TDMQ for CKafka

SDK Documentation

Product Documentation

Contents

SDK Documentation

SDK Overview

Last updated：2024-01-09 15:00:32

CKafka supports SDKs for multiple programming languages. Clients can access CKafka to send/receive messages in VPCs or over public network. The protocols for these two methods are as detailed below:

Network	VPC	Public Domain Name Access
Protocol	PLAINTEXT SASL_PLAINTEXT SASL_SSL (supported by Pro Edition)	SASL_PLAINTEXT SASL_SSL (supported by Pro Edition)

The specific usages of different SDKs are as follows:

SDK Type	Documentation
SDK for Java	VPC Access Public Network Access Through SASL_PLAINTEXT Public Network Access Through SASL_SSL VPC Access Through SASL_SCRAM
SDK for Python	VPC Access Public Network Access Through SASL_PLAINTEXT Public Network Access Through SASL_SSL
SDK for Go	VPC Access Public Network Access Through SASL_PLAINTEXT
SDK for PHP	VPC Access Public Network Access Through SASL_PLAINTEXT
SDK for C++	VPC Access Public Network Access Through SASL_PLAINTEXT
SDK for Node.js	VPC Access Public Network Access Through SASL_PLAINTEXT

SDK for Java VPC Access

Last updated：2024-01-09 15:00:32

Overview

This document describes how to access CKafka to receive/send messages with the SDK for Java in a VPC.

Prerequisites

You have installed JDK 1.8 or later
You have installed Maven 2.5 or later
You have downloaded the demo

Directions

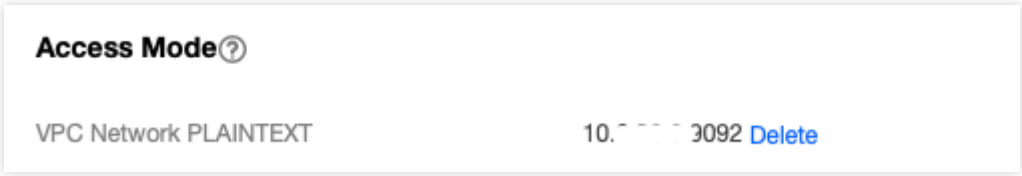
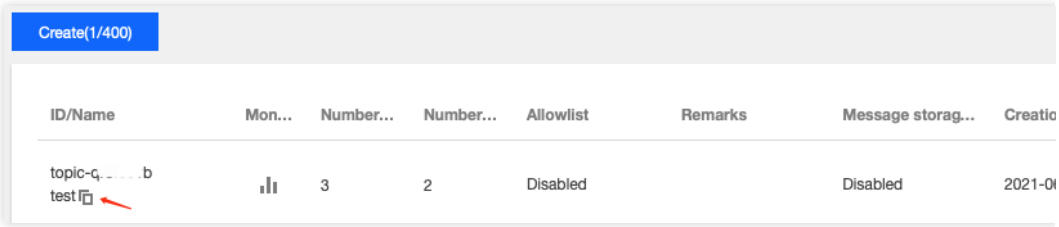
Step 1. Prepare configurations

1. Upload the `javakafkademo` in the downloaded demo to the Linux server.
2. Log in to the Linux server, enter the `javakafkademo` directory, and configure related parameters.
2.1 Add the following dependencies to the `pom.xml` file:

```
<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-clients</artifactId>
    <version>0.10.2.2</version>
</dependency>
```

2.2 Create a Kafka configuration file named `kafka.properties` .

```
## Configure the accessed network by copying the information in the **Network
bootstrap.servers=xx.xx.xx.xx:xxxx
## Configure the topic by copying the information on the **Topic Management**
topic=XXX
## Configure the consumer group as needed
group.id=XXX
```

Parameter	Description
bootstrap.servers	Accessed network, which can be copied in the **Network** column in the Access Mode section c console. 
topic	Topic name, which can be copied from the **Topic Management** page in the console. 
group.id	You can customize it. After the demo runs successfully, you can see the consumer on the **Con**

