Distributed Message Service for Kafka

Developer Guide

Issue 01

Date 2020-12-02

HUAWEI

HUAWEI TECHNOLOGIES CO., LTD.

Trademarks and Permissions


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

1 Overview

Kafka premium instances are compatible with Apache Kafka and can be accessed using **open-source Kafka clients**. To access an instance in SASL mode, you will also need **certificates**.

This document describes how to collect instance connection information, such as the instance connection address, certificate required for SASL connection, and information required for public access. It also provides examples of accessing an instance in Java and Python.

The examples only demonstrate how to invoke Kafka APIs for producing and consuming messages. For more information about the APIs provided by Kafka, visit the **Kafka official website**.

Client Network Environment

A client can access a Kafka instance in any of the following three modes:

1. Within a Virtual Private Network (VPC)

   If the client runs an Elastic Cloud Server (ECS) and is in the same region and VPC as the Kafka instance, the client can access the instance using an IP address within a subnet in the VPC.

2. Using a VPC peering connection

   If the client runs an ECS and is in the same region but not the same VPC as the Kafka instance, the client can access the instance using an IP address within a subnet in the VPC after a VPC peering connection has been established.

3. Over public networks

   If the client is not in the same network environment or region as the Kafka instance, the client can access the instance using a public network IP address.

   For public access, modify the inbound rules of the security group configured for the Kafka instance, allowing access over port 9095.

NOTE

The three modes differ only in the connection address for the client to access the instance. This document takes intra-VPC access as an example to describe how to set up the development environment.

If the connection times out or fails, check the network connectivity. You can use telnet to test the connection address and port of the instance.

2 Collecting Connection Information

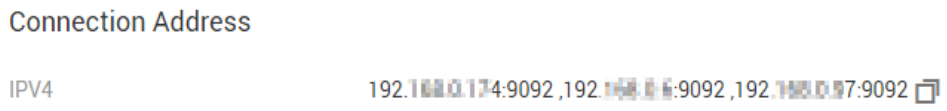
Required Kafka Instance Information

1. Instance connection address and port

   After an instance is created, you can obtain the IP address from the **Basic Information** tab page of the instance. You can configure all IP addresses on the client. For example, as shown in the following figure, a 100 MB/s instance has three addresses. Instances of other specifications may have a different number of brokers, so the number of addresses vary.

   For public access, you can use the connection addresses listed in the **Public Access** section.

   Figure 2-1 Viewing the connection addresses and ports of brokers of a Kafka instance

2. Topic name

   Obtain the topic name on the **Topic Management** tab page.

3. SASL information

   If SASL_SSL is enabled for the instance, the SASL_SSL username, password, and **SSL certificate** are required. The username and password are set during the creation of the instance.

   Obtain the username on the **Basic Information** tab page. If the password is lost, you can reset the password.

3 Java

3.1 Configuring Kafka Clients in Java

This section describes how to add Kafka clients in Maven, and use the clients to access Kafka instances and produce and consume messages. To check how the demo project runs in IDEA, see **Setting Up the Java Development Environment**.

The Kafka instance connection addresses, topic name, and user information used in the following examples are obtained in **Collecting Connection Information**.

Adding Kafka Clients in Maven

```
//Kafka premium instances are based on Kafka 2.3.0. Use the same version of clients.
    <dependency>
       <groupId>org.apache.kafka</groupId>
       <artifactId>kafka-clients</artifactId>
       <version>2.3.0</version>
    </dependency>
```

Preparing Kafka Configuration Files

The following describes example producer and consumer configuration files. If SASL is not enabled for the Kafka instance, comment out lines regarding SASL. If SASL has been enabled, set SASL configurations for encrypted access.

- Producer configuration file (the **dms.sdk.producer.properties** file in the demo project)

