs** folder.
2. Copy **dms.kafka.sasl.client-1.0.0.jar** to **libs**.
3. Add the following information to the **pom.xml** file to import **dms.kafka.sasl.client-1.0.0.jar** into the Maven repository:

```
<dependency>
  <groupId>dms</groupId>
  <artifactId>kafka.sasl.client</artifactId>
  <version>1.0.0</version>
  <scope>system</scope>
  <systemPath>${project.basedir}/libs/dms.kafka.sasl.client-1.0.0.jar</systemPath>
</dependency>
<dependency>
  <groupId>org.apache.kafka</groupId>
  <artifactId>kafka-clients</artifactId>
  <version>0.10.2.1</version>
</dependency>
</dependency>
```

```
<groupId>org.slf4j</groupId>
<artifactId>slf4j-api</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>
```

4. Save the **pom.xml** file.

----End

## 1.2.3 Configuring Parameters

### Procedure

**Step 1** Configure **access\_key**, **secret\_key**, and **project\_id** in the **dms\_kafka\_client\_jaas.conf** file.

The three parameters are used to authenticate DMS Kafka API requests.

```
KafkaClient {
  com.huawei.middleware.kafka.sasl.client.KafkaLoginModule required
  access_key="XXXXXX"
  secret_key="XXXXXX"
  project_id="XXXXXX";
};
```

Replace them with the actual **access\_key**, **secret\_key**, and **project\_id** of your account.

To access the queues authorized by other tenants, set **target\_project\_id** to the project ID of the authorizing tenant.

```
KafkaClient {
  com.huawei.middleware.kafka.sasl.client.KafkaLoginModule required
  access_key="XXXXXX"
  secret_key="XXXXXX"
  project_id="XXXXXX"
  target_project_id="";
};
```

**Step 2** Configure SASL access to start upon process using either of the following methods. In both methods, replace **/path** with the actual path name.

1. Method 1: Configure the following JVM parameter to add the location of SASL configuration file:

```
-Djava.security.auth.login.config=/path/kafka_client_jaas.conf
```

2. Method 2: Add the following information to project code so that SASL access can start before Kafka Producer and Consumer start:

```
System.setProperty("java.security.auth.login.config", "/path/kafka_client_jaas.conf");
```

**Step 3** Add the following information to the **consumer.properties** file:

```
connections.max.idle.ms=30000
```

**Step 4** Configure key parameters in the **consumer.properties/producer.properties** file.

**Table 1-3** Key parameters in the consumer.properties/producer.properties file

Parameter	Description	Setting
bootstrap.servers	IP address or domain name of the DMS server	<p>If the ECS where the client is deployed is in the same region as the DMS queue, access DMS through port 37000.</p> <ul style="list-style-type: none"> <li>• CN North-Beijing1: <b>dms-kafka.cn-north-1.myhuaweicloud.com:37000</b></li> <li>• CN East-Shanghai2: <b>dms-kafka.cn-east-2.myhuaweicloud.com:37000</b></li> <li>• CN South-Guangzhou: <b>dms-kafka.cn-south-1.myhuaweicloud.com:37000</b></li> </ul> <p>If the server where the client is not deployed in HUAWEI CLOUD or is in other regions of HUAWEI CLOUD, access DMS through port 37003.</p> <ul style="list-style-type: none"> <li>• CN North-Beijing1: <b>dms-kafka.cn-north-1.myhuaweicloud.com:37003</b></li> <li>• CN East-Shanghai2: <b>dms-kafka.cn-east-2.myhuaweicloud.com:37003</b></li> <li>• CN South-Guangzhou: <b>dms-kafka.cn-south-1.myhuaweicloud.com:37003</b></li> </ul>
ssl.truststore.location	Path in which the client certificate <b>client.truststore.jks</b> is located	<b><i>/path/client.truststore.jks</i></b> , where <b><i>/path</i></b> must be replaced with the actual path name
ssl.truststore.password	Client certificate password	-
security.protocol	Security protocol	SASL_SSL
sasl.mechanism	Service name	DMS (Note: All letters in the entered service name must be capitalized.)

For details about other Kafka parameters, visit the [official Kafka website](#).

**Step 5** Enable Kafka debug logging by modifying the **log4j.properties** file.

```
log4j.rootLogger=DEBUG, stdout
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=[%d] %p %m (%c:%L)%n
log4j.logger.org.apache.kafka.clients=DEBUG
```

```
log4j.logger.kafka=INFO, stdout
log4j.additivity.kafka=false
log4j.logger.org.apache.kafka=DEBUG, stdout
log4j.additivity.org.apache.kafka=false
```

**Step 6** Write code. For details about APIs, visit the [official Kafka website](#).

----End

## 1.2.4 Running the Sample Project

The following describes how to access DMS Kafka queues to produce and consume messages in Java.

### Procedure

**Step 1** Log in to the ECS.

#### NOTE

You can run the sample project on an ECS with an IP address in the 192 network segment.

**Step 2** Install JDK or Java runtime environment (JRE). Add the following settings of environment variables **JAVA\_HOME** and **PATH** to the `~/.bash_profile`:

```
export JAVA_HOME=/opt/java/jdk1.8.0_151
export PATH=$JAVA_HOME/bin:$PATH
```

Run the **source .bash\_profile** command for the modification to take effect.

#### NOTE

Use Oracle JDK instead of ECS's default JDK (for example, OpenJDK), because ECS's default JDK may not be suitable for the sample project. To obtain Oracle JDK, download Java Development Kit 1.8.111 or a later version from <https://www.oracle.com/technetwork/java/javase/downloads/index.html>.

**Step 3** Run the following command to download the code package of the sample project **DmsKafkaDemo.zip**.

```
$ wget https://dms-demo.obs.cn-north-1.myhuaweicloud.com/DmsKafkaDemo.zip
```

**Step 4** Run the following command to decompress **DmsKafkaDemo.zip**.

```
$ unzip DmsKafkaDemo.zip
```

**Step 5** Run the following command to navigate to the **DmsKafkaDemo/dist** directory, which contains pre-compiled binary files and executable scripts.

```
$ cd DmsKafkaDemo/dist
```

**Step 6** Edit the **config/dms\_kafka\_client\_jaas.conf** file and configure **access\_key**, **secret\_key**, and **project\_id**.

```
$ vim config/dms_kafka_client_jaas.conf
```

The values in bold are examples. Replace them with actual values.

```
KafkaClient {
  com.huawei.middleware.kafka.sasl.client.KafkaLoginModule required
  access_key="*****"
  secret_key="*****"
  project_id="bd67aaead60940d688b872c31bdc653b"
  target_project_id="bd67aaead60940d688b872c31bdc6539";
};
```

To access the queues authorized by other tenants, set **target\_project\_id** to the project ID of the authorizing tenant.

**Step 7** Edit the **config/producer.properties** file and configure **topic** and **bootstrap.servers**.

```
$ vim config/producer.properties
```

The values in bold are examples. Replace them with actual values.

```
topic=k-bd67aaead60940d688b872c31bdc653b-4df89da6-ed4-4072-93e0-28dc6e866299
bootstrap.servers=dms-kafka.cn-north-1.myhuaweicloud.com:37000
ssl.truststore.password=*****
acks=all
retries=1
batch.size=16384
buffer.memory=33554432
key.serializer=org.apache.kafka.common.serialization.StringSerializer
value.serializer=org.apache.kafka.common.serialization.StringSerializer
security.protocol=SASL_SSL
sasl.mechanism=DMS
```

 **NOTE**

The parameter **topic** can be set to a queue name or a Kafka topic name. For more information, see [Table 1-2](#).