3. Create a configuration file loading program named `CKafkaConfigurer.java` .

```
public class CKafkaConfigurer {

    private static Properties properties;

    public synchronized static Properties getCKafkaProperties() {
        if (null != properties) {
            return properties;
        }
        //Obtain the content of the configuration file `kafka.properties`
        Properties kafkaProperties = new Properties();
        try {
            kafkaProperties.load(CKafkaProducerDemo.class.getClassLoader().ge
        } catch (Exception e){
            System.out.println("getCKafkaProperties error");
        }
        properties = kafkaProperties;
        return kafkaProperties;
```

```
            }
        }
```

Step 2. Send messages

1. Write a message production program named `CKafkaProducerDemo.java` .

```
public class CKafkaProducerDemo {

    public static void main(String args[]) {
        //Load `kafka.properties`
        Properties kafkaProperties = CKafkaConfigurer.getCKafkaProperties();

        Properties properties = new Properties();
        //Set the access point. Obtain the access point of the topic via the consol
        properties.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, kafkaProperties.get

        //Set the method for serializing Kafka messages. `StringSerializer` is used
        properties.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
                "org.apache.kafka.common.serialization.StringSerializer");
        properties.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
                "org.apache.kafka.common.serialization.StringSerializer");
        //Set the maximum request wait time
        properties.put(ProducerConfig.MAX_BLOCK_MS_CONFIG, 30 * 1000);
        //Set the number of retries for the client
        properties.put(ProducerConfig.RETRIES_CONFIG, 5);
        //Set the retry interval for the client.
        properties.put(ProducerConfig.RECONNECT_BACKOFF_MS_CONFIG, 3000);
        //Construct a producer object
        KafkaProducer<String, String> producer = new KafkaProducer<>(properties);

        //Construct a Kafka message
        String topic = kafkaProperties.getProperty("topic"); //Topic of the message
        String value = "this is ckafka msg value"; //Message content.

        try {
            //Batch obtaining future objects can speed up the process, but the batc
            List<Future<RecordMetadata>> futureList = new ArrayList<>(128);
            for (int i = 0; i < 10; i++) {
                //Send the message and obtain a future object
                ProducerRecord<String, String> kafkaMsg = new ProducerRecord<>(topi
                        value + ": " + i);
                Future<RecordMetadata> metadataFuture = producer.send(kafkaMsg);
                futureList.add(metadataFuture);

            }
```

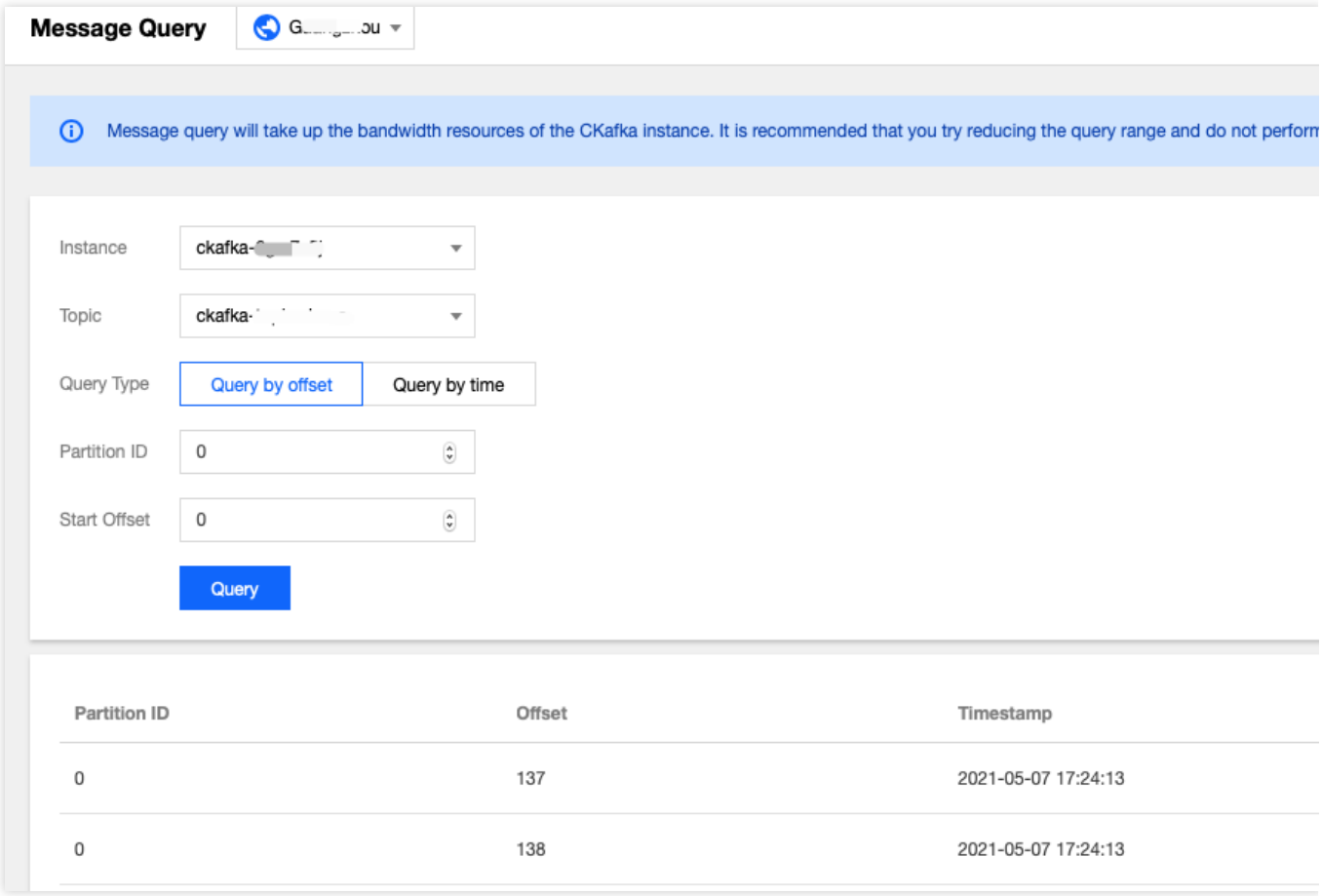
```
            producer.flush();
            for (Future<RecordMetadata> future : futureList) {
                //Sync the future object obtained
                RecordMetadata recordMetadata = future.get();
                System.out.println("produce send ok: " + recordMetadata.toString())
            }
        } catch (Exception e){
            //If the sending still fails after client internal retries, the system
            System.out.println("error occurred");
        }
    }
}
```