  The information in bold is specific to different Kafka instances and must be modified. Other parameters can also be added.

  ```
  #The topic name is in the code for producing or consuming messages.
  ########################
  #Information about Kafka brokers. ip:port are the connection addresses and ports used by the instance. The values can be obtained by referring to the "Collecting Connection Information" section.
  Example: bootstrap.servers=100.xxx.xxx.87:909x,100.xxx.xxx.69:909x,100.xxx.xxx.155:909x
  bootstrap.servers=ip1:port1,ip2:port2,ip3:port3
  #Producer acknowledgement
  acks=all
  #Method of turning the key into bytes
  key.serializer=org.apache.kafka.common.serialization.StringSerializer
  #Method of turning the value into bytes
  value.serializer=org.apache.kafka.common.serialization.StringSerializer
  ```



```
#Memory available to the producer for buffering
buffer.memory=33554432
#Number of retries
retries=0
########################
#Comment out the following parameters if SASL access is not enabled.
########################
#Configure the JAAS username and password. username and password are set when you enable Kafka SASL_SSL during instance creation.
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required \
   username="username" \
   password="password";
#SASL mechanism
sasl.mechanism=PLAIN
#Encryption protocol. Currently, only SASL_SSL is supported.
security.protocol=SASL_SSL
#Location of ssl.truststore
ssl.truststore.location=E:\\temp\\client.truststore.jks
#Password of ssl.truststore
ssl.truststore.password=dms@kafka
#Whether to verify the certificate domain name. This parameter must be left blank, which indicates disabling domain name verification.
ssl.endpoint.identification.algorithm=
```

- Consumer configuration file (the **dms.sdk.consumer.properties** file in the demo project)

  The information in bold is specific to different Kafka instances and must be modified. Other parameters can also be added.

```
#The topic name is in the code for producing or consuming messages.
########################
#Information about Kafka brokers. ip:port are the connection addresses and ports used by the instance. The values can be obtained by referring to the "Collecting Connection Information" section. Example: bootstrap.servers=100.xxx.xxx.87:909x,100.xxx.xxx.69:909x,100.xxx.xxx.155:909x
bootstrap.servers=ip1:port1,ip2:port2,ip3:port3
#Unique string to identify the group of consumer processes to which the consumer belongs. Configuring the same group.id for different processes indicates that the processes belong to the same consumer group.
group.id=1
#Method of turning the key into bytes
key.deserializer=org.apache.kafka.common.serialization.StringDeserializer
#Method of turning the value into bytes
value.deserializer=org.apache.kafka.common.serialization.StringDeserializer
#Offset reset policy
auto.offset.reset=earliest
########################
#Comment out the following parameters if SASL access is not enabled.
########################
#Configure the JAAS username and password. username and password are set when you enable Kafka SASL_SSL during instance creation.
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required \
   username="username" \
   password="password";
#SASL mechanism
sasl.mechanism=PLAIN
#Encryption protocol. Currently, only SASL_SSL is supported.
security.protocol=SASL_SSL
#Location of ssl.truststore
ssl.truststore.location=E:\\temp\\client.truststore.jks
#Password of ssl.truststore
ssl.truststore.password=dms@kafka
# Disable certificate domain name verification.
ssl.endpoint.identification.algorithm=
```

Producing Messages

- Test code

```
package com.dms.producer;
```



```
import org.apache.kafka.clients.producer.Callback;
import org.apache.kafka.clients.producer.RecordMetadata;
import org.junit.Test;

public class DmsProducerTest {
    @Test
    public void testProducer() throws Exception {
        DmsProducer<String, String> producer = new DmsProducer<String, String>();
        int partiton = 0;
        try {
            for (int i = 0; i < 10; i++) {
                String key = null;
                String data = "The msg is " + i;
                //Enter the name of the topic you created. There are multiple APIs for producing messages.
For details, see the Kafka official website or the following code.
                producer.produce("topic-0", partiton, key, data, new Callback() {
                    public void onCompletion(RecordMetadata metadata,
                        Exception exception) {
                        if (exception != null) {
                            exception.printStackTrace();
                            return;
                        }
                        System.out.println("produce msg completed");
                    }
                });
                System.out.println("produce msg:" + data);
            }
        } catch (Exception e) {
            // TODO: Exception handling
            e.printStackTrace();
        } finally {
            producer.close();
        }
    }
}
```