**Step 8** Edit the **config/consumer.properties** file and configure **topic**, **bootstrap.servers**, and **group.id**.

```
$ vim config/consumer.properties
```

The values in bold are examples. Replace them with actual values.

```
topic=k-bd67aaead60940d688b872c31bdc653b-4df89da6-ed4-4072-93e0-28dc6e866299
bootstrap.servers=dms-kafka.cn-north-1.myhuaweicloud.com:37000
group.id=g-7ec0caac-01fb-4f91-a4f2-0a9dd48f8af7
ssl.truststore.password=*****
security.protocol=SASL_SSL
sasl.mechanism=DMS
key.deserializer=org.apache.kafka.common.serialization.StringDeserializer
value.deserializer=org.apache.kafka.common.serialization.StringDeserializer
auto.offset.reset=earliest
enable.auto.commit=false
```

 **NOTE**

The parameter **topic** can be set to a queue name or a Kafka topic name. For more information, see [Table 1-2](#).

**Step 9** Run the sample project to produce messages:

```
$ bash produce.sh
```

After the command is run, 10 messages are automatically sent to the Kafka queue.

**Step 10** Run the sample project to consume messages:

```
$ bash consume.sh
```

----End

## 1.2.5 Compiling the Sample Project Code

**Step 1** Download and decompress [the sample project code DmsKafkaDemo](#).

**Step 2** Import the sample project code.

1. On Eclipse, choose **File > Import**.
2. In the **Select an import source** area, select **Existing Projects Into Workspace**.
3. Select the directory to which the sample project code **DmsKafkaDemo** is decompressed.

**Step 3** Choose **Project > Build Project** to build the project.

**Step 4** Export the new JAR file.

1. Right-click the sample project **DmsKafkaDemo** and choose **Export** from the shortcut menu.
2. Choose **Java > JAR file**. Enter the path and name of the JAR file to be generated.

**Step 5** Replace the **dms.kafka.demo.jar** file in the **DmsKafkaDemo/dist/libs** directory with the new JAR file. Run the newly built project by following the procedure in [Running the Sample Project](#).

----End

## 1.2.6 Code of the Sample Project

### Producer

DMS Kafka APIs are compatible with native open-source Kafka clients. Compared with native Kafka service code, the sample project code additionally contains a client certificate and simple authentication and security layer (SASL) configuration, which are used for identity authentication and secure communication. To realize smooth migration of producer applications, you only need to import the client certificate and SASL configuration before creating the Kafka Producer without modifying any other Kafka service code.

Code pertaining to client certificate and SASL:

```
Properties producerConfig = Config.getProducerConfig();
producerConfig.put("ssl.truststore.location", Config.getTrustStorePath());
System.setProperty("java.security.auth.login.config", Config.getSaslConfig());
```

The code for creating a producer and sending messages does not need to be modified.

### Consumer

DMS Kafka APIs are compatible with native open-source Kafka clients. Compared with native Kafka service code, the sample project code additionally contains a client certificate and SASL configuration, which are used for identity authentication and secure communication. To realize smooth migration of consumer applications, you only need to import the client certificate and SASL configuration before creating the Kafka Consumer without modifying any other Kafka service code.

Code pertaining to client certificate and SASL:

```
Properties consumerConfig = Config.getConsumerConfig();
consumerConfig.put("ssl.truststore.location", Config.getTrustStorePath());
System.setProperty("java.security.auth.login.config", Config.getSaslConfig());
```

The code for creating a consumer and consuming messages does not need to be modified.

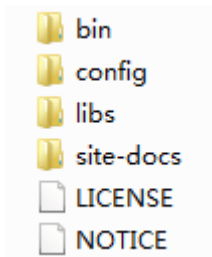
## 1.2.7 Using the Enhanced Java SDK

The enhanced Kafka Java SDK is optimized based on the open-source Kafka 0.10.2.1 client, greatly improving system performance.

### Procedure

**Step 1** **Download** the open-source Kafka 0.10.2.1 client package.

**Step 2** Decompress the following from the open-source client package:



**Step 3** **Download** the enhanced Kafka 0.10.2.1 SDK package.

**Step 4** Decompress the following from the SDK package:

```
[root@dms-APIServer01 dms_kafka_0.10.2.1-client]# ll
total 7932
-rw-r-----. 1 root root 107729 Jan  9 2018 client.truststore.jks
-rw-r-----. 1 root root   161 Oct 11 2017 dms_kafka_client_jaas.conf
-rw-r-----. 1 root root  10192 Mar  6 20:31 dms_kafka_sasl_client-1.0.0.jar
-rw-r-----. 1 root root 947668 May 19 18:51 kafka-clients-0.10.2.1.jar
-rw-r-----. 1 root root 271289 Apr  4 10:10 netty-buffer-4.1.19.Final.jar
-rw-r-----. 1 root root 313264 Apr  4 10:10 netty-codec-4.1.19.Final.jar
-rw-r-----. 1 root root   54840 Apr  4 10:10 netty-codec-dns-4.1.19.Final.jar
-rw-r-----. 1 root root 403636 Apr  4 10:10 netty-codec-http2-4.1.19.Final.jar
-rw-r-----. 1 root root 563491 Apr  4 10:10 netty-codec-http-4.1.19.Final.jar
-rw-r-----. 1 root root 119358 Apr  4 10:10 netty-codec-socks-4.1.19.Final.jar
-rw-r-----. 1 root root 567599 Apr  4 10:10 netty-common-4.1.19.Final.jar
-rw-r-----. 1 root root 375088 Apr  4 10:10 netty-handler-4.1.19.Final.jar
-rw-r-----. 1 root root  20957 Apr  4 10:10 netty-handler-proxy-4.1.19.Final.jar
```

**Step 5** Copy all JAR packages in the **dms\_kafka\_0.10.2.1-client/** directory to the **libs** folder in the directory where the open-source Kafka 0.10.2.1 client package is decompressed, and overwrite the **kafka-clients-0.10.2.1.jar** package with the same name.

**Step 6** Perform the same configuration as other open-source clients.

----End

## 1.2.8 FAQs

### What If the Consumer Group Failed to Be Found?

Symptom: An error message similar to the following is displayed. **k-xx** represents a created Kafka queue.

```
[2019-04-29 11:30:02,173] WARN Error while fetching metadata with correlation id 2 :  
{k-8*****8cd=UNKNOWN}
```

Possible causes and solutions:

- If open JDK is used, use Oracle JDK instead.
- Check if the consumer group is consistent with the topic. By default, a consumer group name starts with the letter g.
- Check if **project\_id** and **target\_project\_id** in the **dms\_kafka\_client\_jaas.conf** file are inconsistent. The two parameters must have consistent values.

## 1.3 Python SDK

This chapter describes how to access a DMS Kafka queue by using a Linux Python client. To obtain related configuration information, see [Preparing the Environment](#).

DMS is compatible with native Kafka APIs. For details on how to call native Kafka APIs, see the [Kafka Documentation](#).

### Preparing the Client Environment

**Step 1** Select an SDK package of the required version based on the Python encoding format.

1. [Download the SDK package](#) and decompress the following from the package:
  - kafka-dms-python-code2.tar.gz
  - kafka-dms-python-code4.tar.gz
2. Run the following command in any directory to add the **test.py** file:

**vi test.py**

Add the following content to the file and save the file.

```
import sys  
if sys.maxunicode > 65535:  
    print 'Your python is suitable for kafka-dms-python-code4.tar.gz'  
else:  
    print 'Your python is suitable for kafka-dms-python-code2.tar.gz'
```

3. Run the following command to select a Python SDK package:  
**python test.py**
4. Select the required SDK package as prompted.