2. Compile and run `CKafkaProducerDemo.java` to send the message.

3. View the execution result.

```
Produce ok:ckafka-topic-demo-0@198
Produce ok:ckafka-topic-demo-0@199
```

4. On the **Topic Management** tab page on the instance details page in the CKafka console, select the topic, and click **More** > **Message Query** to view the message just sent.



Step 3. Consume messages

1. Create a program named `CKafkaConsumerDemo.java` for a consumer to subscribe to messages.

```
public class CKafkaConsumerDemo {

    public static void main(String args[]) {
        //Load `kafka.properties`
        Properties kafkaProperties = CKafkaConfigurer.getCKafkaProperties();

        Properties props = new Properties();
        //Set the access point. Obtain the access point of the topic via the consol
        props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, kafkaProperties.getPrope
        //Set the maximum interval between two polls
        //If the consumer does not return a heartbeat message within the interval,
        props.put(ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG, 30000);
        //Set the maximum number of messages that can be polled at a time
        //Do not set this parameter to an excessively large value. If polled messag
        props.put(ConsumerConfig.MAX_POLL_RECORDS_CONFIG, 30);
        //Set the method for deserializing messages
        props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
                "org.apache.kafka.common.serialization.StringDeserializer");
```

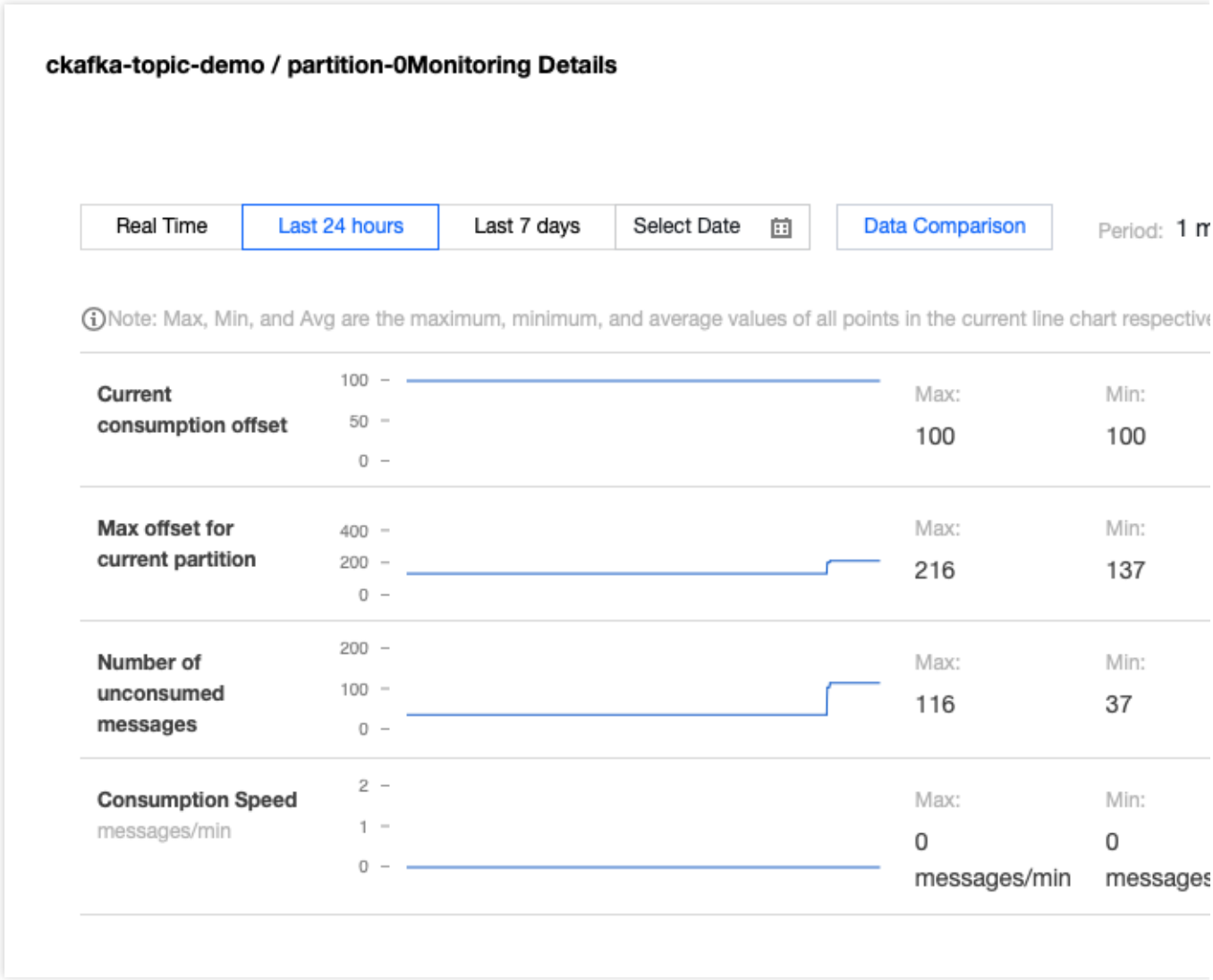
```
        props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
                "org.apache.kafka.common.serialization.StringDeserializer");
        //The instances in the same consumer group consume messages in load balanci