- Message production code

```
package com.dms.producer;

import java.io.BufferedInputStream;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.URL;
import java.util.ArrayList;
import java.util.Enumeration;
import java.util.List;
import java.util.Properties;

import org.apache.kafka.clients.producer.Callback;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.Producer;
import org.apache.kafka.clients.producer.ProducerRecord;

public class DmsProducer<K, V> {
     //Add the producer configurations that have been specified earlier.
    public static final String CONFIG_PRODUCER_FILE_NAME = "dms.sdk.producer.properties";

    private Producer<K, V> producer;

    DmsProducer(String path)
    {
        Properties props = new Properties();
        try {
            InputStream in = new BufferedInputStream(new FileInputStream(path));
            props.load(in);
        }catch (IOException e)
        {
            e.printStackTrace();
            return;
```



```
    }
    producer = new KafkaProducer<K,V>(props);
}
DmsProducer()
{
    Properties props = new Properties();
    try {
        props = loadFromClasspath(CONFIG_PRODUCER_FILE_NAME);
    }catch (IOException e)
    {
        e.printStackTrace();
        return;
    }
    producer = new KafkaProducer<K,V>(props);
}

/**
 * Producing messages
 *
 * @param topic        Topic
 * @param partition    partition
 * @param key          Message key
 * @param data         Message data
 */
public void produce(String topic, Integer partition, K key, V data)
{
    produce(topic, partition, key, data, null, (Callback)null);
}

/**
 * Producing messages
 *
 * @param topic        Topic
 * @param partition    partition
 * @param key          Message key
 * @param data         Message data
 * @param timestamp    timestamp
 */
public void produce(String topic, Integer partition, K key, V data, Long timestamp)
{
    produce(topic, partition, key, data, timestamp, (Callback)null);
}
/**
 * Producing messages
 *
 * @param topic        Topic
 * @param partition    partition
 * @param key          Message key
 * @param data         Message data
 * @param callback     callback
 */
public void produce(String topic, Integer partition, K key, V data, Callback callback)
{
    produce(topic, partition, key, data, null, callback);
}

public void produce(String topic, V data)
{
    produce(topic, null, null, data, null, (Callback)null);
}

/**
 * Producing messages
 *
 * @param topic        Topic
 * @param partition    partition
 * @param key          Message key
 * @param data         Message data
 * @param timestamp    timestamp
```



```
     * @param callback    callback
     */
    public void produce(String topic, Integer partition, K key, V data, Long timestamp, Callback
callback)
    {
        ProducerRecord<K, V> kafkaRecord =
                timestamp == null ? new ProducerRecord<K, V>(topic, partition, key, data)
                        : new ProducerRecord<K, V>(topic, partition, timestamp, key, data);
        produce(kafkaRecord, callback);
    }

    public void produce(ProducerRecord<K, V> kafkaRecord)
    {
        produce(kafkaRecord, (Callback)null);
    }

    public void produce(ProducerRecord<K, V> kafkaRecord, Callback callback)
    {
        producer.send(kafkaRecord, callback);
    }

    public void close()
    {
        producer.close();
    }

    /**
     * get classloader from thread context if no classloader found in thread
     * context return the classloader which has loaded this class
     *
     * @return classloader
     */
    public static ClassLoader getCurrentClassLoader()
    {
        ClassLoader classLoader = Thread.currentThread()
                .getContextClassLoader();
        if (classLoader == null)
        {
            classLoader = DmsProducer.class.getClassLoader();
        }
        return classLoader;
    }

    /**
     * Load configuration information from classpath.
     *
     * @param configFileName Configuration file name
     * @return Configuration information
     * @throws IOException
     */
    public static Properties loadFromClasspath(String configFileName) throws IOException
    {
        ClassLoader classLoader = getCurrentClassLoader();
        Properties config = new Properties();

        List<URL> properties = new ArrayList<URL>();
        Enumeration<URL> propertyResources = classLoader
                .getResources(configFileName);
        while (propertyResources.hasMoreElements())
        {
            properties.add(propertyResources.nextElement());
        }

        for (URL url : properties)
        {
            InputStream is = null;
            try
            {
                is = url.openStream();
```



```
            config.load(is);
        }
        finally
        {
            if (is != null)
            {
                is.close();
                is = null;
            }
        }
    }

    return config;
  }
}
```

Consuming Messages

- Test code

```
package com.dms.consumer;

import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.junit.Test;
import java.util.Arrays;

public class DmsConsumerTest {
   @Test
   public void testConsumer() throws Exception {
      DmsConsumer consumer = new DmsConsumer();
      consumer.consume(Arrays.asList("topic-0"));
      try {
         for (int i = 0; i < 10; i++){
            ConsumerRecords<Object, Object> records = consumer.poll(1000);
            System.out.println("the numbers of topic:" + records.count());
            for (ConsumerRecord<Object, Object> record : records)
            {
               System.out.println(record.toString());
            }
         }
      }catch (Exception e)
      {
         // TODO: Exception handling
         e.printStackTrace();
      }finally {
         consumer.close();
      }
   }
}
```