**Step 2** Decompress the SDK package.

The following assumes that the SDK package is decompressed in the **{root\_dir}** package.

**Step 3** Run the following commands to configure environment variables:

```
vi ~/.bash_profile
```

Add the following content to the configuration file, and then save and exit.

```
export LD_LIBRARY_PATH={root_dir}/kafka-dms-python/confluent_kafka/lib:{root_dir}/kafka-dms-python/  
confluent_kafka/thirdPart
```

Run the following command to make the modification take effect:

```
source ~/.bash_profile
```

**Step 4** Configure the following parameters in the `{root_dir}/kafka-dms-python/conf.ini` file:

```
sasl.project.id=your project  
sasl.access.key=your ak  
sasl.security.key=your sk
```

For details on how to obtain the values of these parameters, see [Preparing the Environment](#). After the modification is complete, save the file.

----End

## Running Example

**Step 1** Run the following command to produce messages:

```
cd {root_dir}/kafka-dms-python/  
python producer.py <bootstrap-brokers> <topic> <msg1> <msg2> <msg3> ..
```

Set **bootstrap-brokers** to a Kafka endpoint and **topic** to a Kafka Topic ID. For details on how to obtain an endpoint and a Kafka topic ID, see [Preparing the Environment](#).

Examples:

```
python producer.py dms-kafka.cn-north-1.myhuaweicloud.com:37003 k-  
bd67aaead60940d688b872c31bdc653b-4df89da6-ede4-4072-93e0-28dc6e866299 123 456 789
```

**Step 2** Run the following command to start consuming message from the earliest message in the queue.

```
python consumer.py <bootstrap-brokers> <group> <topic>
```

Set **group** to the ID of a consumer group for the Kafka queue. For details on how to obtain a consumer group ID, see [Preparing the Environment](#).

Examples:

```
python consumer.py dms-kafka.cn-north-1.myhuaweicloud.com:37003 g-7ec0caac-01fb-4f91-  
a4f2-0a9dd48f8af7 k-bd67aaead60940d688b872c31bdc653b-4df89da6-ede4-4072-93e0-28dc6e866299
```

----End

## Code Example

Producer parameters

```
# Read lines from stdin, produce each line to Kafka  
for msg in msgs:  
    try:
```

```
# Produce line (without newline)
p.produce(topic, msg.rstrip(), callback=delivery_callback)

except BufferError as e:
    sys.stderr.write('%% Local producer queue is full (%d messages awaiting delivery): try again\n' %
                    len(p))

# Serve delivery callback queue.
# NOTE: Since produce() is an asynchronous API this poll() call
# will most likely not serve the delivery callback for the
# last produce()d message.
p.poll(0)
```

### Consumer parameters

```
# Read messages from Kafka, print to stdout
try:
    while True:
        msg = c.poll(timeout=1.0)
        if msg is None:
            continue
        if msg.error():
            # Error or event
            if msg.error().code() == KafkaError._PARTITION_EOF:
                # End of partition event
                sys.stderr.write('%% %s [%d] reached end at offset %d\n' %
                                (msg.topic(), msg.partition(), msg.offset()))
            elif msg.error():
                # Error
                raise KafkaException(msg.error())
        else:
            # Proper message
            sys.stderr.write('%% %s [%d] at offset %d with key %s:\n' %
                            (msg.topic(), msg.partition(), msg.offset(),
                             str(msg.key())))
            print(msg.value())

except KeyboardInterrupt:
    sys.stderr.write('%% Aborted by user\n')
```

## 1.4 Lua SDK

This chapter describes how to access a DMS Kafka queue by using a Linux Lua client. To obtain related configuration information, see [Preparing the Environment](#).

DMS is compatible with native Kafka APIs. For details on how to call native Kafka APIs, see the [Kafka Documentation](#).

### Preparing the Client Environment

#### Step 1 [Download the SDK package](#).

Decompress the **kafka-dms-lua.tar.gz** file from the SDK package.

#### Step 2 Install the LuaJIT tool.

Run the following command to check whether the LuaJIT tool has been installed:

```
luajit -v
```

Install the LuaJIT tool if it has not been installed.

- Ubuntu OS:

- Run the **apt-cache search luajit** command to check whether LuaJIT exists in the software repository. If not, configure the correct software repository address.
- Run the **apt-get install luajit** command to install LuaJIT.
- SUSE OS:
  - Run the **zypper search luajit** command to check whether LuaJIT exists in the software repository. If not, configure the correct software repository address.
  - Run the **zypper install luajit** command to install LuaJIT.
- CentOS or Red Hat OS:
  - **yum search luajit**
  - **yum install luajit**

**Step 3** Decompress the **kafka-dms-lua.tar.gz** package.

The following assumes that the package is decompressed in the **{DMSPATH}** directory.

**Step 4** Run the following command to copy the Kafka dependency packages to the **lib** directory:

```
cp {DMSPATH}/kafka-dms-lua/librdkafka/lib/* /usr/lib/ -R
```

**Step 5** Modify the parameters in the **example\_dms.lua** file.

```
cd {DMSPATH}/kafka-dmslua/luadms
```

```
vi example_dms.lua
```

Modify the following parameters:

```
local BROKERS_ADDRESS = { "broker" }
local TOPIC_NAME = "your kafka topic"

config["sasl.project.id"]="your projectId"
config["sasl.access.key"]="your ak"
config["sasl.security.key"]="your sk"
```

**Table 1-4** Parameter description

Parameter	Description	How to Obtain
BROKERS_ADDRESS	Kafka endpoint	For details, see <a href="#">Preparing the Environment</a> .
TOPIC_NAME	Kafka topic ID	
sasl.project.id	Project ID	
sasl.access.key	Tenant AK	
security.key	Tenant SK	

----End

## Running Example

Run the following command to produce messages:

```
luajit example.lua
```

After this command is run, 10 messages are automatically sent to the Kafka queue.

## Code Example

Producer parameters

```
for i = 0,10 do
  producer:produce(topic, KAFKA_PARTITION_UA, "this is test message"..tostring(i))
end

while producer:outq_len() ~= 0 do
  producer:poll(10)
end
```

## 1.5 C SDK

This section describes how to access a DMS Kafka queue by using a Linux C client. To obtain related configuration information, see [Preparing the Environment](#).

DMS is compatible with native Kafka APIs. For details on how to call native Kafka APIs, see the [Kafka Documentation](#).

### Preparing the Client Environment

#### Step 1 [Download the SDK package](#).

Decompress the **kafka-dms-c.tar.gz** file from the SDK package.

#### Step 2 Install the GNU Compiler Collection (GCC) tool.

Run the following command to check whether the GCC tool has been installed:

```
gcc -v
```

Install the GCC tool if it has not been installed.

- Ubuntu OS:
  - Run the **apt-cache search gcc** command to check whether GCC exists in the software repository. If not, configure the correct software repository address.
  - Run the **apt-get install gcc** command to install GCC.
- SUSE OS:
  - Run the **zypper search gcc** command to check whether GCC exists in the software repository. If not, configure the correct software repository address.
  - Run the **zypper install gcc** command to install GCC.
- CentOS or Red Hat OS:

- **yum search gcc**
- **yum install gcc**

**Step 3** Decompress the **kafka-dms-c.tar.gz** package.

The following assumes that the package is decompressed in the **{DMSPATH}** directory.