        props.put(ConsumerConfig.GROUP_ID_CONFIG, kafkaProperties.getProperty("grou
        //Create a consumer object, which means generating a consumer instance
        KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props);
        //Set one or more topics to which the consumer group subscribes
        //You are advised to configure consumer instances with the same `GROUP_ID_C
        List<String> subscribedTopics = new ArrayList<>();
        //If you want to subscribe to multiple topics, add the topics here
        //You must create the topics in the console in advance.
        String topicStr = kafkaProperties.getProperty("topic");
        String[] topics = topicStr.split(",");
        for (String topic : topics) {
            subscribedTopics.add(topic.trim());
        }
        consumer.subscribe(subscribedTopics);

        //Consume messages in loop
        while (true) {
            try {
                ConsumerRecords<String, String> records = consumer.poll(1000);
                //All messages must be consumed before the next poll, and the total
                //You are advised to create a separate thread to consume messages a
                for (ConsumerRecord<String, String> record : records) {
                    System.out.println(
                            String.format("Consume partition:%d offset:%d", record.
                }
            } catch (Exception e){
                System.out.println("consumer error!");
            }
        }
    }
}
```

2. Compile and run `CKafkaConsumerDemo.java` to consume messages.

3. View the execution result.

```
Consume partition:0 offset:298
Consume partition:0 offset:299
```

4. On the **Consumer Group** tab page in the CKafka console, select the consumer group name, enter the topic name, and click **View Details** to view the consumption details.