- Message consumption code

```
package com.dms.consumer;

import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import java.io.BufferedInputStream;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.URL;
import java.util.*;

public class DmsConsumer {

   public static final String CONFIG_CONSUMER_FILE_NAME = "dms.sdk.consumer.properties";

   private KafkaConsumer<Object, Object> consumer;

   DmsConsumer(String path)
```



```
{
    Properties props = new Properties();
    try {
        InputStream in = new BufferedInputStream(new FileInputStream(path));
        props.load(in);
    }catch (IOException e)
    {
        e.printStackTrace();
        return;
    }
    consumer = new KafkaConsumer<Object, Object>(props);
}

DmsConsumer()
{
    Properties props = new Properties();
    try {
        props = loadFromClasspath(CONFIG_CONSUMER_FILE_NAME);
    }catch (IOException e)
    {
        e.printStackTrace();
        return;
    }
    consumer = new KafkaConsumer<Object, Object>(props);
}
public void consume(List topics)
{
    consumer.subscribe(topics);
}

public ConsumerRecords<Object, Object> poll(long timeout)
{
    return consumer.poll(timeout);
}

public void close()
{
    consumer.close();
}

/**
 * get classloader from thread context if no classloader found in thread
 * context return the classloader which has loaded this class
 *
 * @return classloader
 */
public static ClassLoader getCurrentClassLoader()
{
    ClassLoader classLoader = Thread.currentThread()
            .getContextClassLoader();
    if (classLoader == null)
    {
        classLoader = DmsConsumer.class.getClassLoader();
    }
    return classLoader;
}

/**
 * Load configuration information from classpath.
 *
 * @param configFileName Configuration file name
 * @return Configuration information
 * @throws IOException
 */
public static Properties loadFromClasspath(String configFileName) throws IOException
{
    ClassLoader classLoader = getCurrentClassLoader();
    Properties config = new Properties();
```



```
        List<URL> properties = new ArrayList<URL>();
        Enumeration<URL> propertyResources = classLoader
                .getResources(configFileName);
        while (propertyResources.hasMoreElements())
        {
            properties.add(propertyResources.nextElement());
        }

        for (URL url : properties)
        {
            InputStream is = null;
            try
            {
                is = url.openStream();
                config.load(is);
            }
            finally
            {
                if (is != null)
                {
                    is.close();
                    is = null;
                }
            }
        }

        return config;
    }
}
```

3.2 Setting Up the Java Development Environment

With the information collected in **Collecting Connection Information** and the network environment prepared for Kafka clients, you can proceed to configuring Kafka clients. This section describes how to configure Kafka clients to produce and consume messages.

Preparing Tools

- Maven

  Apache Maven 3.0.3 or later can be downloaded from the **Maven official website**.

- JDK

  Java Development Kit1.8.111 or later can be downloaded from the **Oracle official website**.

  After the installation, configure the Java environment variables.

- IntelliJ IDEA

  IntelliJ IDEA can be downloaded from the **IntelliJ IDEA official website** and be installed.

Procedure

Step 1 Download **the demo package**.

Decompress the package to obtain the following files.

Table 3-1 Files in the demo package

File	Directory	Description
DmsConsumer.java	.\src\main\java\com\dms\consumer	API for consuming messages
DmsProducer.java	.\src\main\java\com\dms\producer	API for producing messages
dms.sdk.consumer.properties	.\src\main\resources	Configuration information for consuming messages
dms.sdk.producer.properties	.\src\main\resources	Configuration information for producing messages
client.truststore.jks	.\src\main\resources	SSL certificate, used for SASL connection
DmsConsumerTest.java	.\src\test\java\com\dms\consumer	Test code of consuming messages
DmsProducerTest.java	.\src\test\java\com\dms\producer	Test code of producing messages
pom.xml	.\	Maven configuration file, containing the Kafka client dependencies

Step 2 In IntelliJ IDEA, import the demo project.

The demo project is a Java project built in Maven. Therefore, you need the JDK and the Maven plugin in IDEA.

Figure 3-1 Click **Import Project**.