**Step 4** Run the following command to copy Kafka dependency packages to the local include directory:

```
cp {DMSPATH}/kafka-dms-c/librdkafka/include/librdkafka /usr/include/ -R
```

**Step 5** Run the following command to copy the Kafka dependency packages to the **lib** directory:

```
cp {DMSPATH}/kafka-dms-c/librdkafka/lib/* /usr/lib/ -R
```

**Step 6** Run the following commands to compile the sample code:

```
cd {DMSPATH}/kafka-dms-c/example
```

```
gcc my_consumer.c -o my_consumer -lrdkafka -lz -lpthread -lrt
```

```
gcc my_producer.c -o my_producer -lrdkafka -lz -lpthread -lrt
```

If the message "cannot find -lz" is displayed, run the **apt-get install zlib1g-dev** command.

----End

## Running Example

**Step 1** Run the following command to produce messages:

```
./my_producer <broker> <topic> <project_id> <access_key> <secret_key>
```

**Table 1-5** Parameter description

Parameter	Description	How to Obtain
broker	Kafka endpoint	For details, see <a href="#">Preparing the Environment</a> .
topic	Kafka topic ID	
project_id	Project ID	
access_key	Tenant AK	
secret_key	Tenant SK	

After running the preceding command, you can send a message to the Kafka instance by writing it and pressing **Enter**. Each line of content is sent as a message.

To stop creating messages, press **Ctrl+C** to exit.

**Step 2** Run the following command to consume messages:

```
./my_consumer <broker> <topic> <group> <project_id> <access_key>
<secret_key>
```

**Table 1-6** Parameter description

Parameter	Description	How to Obtain
broker	Kafka endpoint	For details, see <a href="#">Preparing the Environment</a> .
topic	Kafka topic ID	
group	Kafka consumer group ID	
project_id	Project ID	
access_key	Tenant AK	
secret_key	Tenant SK	

After the preceding command is run, messages will be consumed from the earliest message in the queue. To stop consuming messages, press **Ctrl+C** to exit.

----End

## Code Example

Producer parameters

```
if (rd_kafka_produce(
    /* Topic object */
    rkt,
    /* Selects built-in partitions.*/
    RD_KAFKA_PARTITION_UA,
    /* Generates a copy of the payload.*/
    RD_KAFKA_MSG_F_COPY,
    /* Message body and length*/
    buf, len,
    /* Optional key and its length*/
    NULL, 0,
    NULL) == -1){
    fprintf(stderr,
        "%s Failed to produce to topic %s: %s\n",
        rd_kafka_topic_name(rkt),
        rd_kafka_err2str(rd_kafka_last_error()));

    if (rd_kafka_last_error() == RD_KAFKA_RESP_ERR_QUEUE_FULL){
        /* If the internal queue is full, wait until the message transmission is complete and then try
again.
The internal queue indicates messages to be sent and messages that have been sent or failed to
be sent.
The internal queue is limited by the queue.buffering.max.messages configuration.*/
        rd_kafka_poll(rk, 1000);
        goto retry;
    }
}
else{
    fprintf(stderr, "%s Enqueued message (%zd bytes) for topic %s\n",
        len, rd_kafka_topic_name(rkt));
```

### Consumer parameters

```
if(!initKafka(brokers, group, topic, project_id, access_key, secret_key)){
    fprintf(stderr, "kafka server initialize error\n");
}else{
    while(run){
        rd_kafka_message_t *rkmessage;
        /* Polls for consumers' messages or events, which will block messages for timeout_ms at most.
        - The application should periodically call consumer_poll () even if there are no expected
        messages. Calling this function is important especially when rebalance_cb has been registered.
        This is because this function needs to be correctly called and processed to synchronize the
        internal consumer status. */
        rkmessage = rd_kafka_consumer_poll(rk, 1000);
        if(rkmessage){
            msg_consume(rkmessage, NULL);
            /* Releases the resources of rkmessage and gives the ownership back to rd_kafka*/.
            rd_kafka_message_destroy(rkmessage);
        }
    }
}
```

## 1.6 Go SDK

This section describes how to access a DMS Kafka queue by using a Linux Go client. To obtain related configuration information, see [Preparing the Environment](#).

DMS is compatible with native Kafka APIs. For details on how to call native Kafka APIs, see the [Kafka Documentation](#).

### Preparing the Client Environment

#### Step 1 [Download the SDK package](#).

Decompress the **kafka-dms-go.tar.gz** file from the SDK package.

#### Step 2 Install the Go and pkg-config tools.

1. Run the following command to check whether the Go tool has been installed:  
**go version**
2. If not, download the tool at the following address and then install the tool.  
<https://golang.org/doc/install?download=go1.10.3.linux-amd64.tar.gz>
3. Run the following command to check whether the pkg-config tool has been installed:  
**pkg-config --version**
4. If not, download the tool at the following address and then install the tool.  
<http://www.linuxfromscratch.org/blfs/view/7.4/general/pkgconfig.html>

#### Step 3 Decompress the **kafka-dms-go.tar.gz** package.

The following assumes that the package is decompressed in the **{DMSPATH}** directory.

#### Step 4 Run the following command to copy the **kafka** folder to the **src** directory of the Go tool:

```
cp {DMSPATH}/kafka-dms-go/confluent-kafka-go-0.11.0/kafka
{GO_HOME}/src -R
```

**Step 5** Run the following command to copy the **pkgconfig** file to the configuration directory of the pkg-config tool:

```
cp {DMSPATH}/kafka-dms-go/librdkafka/lib/pkgconfig/* /usr/share/pkgconfig
```

**Step 6** Run the following command to copy the Kafka dependency packages to the **include** directory:

```
cp {DMSPATH}/kafka-dms-go/librdkafka/include/librdkafka /usr/include/ -R
```

**Step 7** Run the following command to copy the Kafka dependency packages to the **lib** directory:

```
cp {DMSPATH}/kafka-dms-go/librdkafka/lib/* /usr/lib -R
```

**Step 8** Run the following command to install Kafka:

```
go install kafka
```

**Step 9** Modify the parameters in the **producer\_example.go** file.

```
cd {DMSPATH}/kafka-dms-go/example
```

```
vi producer_example.go
```

Modify the following parameters:

```
broker := "broker address"
topic := "kafka topic"

config["sasl.project.id"] = "your projectId"
config["sasl.access.key"] = "your ak"
config["sasl.security.key"] = "your sk"
```

**Table 1-7** Parameter description

Parameter	Description	How to Obtain
broker	Kafka endpoint	For details, see <a href="#">Preparing the Environment</a> .
topic	Kafka topic ID	
sasl.project.id	Project ID	
sasl.access.key	Tenant AK	
security.key	Tenant SK	

**Step 10** Modify the parameters in the **consumer\_example.go** file.

```
vi consumer_example.go
```

Modify the following parameters:

```
broker := "broker address"
group := "group id"
var topics = []string{"kafka topic"}

config["sasl.project.id"] = "your projectId"
config["sasl.access.key"] = "your ak"
config["sasl.security.key"] = "your sk"
```

**Table 1-8** Parameter description

Parameter	Description	How to Obtain
broker	Kafka endpoint	For details, see <a href="#">Preparing the Environment</a> .
group	Kafka consumer group ID	
topics	Kafka topic ID	
sasl.project.id	Project ID	
sasl.access.key	Tenant AK	
security.key	Tenant SK	

----End

## Running Example

**Step 1** Run the following command to produce messages:

```
go run producer_example.go
```

After the command is run, one message will be automatically sent to the Kafka queue. To send multiple messages, repeat running this command.