VPC Access Through SASL_SCRAM

Last updated：2024-01-09 15:00:32

Overview

This document describes how to access CKafka to receive/send messages with the SDK for Java through SASL_SCRAM in a VPC.

Note:

Access through SASL_SCRAM is only supported for instances on v2.4.1 in the Beijing region. For other regions or existing instances, submit a ticket to apply for this access mode.

Prerequisites

You have installed JDK 1.8 or later.

You have installed Maven 2.5 or later.

You have configured an ACL policy.

You have downloaded the demo.

Directions

Step 1. Create resources in the console

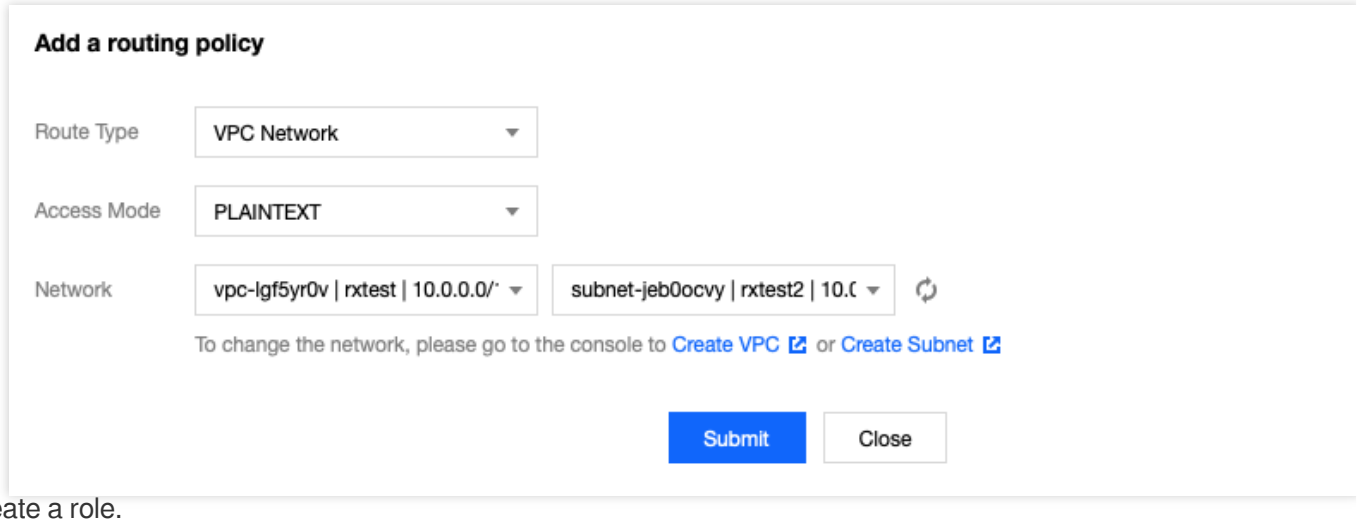
1. Create an access point.

1.1 On the Instance List page in the CKafka console, click the target instance ID to enter the instance details page.

1.2 In **Basic Info** > **Access Mode**, click **Add a routing policy**. In the pop-up window, configure the following items:

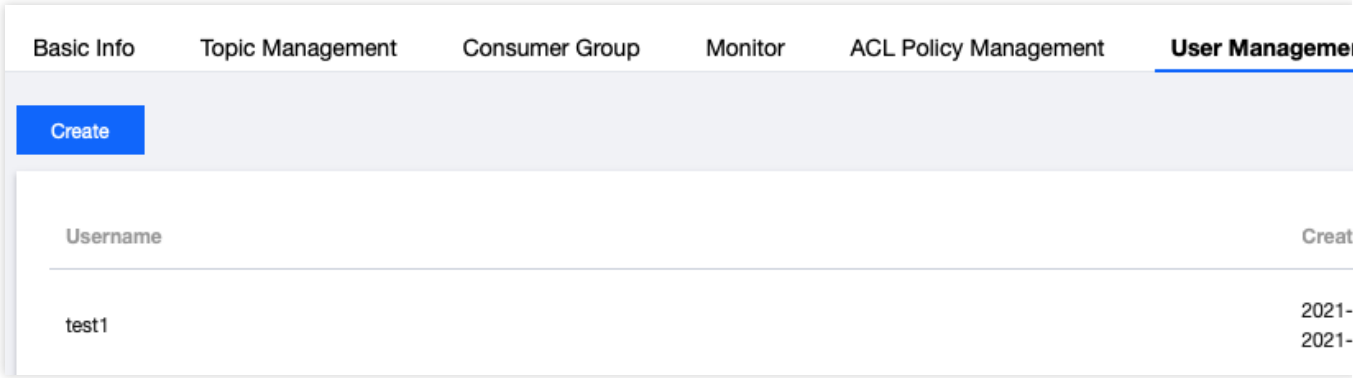
Route Type: Select "Public domain name access".