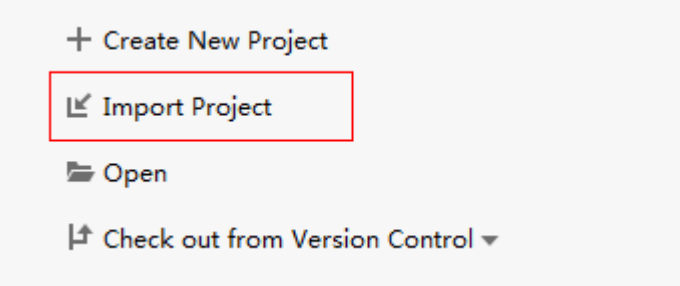


Figure 3-2 Choose **Maven**.

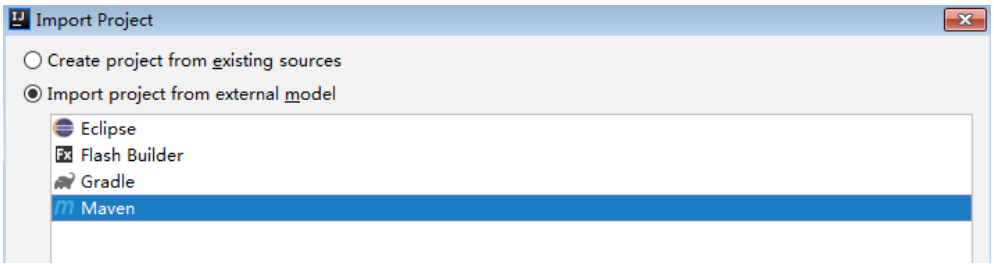
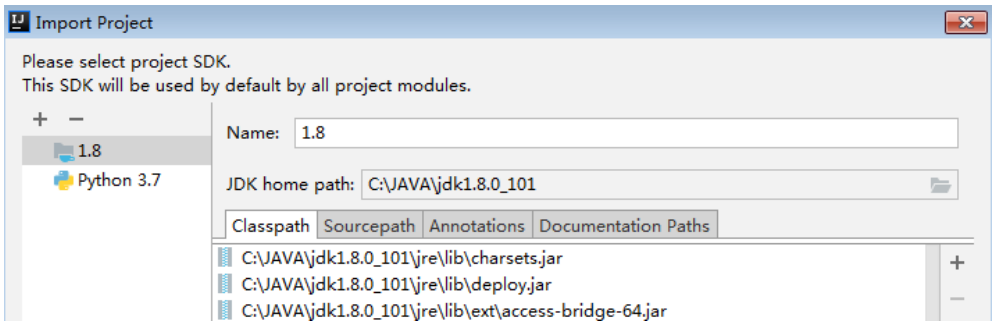
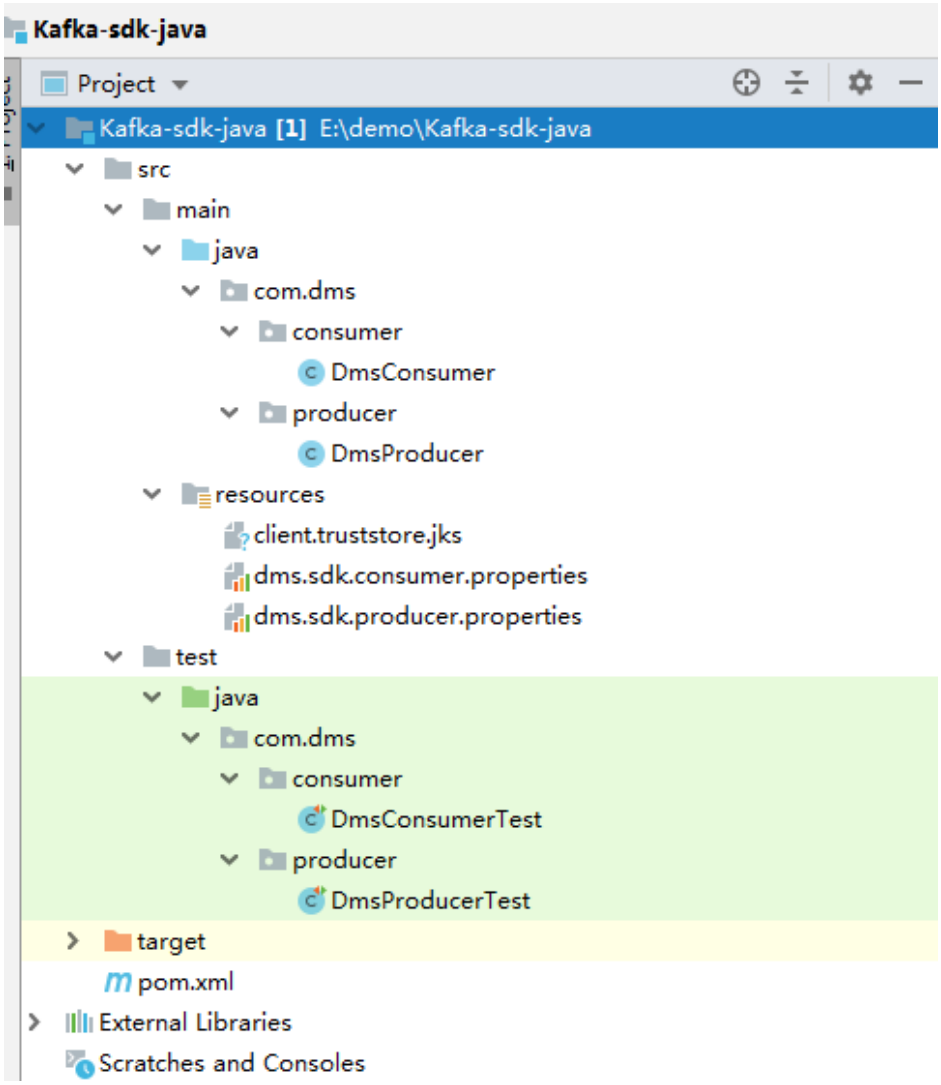