**Step 2** Run the following command to consume messages:

```
go run consumer_example.go
```

----End

## Code Example

Producer parameters

```
// Optional delivery channel, if not specified the Producer object's
// .Events channel is used.
deliveryChan := make(chan kafka.Event)

value := "Hello Go!"
err = p.Produce(&kafka.Message{TopicPartition: kafka.TopicPartition{Topic: &topic, Partition:
kafka.PartitionAny}, Value: []byte(value)}, deliveryChan)

e := <-deliveryChan
m := e.(*kafka.Message)
```

Consumer parameters

```
for run == true {
    select {
    case sig := <-sigchan:
        fmt.Printf("Caught signal %v: terminating\n", sig)
        run = false
    default:
        ev := c.Poll(100)
        if ev == nil {
            continue
        }
        switch e := ev.(type) {
```

```

case *kafka.Message:
    fmt.Printf("%% Message on %s:\n%s\n",
        e.TopicPartition, string(e.Value))
case kafka.PartitionEOF:
    fmt.Printf("%% Reached %v\n", e)
case kafka.Error:
    fmt.Fprintf(os.Stderr, "%% Error: %v\n", e)
    run = false
default:
    fmt.Printf("Ignored %v\n", e)
    }
    }
}

```

## 1.7 Recommended Parameter Settings for Kafka Clients

Table 1-9 Producer parameters

Parameter	Default Value	Recommended Value	Description
acks	1	<p><b>all</b> (if high reliability mode is selected)</p> <p><b>1</b> (if high throughput mode is selected)</p>	<p>Indicates the number of acknowledgments the producer requires the server to return before considering a request complete. This controls the durability of records that are sent. The value of this parameter can be any of the following:</p> <p><b>0:</b> The producer will not wait for any acknowledgment from the server at all. The record will be immediately added to the socket buffer and considered sent. No guarantee can be made that the server has received the record, and the retries configuration will not take effect (as the client generally does not know of any failures). The offset given back for each record will always be set to -1.</p> <p><b>1:</b> The leader will write the record to its local log but will respond without waiting until receiving full acknowledgement from all followers. If the leader fails immediately after acknowledging the record but before the followers have replicated it, the record will be lost.</p> <p><b>all:</b> The leader will wait for the full set of replicas to acknowledge the record. This is the strongest available guarantee that the record will not be lost as long as there is at least one replica.</p>

Parameter	Default Value	Recommended Value	Description
retries	0	0	Setting this parameter to a value greater than zero will cause the client to resend any record that failed to be sent due to a potentially transient error. Note that this retry is no different than if the client resent the record upon receiving the error. Allowing retries will potentially change the ordering of records because if two batches are sent to the same partition, and the first fails and is retried but the second succeeds, then the records in the second batch may appear first.
request.timeout.ms	30000	120000	Indicates the maximum amount of time the client will wait for the response of a request. If the response is not received before the timeout elapses, the client will throw a Timeout exception.
block.on.buffer.full	TRUE	Default value	When buffer memory is exhausted, the producer must stop receiving new message records or throw an exception. By default, this parameter is set to <b>TRUE</b> . However, in some cases, non-blocking usage is desired and it is better to throw an exception immediately. Setting this parameter to <b>FALSE</b> will cause the producer to instead throw "BufferExhaustedException" when buffer memory is exhausted.
batch.size	16384	262144	<p>The producer will attempt to batch records together into fewer requests whenever multiple records are being sent to the same partition. This helps improve performance of both the client and the server. This parameter controls the default batch size in bytes.</p> <p>No attempt will be made to batch records larger than this size.</p> <p>Requests sent to brokers will contain multiple batches, one for each partition with data available to be sent.</p> <p>A smaller batch size will make batching less common and may reduce throughput (a batch size of zero will disable batching entirely). A larger batch size may use more memory as a buffer of the specified batch size will always be allocated in anticipation of additional records.</p>

Parameter	Default Value	Recommended Value	Description
buffer.memory	33554432	536870912	<p>The total bytes of memory the producer can use to buffer records waiting to be sent to the server. If records are sent faster than they can be delivered to the broker, the producer will stop sending records or throw a "block.on.buffer.full" exception.</p> <p>This setting should correspond roughly to the total memory the producer will use, but is not a rigid bound since not all memory the producer uses is used for buffering. Some additional memory will be used for compression (if compression is enabled) as well as for maintaining in-flight requests.</p>

**Table 1-10** Consumer parameters

Parameter	Default Value	Recommended Value	Description
auto.commit.enable	TRUE	FALSE	<p>If this parameter is set to <b>TRUE</b>, the offset of messages already fetched by the consumer will be periodically committed to ZooKeeper. This committed offset will be used when the process fails as the position from which the new consumer will begin.</p> <p>Constraints: If this parameter is set to <b>FALSE</b>, to avoid message loss, an offset must be committed to ZooKeeper after the messages are successfully consumed.</p>
auto.offset.reset	latest	earliest	<p>Indicates what to do when there is no initial offset in ZooKeeper or if the current offset has been deleted. Options:</p> <p><b>earliest</b>: The offset is automatically reset to the smallest offset.</p> <p><b>latest</b>: The offset is automatically reset to the largest offset.</p> <p><b>none</b>: The system throws an exception to the consumer if no offset is available.</p> <p><b>anything else</b>: The system throws an exception to the consumer.</p>

Parameter	Default Value	Recommended Value	Description
connections.max.idle.ms	60000	30000	Indicates the timeout interval for an idle connection. The server closes the idle connection after this period of time ends. Setting this parameter to 30000 can reduce the server response failures when the network condition is poor.

# 2 Standard Queue Developer Guide

---

## 2.1 Overview

### DMS TCP SDK

Distributed Message Service (DMS) TCP Software Development Kit (SDK) provides TCP-based application programming interfaces (APIs). Third-party applications can call DMS TCP SDK APIs to produce, consume, and acknowledge messages. [Table 2-1](#) describes the APIs provided by DMS for third-party applications to call. For details about the APIs, see the corresponding API reference document.

**Table 2-1** DMS TCP SDK APIs

API	Description	Type
Message production API	Sends messages to DMS queues.	Synchronous, asynchronous, and one-way
Message consumption API	Retrieves messages from DMS queues.	Synchronous and asynchronous
Message acknowledgment API	Acknowledges that messages are successfully retrieved from DMS queues.	Synchronous and asynchronous

 **NOTE**

- After an asynchronous or one-way API is called, the process calling this API exits immediately. This may cause incomplete message delivery.
- If a one-way API is called, the client does not receive responses (including error messages) from the server.
- DMS does not support "exactly once" message delivery. If a message is displayed indicating that the operation has failed, there is a low possibility that the operation has actually succeeded. DMS provides "at-least-once" message delivery. To avoid any adverse effects from processing the same message multiple times, ensure your application (consumer) processes messages idempotently.