Access Mode: Select "SASL_SCRAM".



2. Create a role.

In **ACL Policy Management** > **User Management**, create a role and set the password.



3. Create a topic.

Create a topic on the **Topic Management** tab as instructed in Creating Topic.

Step 2. Add the configuration file

1. Add the following Java dependent library information to the `pom.xml` file:

```
<dependencies>
   <dependency>
      <groupId>org.apache.kafka</groupId>
      <artifactId>kafka-clients</artifactId>
      <version>2.1.0</version>
   </dependency>
   <dependency>
      <groupId>org.slf4j</groupId>
      <artifactId>slf4j-api</artifactId>
      <version>1.7.5</version>
   </dependency>
   <dependency>
```

```
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-simple</artifactId>
        <version>1.6.4</version>
    </dependency>
</dependencies>
```

2. Create a JAAS configuration file named `ckafka_client_jaas.conf` and modify it with the user created in **ACL Policy Management** > **User Management**.

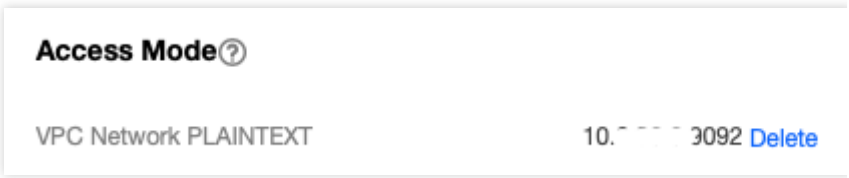
```
KafkaClient {
org.apache.kafka.common.security.plain.PlainLoginModule required
username="yourinstance#yourusername"
password="yourpassword";
};
```

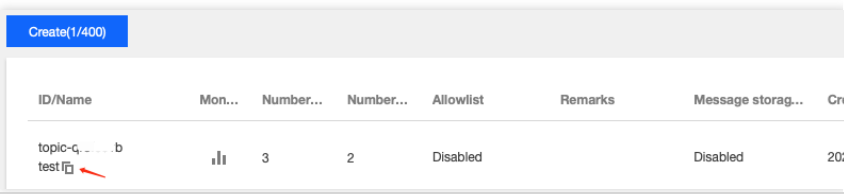
Note:

Set `username` to a value in the format of `instance ID` + `#` + `configured username` , and `password` to a configured password.

3. Create a CKafka configuration file named `kafka.properties` .

```
## Configure the accessed network by copying the information in the **Network** col
bootstrap.servers=xx.xx.xx.xx:xxxx
## Configure the topic by copying the information on the **Topic Management** page
topic=XXX
## Configure the consumer group as needed
group.id=XXX
## SASL Configuration
java.security.auth.login.config.plain=/xxxx/ckafka_client_jaas.conf
```

Parameter	Description
bootstrap.servers	Accessed network, which can be copied in the **Network** column in the **Acce** instance details page in the console. 
topic	Topic name, which can be copied in **Topic Management** on the instance de

	
group.id	You can customize it. After the demo runs successfully, you can see the cons page.
java.security.auth.login.config.plain	Enter the path of the JAAS configuration file `ckafka_client_jaas.con`