Figure 3-3 Select the JDK.



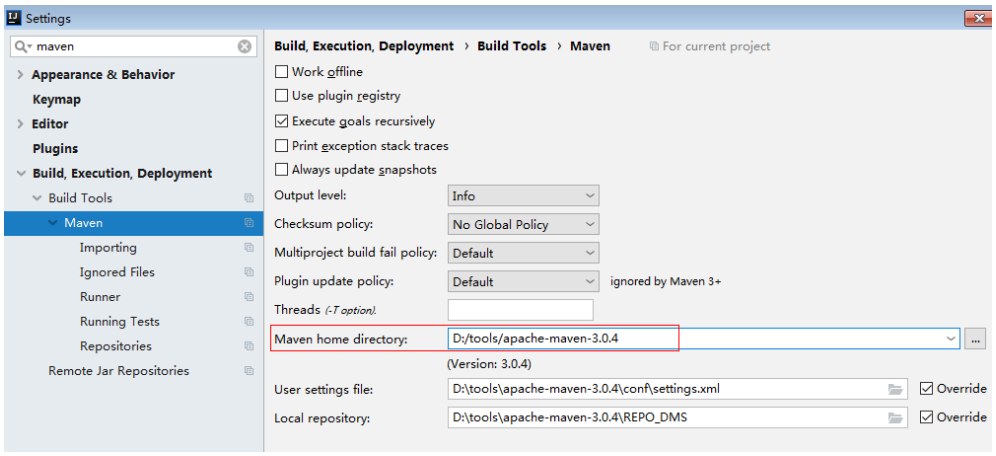
You can select other options or retain the default settings. Click **Finish**.

The demo project has been imported.



Step 3 Configure Maven.

Choose **Files** > **Settings**, set **Maven home directory** correctly, and select the required **settings.xml** file.



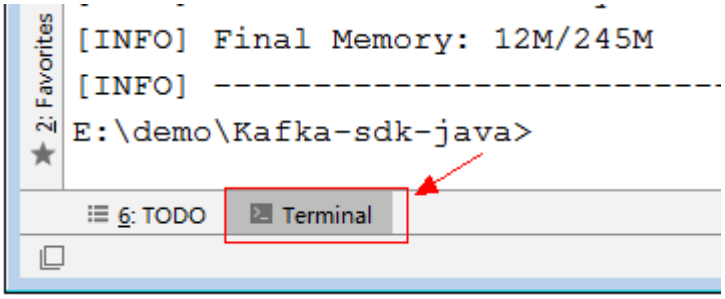
Step 4 Specify Kafka configurations.