## Features

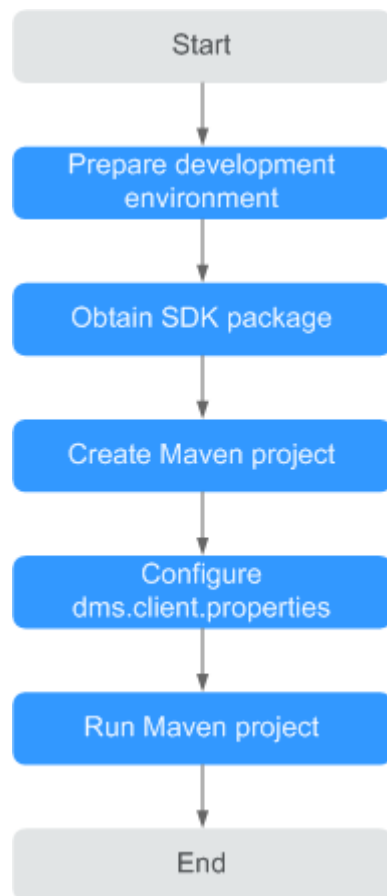
- **Message redelivery**  
The DMS TCP SDK supports message redelivery. If a message fails to be retrieved, it can be flagged as redelivery and dumped to the message queue. Consumers can retrieve the message later. Messages can be retrieved at least 30 seconds after being redelivered. When you acknowledge message retrieval, you can choose to flag messages as success, failure, or redelivery.
- **Message broadcasting**  
All consumers in a consumer group can consume the same message. You can configure whether to automatically acknowledge message consumption.  
Consumers can choose any of the following message consumption modes:
  - **Multicast:** After a message is successfully consumed, other consumers in the same consumer group can no longer consume the message.
  - **Broadcast from the earliest message:** All consumers in a consumer group can consume the first message and subsequent messages in the queue.
  - **Broadcast from the previous consumption position:** If a fixed **Consumer id** is set, all consumers in a consumer group can consume all the messages that are never consumed in the queue from the specified retrieval position.
  - **Broadcast from the latest message:** All consumers in a consumer group can consume new incoming messages in the queue.
- Generally, the DMS TCP SDK can process thousands of messages per second. If more messages need to be processed per second, submit a work order or contact the customer service.

## Development Process

The Java language is used as an example to describe how to perform secondary development based on DMS TCP SDK.

The development process is as follows:

**Figure 2-1** Development process



## 2.2 Preparing the Environment

### Helpful Links

- [DMS TCP API Documents](#)
- [DMS TCP SDK](#)
- [Sample Project](#)

 **NOTE**

To create a new project, use the downloaded SDK. To write code based on the sample project, use the SDK included in the project.

The following describes how to access DMS using a Java client in Linux.

### Preparing Tools

Eclipse: Download Eclipse 3.6.0 or later from the [Eclipse official website](#).

JDK: Download Java Development Kit 1.8.111 or later from the [Oracle official website](#).

Apache Maven: Download Apache Maven 3.0.3 or later from the [Maven official website](#).

## Obtaining a Queue ID and Consumer Group ID

### NOTE

Before accessing DMS using the SDK, create a queue and consume group on the DMS console or by calling APIs.

[Code of the Sample Project](#) provides examples of the processes for producing and consuming messages.

The following describes the processes of creating a queue and a consumer group and obtaining the queue ID and consumer group ID on the DMS console. To learn how to perform these operations by calling APIs, see the [DMS API Reference](#).

Procedure:

- Step 1** Log in to the management console.
- Step 2** Choose **Service List > Application > Distributed Message Service** to launch the DMS console.
- Step 3** In the navigation pane, choose **Queue Manager**.
- Step 4** On the **Queue Manager** page, click **Create Queue**.  
  
By default, a user can create a maximum of 30 queues. The number of queues that can be created is shown above the **Delete** button. If you want to create more queues, contact customer service to increase your quota.
- Step 5** Specify queue parameters.

**Table 2-2** Parameter description

Parameter	Description
Queue Name	<p>Name of the queue you want to create. The name must be unique.</p> <p>When creating a queue, a default queue name is generated, which you can change if required. A queue name consists of 1 to 64 characters. Only letters, digits, underscores (_), and hyphens (-) are allowed.</p> <p>The queue name cannot be modified after the queue is created.</p>
Queue Type	<ul style="list-style-type: none"> <li>• <b>Standard queue:</b> Partition-level FIFO mode and global FIFO mode are supported.</li> <li>• <b>Kafka:</b> Open-source Kafka clients can be used for creating and consuming messages. High throughput mode and high reliability mode are supported.</li> </ul>

Parameter	Description
Queue Mode	<p>When <b>Queue Type</b> is <b>Standard</b>, the queue mode can be:</p> <ul style="list-style-type: none"> <li>• <b>Partition-level FIFO:</b> Messages may be consumed out of sequence, but the concurrency is higher.</li> <li>• <b>Global FIFO:</b> Messages are consumed in the order they were sent.</li> </ul> <p>When <b>Queue Type</b> is <b>Kafka</b>, the queue mode can be:</p> <ul style="list-style-type: none"> <li>• <b>High throughput:</b> All message replicas are flushed to a disk asynchronously. Select this mode when high message delivery performance is required.</li> <li>• <b>High reliability:</b> All message replicas are flushed to disk synchronously. Select this mode when high message delivery reliability is required.</li> </ul>
Dead Letter Queue	<p>This parameter is available only for standard queues.</p> <p>This parameter indicates whether to enable dead letter messages. Dead letter messages are messages that cannot be correctly consumed.</p> <p>If a message fails to be consumed for a preset number of times, it will be sent to the dead letter queue and retained for 72 hours. You can then consume the message from the dead letter queue.</p> <p>Dead letter messages can be consumed only by the consumer group that generated these dead letter messages.</p> <p>Dead letter messages from global FIFO queues are sent to the dead letter queue in the FIFO order.</p> <p>By default, <b>Dead Letter Queue</b> is disabled.</p>
Maximum Retrievals	<p>This parameter is available only when dead letter messages are enabled.</p> <p>Maximum number of times a message can be consumed before it is sent to the dead letter queue.</p> <p>Value range: 1–100.</p> <p>Default value: <b>3</b></p>
Description (optional)	<p>The description consists of a maximum of 160 characters and cannot contain angle brackets (&lt; and &gt;).</p>

**Figure 2-2** Creating a queue

The screenshot shows a 'Create Queue' dialog box. The title bar reads 'Create Queue' with a close button 'x'. The dialog contains the following fields and options:

- Name:** A text input field containing 'queue-167538317'.
- Type:** A button group with 'Standard' selected and 'Kafka' as an alternative.
- Mode:** A button group with 'Partition-level FIFO' selected and 'Global FIFO' as an alternative.
- Dead Letter Queue:** A radio button that is currently unselected.
- Description:** A large text area for entering a description, with a character count '0/160' at the bottom right.

At the bottom of the dialog, there are two buttons: 'OK' (highlighted in red) and 'Cancel'.

**Step 6** Click **OK**.

**Step 7** Click the name of the queue. On the displayed queue details page, obtain the queue ID, as shown in [Figure 2-3](#).

**Figure 2-3** Obtaining the queue ID

Basic Information

Queue Name: queue-1381799375

Queue ID: e18aea7c-3a95-4429-b46b-3e5022f5fd7f

Kafka Topic: k-1bed00d7fc8145dbaa782cbdcf7cdec1-e18aea7c-3a95-4429-b46b-3e5022f5fd7f

Queue Type: Kafka

Description:

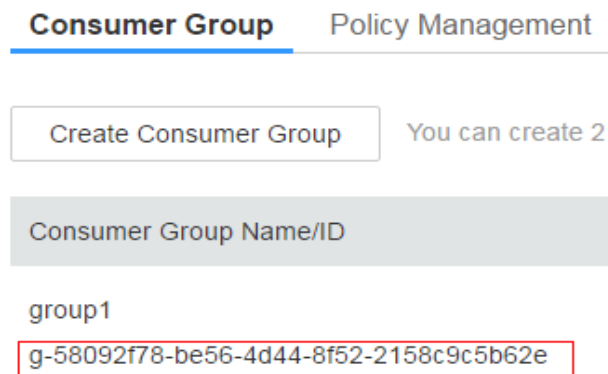
**Step 8** Click **Create Consumer Group**.