4. Create the configuration file loading program `CKafkaConfigurer.java` .

```
public class CKafkaConfigurer {

 private static Properties properties;

 public static void configureSaslPlain() {
     // If you have used the `-D` parameter or another method to set the path, do
     if (null == System.getProperty("java.security.auth.login.config")) {
         // Replace `XXX` with your own path
         System.setProperty("java.security.auth.login.config",
                 getCKafkaProperties().getProperty("java.security.auth.login.confi
     }
 }

 public synchronized static Properties getCKafkaProperties() {
     if (null != properties) {
         return properties;
     }
     // Get the content of the configuration file `kafka.properties`.
     Properties kafkaProperties = new Properties();
     try {
         kafkaProperties.load(CKafkaProducerDemo.class.getClassLoader().getResourc
     } catch (Exception e){
         System.out.println("getCKafkaProperties error");
     }
     properties = kafkaProperties;
     return kafkaProperties;
 }
}
```

Step 3. Send messages

1. Create a message sending program named `KafkaSaslProducerDemo.java` .

```
public class KafkaSaslProducerDemo {

public static void main(String[] args) {
   // Set the path of the JAAS configuration file.
   CKafkaConfigurer.configureSaslPlain();

   // Load `kafka.properties`.
   Properties kafkaProperties = CKafkaConfigurer.getCKafkaProperties();

   Properties props = new Properties();
   // Set the access point. Get the access point of the corresponding topic in the
   props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
           kafkaProperties.getProperty("bootstrap.servers"));

   //
   // Access through SASL_SCRAM
   //
   props.put(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG, "SASL_PLAINTEXT");
   // Select `PLAIN` for the SASL mechanism
   props.put(SaslConfigs.SASL_MECHANISM, "SCRAM-SHA-256");

   // Set the method for serializing Kafka messages.
   props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
           "org.apache.kafka.common.serialization.StringSerializer");
   props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
           "org.apache.kafka.common.serialization.StringSerializer");
   // Set the maximum request wait time.
   props.put(ProducerConfig.MAX_BLOCK_MS_CONFIG, 30 * 1000);
   // Set the number of retries for the client.
   props.put(ProducerConfig.RETRIES_CONFIG, 5);
   // Set the internal retry interval for the client.
   props.put(ProducerConfig.RECONNECT_BACKOFF_MS_CONFIG, 3000);
   // If `ack` is 0, the producer will not wait for the acknowledgment from the br
   // If `ack` is 1, the broker leader will directly return `ack` without waiting
   // If `ack` is `all`, the broker leader will return `ack` only after receiving
   props.put(ProducerConfig.ACKS_CONFIG, "all");
   // Construct a producer object. Note: The producer object is thread-safe. Gener
   KafkaProducer<String, String> producer = new KafkaProducer<>(props);

   // Construct a CKafka message.
   String topic = kafkaProperties.getProperty("topic"); // Topic of the message. E
   String value = "this is ckafka msg value"; // Message content

   try {
      // Obtaining the future objects in batches can accelerate the speed. Do not
      List<Future<RecordMetadata>> futures = new ArrayList<>(128);
```