The following is a configuration example for producing messages. Replace the information in bold with the actual values.

```
#The information in bold is specific to different Kafka instances and must be modified. Other parameters can also be added.
#The topic name is in the code for producing or consuming messages.
########################
#Information about Kafka brokers. ip:port are the connection addresses and ports used by the instance. The values can be obtained by referring to the "Collecting Connection Information" section. Example:
bootstrap.servers=100.xxx.xxx.87:909x,100.xxx.xxx.69:909x,100.xxx.xxx.155:909x
bootstrap.servers=ip1:port1,ip2:port2,ip3:port3
#Producer acknowledgement
acks=all
#Method of turning the key into bytes
key.serializer=org.apache.kafka.common.serialization.StringSerializer
#Method of turning the value into bytes
value.serializer=org.apache.kafka.common.serialization.StringSerializer
#Memory available to the producer for buffering
buffer.memory=33554432
#Number of retries
retries=0
########################
#Comment out the following parameters if SASL access is not enabled.
########################
#Configure the JAAS username and password. username and password are set when you enable Kafka SASL_SSL during instance creation.
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required \
    username="username" \
    password="password";
#SASL mechanism
sasl.mechanism=PLAIN
#Encryption protocol. Currently, only SASL_SSL is supported.
security.protocol=SASL_SSL
#Location of ssl.truststore
ssl.truststore.location=E:\\temp\\client.truststore.jks
#Password of ssl.truststore
ssl.truststore.password=dms@kafka
```

Step 5 In the down left corner of IDEA, click **Terminal**. In terminal, run the **mvn test** command to see how the demo project goes.

Figure 3-4 Opening terminal in IDEA



The following information is displayed for the producer:

```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.dms.producer.DmsProducerTest
produce msg:The msg is 0
produce msg:The msg is 1
produce msg:The msg is 2
produce msg:The msg is 3
produce msg:The msg is 4
produce msg:The msg is 5
produce msg:The msg is 6
produce msg:The msg is 7
produce msg:The msg is 8
produce msg:The msg is 9
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 138.877 sec
```

The following information is displayed for the consumer:

```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.dms.consumer.DmsConsumerTest
the numbers of topic:0
the numbers of topic:0
the numbers of topic:6
ConsumerRecord(topic = topic-0, partition = 2, offset = 0, CreateTime = 1557059377179, serialized key size = -1, serialized value size = 12, headers = RecordHeaders(headers = [], isReadOnly = false), key = null, value = The msg is 2)
ConsumerRecord(topic = topic-0, partition = 2, offset = 1, CreateTime = 1557059377195, serialized key size = -1, serialized value size = 12, headers = RecordHeaders(headers = [], isReadOnly = false), key = null, value = The msg is 5)
```

----End

4 Python

This section describes how to access a Kafka premium instance using a Kafka client in Python on the Linux CentOS, including how to install the client, and produce and consume messages.

Before getting started, ensure that you have collected the information listed in **Collecting Connection Information**.

Preparing the Environment

- Python

  Generally, Python is pre-installed in the system. Enter **python** in a CLI. If the following information is displayed, Python has already been installed.

  ```
  [root@ecs-test python-kafka]# python3
  Python 3.7.1 (default, Jul  5 2020, 14:37:24)
  [GCC 4.8.5 20150623 (Red Hat 4.8.5-39)] on linux
  Type "help", "copyright", "credits" or "license" for more information.
  >>>
  ```

  If Python is not installed, run the following command:

  yum install python

- Kafka clients in Python

  Run the following command to install a Python client of the recommended version:

  pip install kafka-python==2.0.1

Producing Messages

NOTE

Replace the following information in bold with the actual values.

- With SASL

  ```
  from kafka import KafkaProducer
  import ssl
  ##Connection information
  conf = {
      'bootstrap_servers': ["ip1:port1","ip2:port2","ip3:port3"],
      'topic_name': 'topic_name',
      'sasl_plain_username': 'username',
      'sasl_plain_password': 'password'
  }
  ```



```
context = ssl.create_default_context()
context = ssl.SSLContext(ssl.PROTOCOL_SSLv23)
context.verify_mode = ssl.CERT_REQUIRED
##Certificate
context.load_verify_locations("phy_ca.crt")

print('start producer')
producer = KafkaProducer(bootstrap_servers=conf['bootstrap_servers'],
                         sasl_mechanism="PLAIN",
                         ssl_context=context,
                         security_protocol='SASL_SSL',
                         sasl_plain_username=conf['sasl_plain_username'],
                         sasl_plain_password=conf['sasl_plain_password'])

data = bytes("hello kafka!", encoding="utf-8")
producer.send(conf['topic_name'], data)
producer.close()
print('end producer')
```