**Step 9** Enter a consumer group name.

A default consumer group name is generated, which you can change if required. A consumer group name consists of 1 to 32 characters. Only letters, digits, underscores (\_), and hyphens (-) are allowed. Consumer group names must be unique within the same queue.

**Step 10** Click **OK**. Obtain the ID of the consumer group in the consumer group list, as shown in [Figure 2-4](#).

**Figure 2-4** Obtaining the consumer group ID



----End

## Obtaining a Project ID

When calling APIs, you need to specify **project\_id** in API requests. Obtain a project ID by performing the following procedure:

- Step 1** Log in to the management console.
- Step 2** Click the username and choose **My Credential** from the drop-down list.
- Step 3** On the **My Credentials** page, view project IDs in the project list.

**Figure 2-5** Obtaining the project ID

The screenshot shows a web interface with two tabs: "Project List" (active) and "Access Keys". Below the tabs is a table listing projects. The table has columns for Region, Project, and Project ID.

Region	Project	Project ID
cn-east-2	cn-east-2	d025c1ff85a24e1d86e15de3bd308a27
cn-north-1	cn-north-1	1bed00d7fc8145dbaa782cbdcf7cdec1
cn-northeast-1	cn-northeast-1	8c3d5005e3494e139c1f21c4854e79de
cn-south-1	cn-south-1	bd67310717144d19a60a3fae5130c36f

----End

## Obtaining an AK/SK

- Step 1** Log in to the management console.
- Step 2** Click the username and choose **My Credential** from the drop-down list.
- Step 3** On the **My Credentials** page, click the **Access Keys** tab.

**Step 4** Click **Add Access Key**.

**Step 5** Enter the password for login.

**Step 6** Enter the verification code sent to your mailbox or mobile phone.

**Step 7** Click **OK**.

 **NOTE**

Keep the access key ID (AK)/secret access key (SK) file secure.

**Step 8** Download the **credentials.csv** file containing your AK and SK to a local computer.

----End

## Obtaining Region and Endpoint Information

Region	Endpoint	Description
CN North-Beijing1	dms-tcp.cn-north-1.myhuaweicloud.com:60010	Use the endpoint to access DMS.
CN East-Shanghai2	dms-tcp.cn-east-2.myhuaweicloud.com:60010	
CN South-Guangzhou	dms-tcp.cn-south-1.myhuaweicloud.com:60010	

## Creating an ECS

**Step 1** Before using DMS TCP SDK to access DMS over public networks, you need to create an Elastic Cloud Server (ECS).

 **NOTE**

Elastic IP addresses (EIPs) are used to log in to ECSs and upload files.

----End

## Collecting Environment Information

**Table 2-3** Required environment information

Category	Information	Example
ECS	EIP	192.168.141.228
	Username	huaweicloud
	Password	*****
DMS	Queue name	my-tcp-queue
	Queue ID	5964181e-b67e-4d35-9281-a58c352abda6

Category	Information	Example
	Queue type	Kafka
	Queue mode	Global FIFO
	Consumer group name	my-tcp-group
	Consumer group ID	g-c861aa48-a082-48ff-bc9d-ff36fe7aed38
AK/SK	AK	*****
	SK	*****
Project	Region	CN North-Beijing1
	Project name	cn-north-1
	Project ID	bd67aaead60940d688b872c31bdc663b
Region and endpoint	Region	CN North-Beijing1
	Endpoint	dms-tcp.cn-north-1.myhuaweicloud.com:60010

## 2.3 Creating a Project

This section uses the Maven project **tcpdemo** as an example to describe how to create a project.

### Procedure

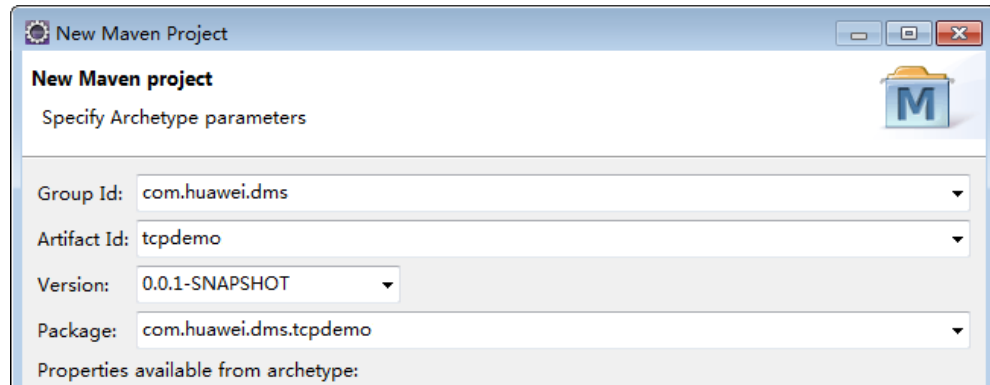
**Step 1** Download the SDK package **dmssdk.zip**.

#### Download address

The **dmssdk.zip** contains the following files:

- **dms.sdk-1.0.0.jar**: DMS TCP SDK package
- **dms.protocol-1.0.0.jar**: DMS TCP SDK package
- **dms.client.properties**: configuration file
- **README.txt**: release description file
- **third\_lib.zip**: third-party JAR package on which DMS TCP SDK depends
- **dms.sdk.api.chm**: API reference

**Step 2** On Eclipse (the recommended version is 4.6 or later), create a Maven project. The project name **tcpdemo** is used as an example.



**Step 3** Click **Finish**.

**Step 4** Import the DMS TCP SDK JAR packages.

1. Right-click the new project **tcpdemo**, and create a **libs** folder.
2. Copy **dms.sdk-1.0.0.jar** and **dms.protocol-1.0.0.jar** to the **libs** directory.
3. Add the following information to the **pom.xml** file to import the JAR packages into the Maven repository:

```
<dependency>
  <groupId>com.huawei.middleware.dms</groupId>
  <artifactId>dms.protocol</artifactId>
  <version>1.0.0</version>
  <type>jar</type>
  <scope>system</scope>
  <systemPath>${basedir}/libs/dms.protocol-1.0.0.jar</systemPath>
</dependency>
<dependency>
  <groupId>com.huawei.middleware.dms</groupId>
  <artifactId>dms.sdk</artifactId>
  <version>1.0.0</version>
  <type>jar</type>
  <scope>system</scope>
  <systemPath>${basedir}/libs/dms.sdk-1.0.0.jar</systemPath>
</dependency>
```

4. Save the **pom.xml** file.
5. If the third-party dependency package is not automatically imported, import the package in **third\_lib.zip** into the Maven repository by referring to [Compiling the Sample Project Code](#).

----End

## 2.4 Configuring Parameters

**Step 1** Configure parameters described in [Table 2-4](#).

**Table 2-4** Parameters in the dms.client.properties file

Parameter	Description
dms.server.ip	DMS server address (IP address or domain name) <ul style="list-style-type: none"><li>• CN North-Beijing1: <b>dms-tcp.cn-north-1.myhuaweicloud.com</b></li><li>• CN East-Shanghai2: <b>dms-tcp.cn-east-2.myhuaweicloud.com</b></li><li>• CN South-Guangzhou: <b>dms-tcp.cn-south-1.myhuaweicloud.com</b></li></ul>
dms.server.port	DMS TCP port Set this parameter to <b>60010</b> .
dms.client.blocked.worker.num	Number of threads that executes blocking callbacks By default, this parameter is set to <b>-1</b> . If this parameter is set to a value smaller than 0, then: Number of threads = 2 x Number of CPU cores
dms.client.conn.timeout	Connection timeout time Unit: ms Default value: <b>6000</b>
dms.client.reconn.attempt.times	Number of reconnection attempts Unit: ms Default value: <b>3</b>
dms.client.reconn.attempt.interval	Interval at which reconnection attempts are initiated Unit: ms Default value: <b>3000</b>
dms.client.request.timeout	Request timeout time Unit: ms Default value: <b>60000</b>
dms.client.start.timeout	Client startup timeout time Duration: ms Default value: <b>60000</b>
dms.client.project.id	Project ID For details on how to obtain a project ID, see <a href="#">Obtaining a Project ID</a> .
dms.client.access.key	Access key ID For details on how to obtain an access key ID, see <a href="#">Obtaining an AK/SK</a> .

Parameter	Description
dms.client.secret.key	Secret access key For details on how to obtain a secret access key, see <a href="#">Obtaining an AK/SK</a> .

**Step 2** Write code. For details about APIs, see the `dms.sdk.api.chm` file described in [Step 1](#).

----End

## 2.5 Running the Sample Project

**Step 1** Log in to the ECS.

**Step 2** Install JDK or Java runtime environment (JRE). Add the following settings of environment variables `JAVA_HOME` and `PATH` to the `~/.bash_profile`:

```
export JAVA_HOME=/opt/java/jdk1.8.0_151
export PATH=$JAVA_HOME/bin:$PATH
```

Run the `source .bash_profile` command for the modification to take effect.

### NOTE

Use Oracle JDK instead of ECS's default JDK (for example, OpenJDK), because ECS's default JDK may not be suitable for the sample project. To obtain Oracle JDK, download Java Development Kit 1.8.111 or a later version from the [Oracle official website](#).

**Step 3** Run the following command to download [the code package of the sample project DmsTcpDemo](#):

```
$ wget https://dms-demo.obs.cn-north-1.myhuaweicloud.com/DmsTcpDemoClient.zip
```

**Step 4** Run the following command to decompress the `DmsTcpDemoClient.zip` package.

```
$ unzip DmsTcpDemoClient.zip
```

**Step 5** Run the following command to navigate to the `DmsTcpDemoClient/dist` directory which contains pre-compiled binary files and executable scripts.

```
$ cd DmsTcpDemoClient/dist
```

**Step 6** Edit the `config/dms.client.properties` file and configure `access_key`, `secret_key` and `project_id`.

```
$ vim config/dms.client.properties
```

The red values are examples. Replace them with actual values.

```
dms.server.ip=dms-tcp.cn-north-1.myhuaweicloud.com
dms.server.port=60010
dms.client.project.id=bd67aaead60940d688b872c31bdc663b
dms.client.access.key=*****
dms.client.secret.key=*****
```

**Step 7** Execute the demo script by running the following command containing the queue ID and consumer group ID to test message production and consumption.

```
$ bash dms_tcp_demo.sh 5964181e-b67e-4d35-9281-a58c352abda6 g-c861aa48-a082-48ff-bc9d-ff36fe7aed38
```

After the command is run, 10 messages are sent to the queue, and then consumed. The consumption results will be returned.

**Step 8** Log in to the [DMS console](#) to view the number of messages in a specified queue and verify the execution result of the demo script.

----End

## 2.6 Compiling the Sample Project Code

**Step 1** Download and decompress the [DMS TCP sample project code](#).

**Step 2** Import the sample project code.

1. On Eclipse, choose **File > Import**.
2. Choose **Maven > Existing Maven Projects**, click **Next**.
3. Select the directory to which the sample project code **DmsTcpDemoClient.zip** is decompressed.

**Step 3** Right-click the **dms.tcp.demo** project. Choose **Run As > Maven build...** from the shortcut menu. Enter **clean package** to build the project.

**Step 4** Save the build output in the **target** directory. The default file name is **dms.tcp.demo-1.0.0-SNAPSHOT.jar**.

**Step 5** Replace the **dms.tcp.demo-1.0.0.jar** file in the **DmsTcpDemoClient/dist/libs** directory on the ECS with **dms.tcp.demo-1.0.0-SNAPSHOT.jar**. Then run the newly built project following the procedure described in [Running the Sample Project](#).

----End

## 2.7 Code of the Sample Project

The DMS TCP sample project contains two examples:

DmsProducerDemo: demonstrates the message production process.

DmsConsumerDemo: demonstrates the message consumption and acknowledgment processes.

### Producing Messages

**Step 1** Create and start DmsProducer.

```
DmsProducer producer = new DmsProducerImpl();
producer.start();
```

**Step 2** Construct the content of the message to be sent.

```
List<DmsMessage> messages = new ArrayList<>();
final int messageNum = 10;
for (int i = 0; i < messageNum; i++)
{
    DmsMessage record = new DmsMessage();
    record.setBody(("Hello DMS syn produce: " + i).getBytes());
    messages.add(record);
}
```

**Step 3** Send the message.

```
try
{
    List<DmsProduceResult> result = producer.produce(queueId, messages);
    result.forEach(r ->
    {
        System.out.println(r.getState());
    });
}
catch (Throwable t)
{
    t.printStackTrace();
}
```

**Step 4** Stop message production.

```
producer.stop();
```

----End

## Consuming Messages

**Step 1** Create and start a consumer instance.

```
DmsConsumer consumer = new DmsConsumerImpl();
consumer.start();
```

**Step 2** Consume messages.

```
List<DmsConsumeResult> records = consumer.consume(queueId, groupId);
```

**Step 3** Construct a request for acknowledging the message consumption.

```
DmsCommitRequest commitRequest = new DmsCommitRequest();
List<DmsCommitItem> messages = new ArrayList<>();

for (DmsConsumeResult record : records)
{
    System.out.println(record.getHandler());
    DmsCommitItem commitItem = new DmsCommitItem();
    //The corresponding message handler needs to be filled in during acknowledgment.
    commitItem.setHandler(record.getHandler());
    //Set the message handling status to true or false.
    commitItem.setStatus(true);
    messages.add(commitItem);
}
//Multiple messages can be acknowledged at a time.
commitRequest.setMessage(messages);
```

**Step 4** Acknowledge the message consumption.

```
DmsCommitResult commitResult = consumer.commit(queueId, groupId, commitRequest);
```

**Step 5** Stop message consumption.

```
consumer.stop();
```

----End

# 3 FAQs

---

## 3.1 RESTful APIs

For details about RESTful APIs, see [Calling APIs](#).

Use RESTful APIs to create Kafka queues, create consumer groups, and produce and consume messages on the DMS console.

RESTful APIs can also be used to create queues and consumer groups in Kafka SDK.

# 4 Change History

---

Date	Change History
2019-01-04	Removed the ActiveMQ queue type.
2018-07-19	This issue is the fifth official release. <ul style="list-style-type: none"><li>• Added descriptions of Python, Lua, C, and Go SDKs in the Kafka developer guide.</li></ul>
2018-06-27	This issue is the fourth official release. Revised description in the previous issue.
2018-04-20	This issue is the fourth official release. Revised description in the previous issue.
2018-03-15	This issue is the second official release. Added the ActiveMQ developer guide.
2017-12-05	This issue is the first official release.