- Without SASL

```
from kafka import KafkaProducer

conf = {
    'bootstrap_servers': ["ip1:port1","ip2:port2","ip3:port3"],
    'topic_name': 'topic-name',
}

print('start producer')
producer = KafkaProducer(bootstrap_servers=conf['bootstrap_servers'])

data = bytes("hello kafka!", encoding="utf-8")
producer.send(conf['topic_name'], data)
producer.close()
print('end producer')
```

Consuming Messages

- With SASL

```
from kafka import KafkaConsumer
import ssl
##Connection information
conf = {
    'bootstrap_servers': ["ip1:port1","ip2:port2","ip3:port3"],
    'topic_name': 'topic_name',
    'sasl_plain_username': 'username',
    'sasl_plain_password': 'password',
    'consumer_id': 'consumer_id'
}

context = ssl.create_default_context()
context = ssl.SSLContext(ssl.PROTOCOL_SSLv23)
context.verify_mode = ssl.CERT_REQUIRED
##Certificate
context.load_verify_locations("phy_ca.crt")

print('start consumer')
consumer = KafkaConsumer(conf['topic_name'],
                         bootstrap_servers=conf['bootstrap_servers'],
                         group_id=conf['consumer_id'],
                         sasl_mechanism="PLAIN",
                         ssl_context=context,
                         security_protocol='SASL_SSL',
                         sasl_plain_username=conf['sasl_plain_username'],
                         sasl_plain_password=conf['sasl_plain_password'])

for message in consumer:
    print("%s:%d:%d: key=%s value=%s" % (message.topic, message.partition,message.offset,
message.key,message.value))
```



```
print('end consumer')
```

- Without SASL

  Replace the information in bold with the actual values.

```
from kafka import KafkaConsumer

conf = {
    'bootstrap_servers': ["ip1:port1","ip2:port2","ip3:port3"],
    'topic_name': 'topic-name',
    'consumer_id': 'consumer-id'
}

print('start consumer')
consumer = KafkaConsumer(conf['topic_name'],
                         bootstrap_servers=conf['bootstrap_servers'],
                         group_id=conf['consumer_id'])

for message in consumer:
    print("%s:%d:%d: key=%s value=%s" % (message.topic, message.partition,message.offset,
message.key,message.value))

print('end consumer')
```


5 Obtaining Kafka Clients

Kafka premium instances are fully compatible with open-source clients. You can obtain **clients in other programming languages** and access your instance as instructed by the official Kafka website.

6 Recommendations on Client Usage

- Producer parameters
  - Allow for retries in cases where messages fail to be sent.

    For example, allow for three retries by setting the value of **retries** to **3**.
  - The producer cannot block on callback functions. Otherwise, messages may fail to be sent.

    For messages that need to be send immediately, set **linger.ms** to **0**.

    Ensure that the producer has sufficient JVM memory to avoid blockages.
- Consumer parameters
  - Ensure that the owner thread does not exit abnormally. Otherwise, the client may fail to initiate consumption requests and the consumption will be blocked.
  - Use long polling to consume messages and do not close the consumer connection immediately after the consumption is completed. Otherwise, rebalancing will take place frequently, blocking consumption.
  - Ensure that the consumer polls at regular intervals (for example, every 200 ms) for it to keep sending heartbeats to the server. If the consumer stops sending heartbeats for long enough, the consumer session will time out and the consumer will be considered to have stopped. This will also block consumption.
  - Always close the consumer before exiting. Otherwise, consumers in the same group may block the **session.timeout.ms** time.
  - Set the timeout time for the consumer session to a reasonable value. For example, set **session.timeout.ms** to **30000** so that the timeout time is 30s.
  - The number of consumers cannot be greater than the number of partitions in the topic. Otherwise, some consumers may fail to poll for messages.
  - Commit messages after they have been processed. Otherwise, the messages may fail to be processed and cannot be polled for again.
  - Ensure that there is a maximum limit on the size of messages buffered locally to avoid an out-of-memory (OOM) situation.

- Kafka supports the exactly-once delivery. Therefore, ensure the idempotency of processing messages for services.

A Change History

Date	Description
2020-12-02	This issue is the first official release.