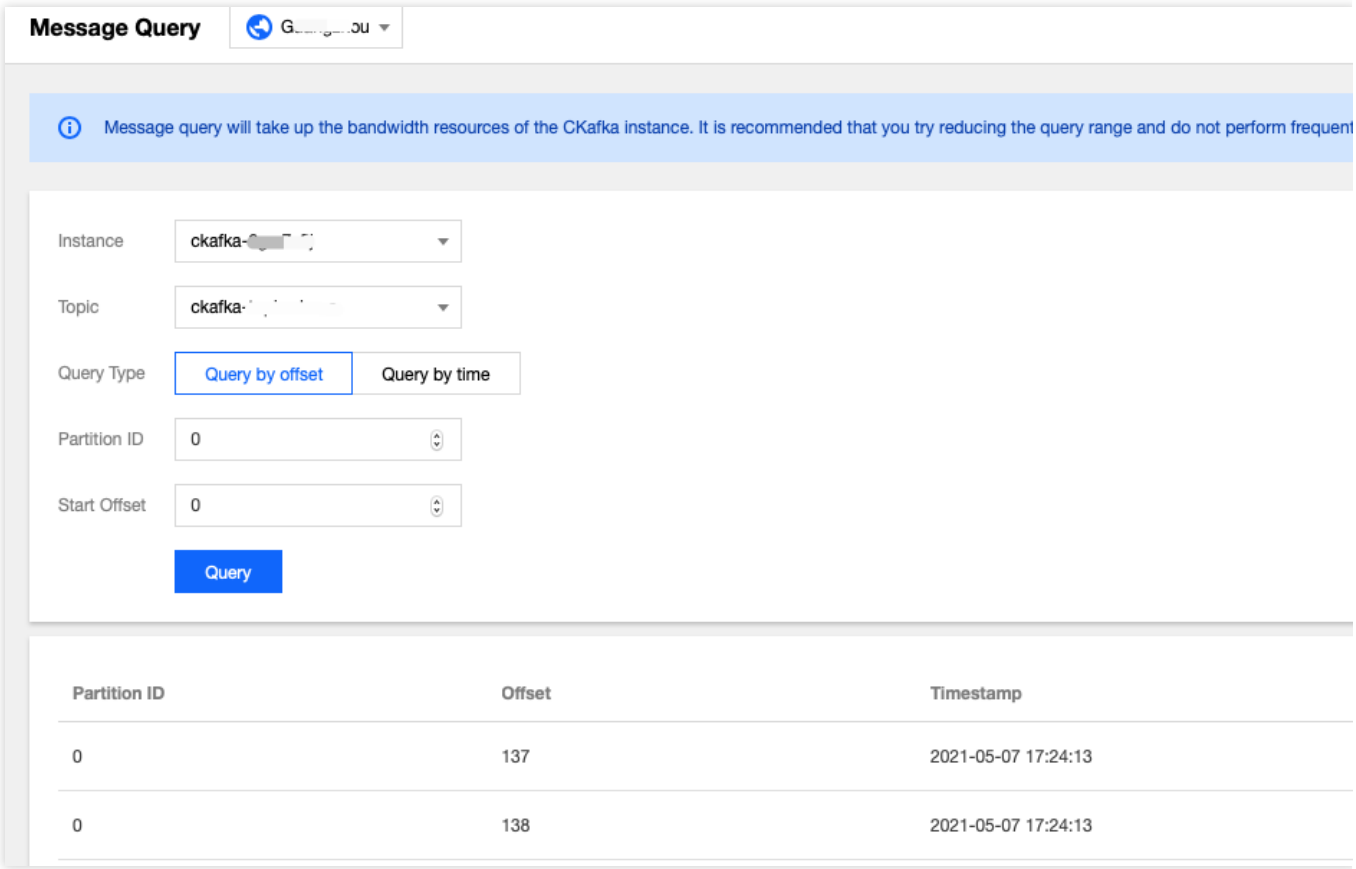
```
        for (int i = 0; i < 100; i++) {
            // Send the message and obtain a future object.
            ProducerRecord<String, String> kafkaMessage = new ProducerRecord<>(topic,
                    value + ": " + i);
            Future<RecordMetadata> metadataFuture = producer.send(kafkaMessage);
            futures.add(metadataFuture);

        }
        producer.flush();
        for (Future<RecordMetadata> future : futures) {
            // Sync the obtained future object.
            RecordMetadata recordMetadata = future.get();
            System.out.println("Produce ok:" + recordMetadata.toString());
        }

    } catch (Exception e){
        // If the sending still fails after the internal retries in the client, the
        System.out.println("error occurred");
    }
}
}
```

2. Compile and run `KafkaSaslProducerDemo.java` to send the message.

3. View the execution result (output).

```
Produce ok:ckafka-topic-demo-0@198
Produce ok:ckafka-topic-demo-0@199
```

4. On the **Topic Management** tab page on the instance details page in the CKafka console, select the target topic and click **More** > **Message Query** to view the message just sent.



Step 4. Consume messages

1. Create a program named `KafkaSaslConsumerDemo.java` for a consumer to subscribe to messages.

```
public class KafkaSaslConsumerDemo {

public static void main(String[] args) {
    // Set the path of the JAAS configuration file.
    CKafkaConfigurer.configureSaslPlain();

    // Load `kafka.properties`.
    Properties kafkaProperties = CKafkaConfigurer.getCKafkaProperties();

    Properties props = new Properties();
    // Set the access point. Obtain the access point of the corresponding topic in
    props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
            kafkaProperties.getProperty("bootstrap.servers"));

    //
    // Access through SASL_SCRAM
    //
    props.put(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG, "SASL_PLAINTEXT");
    // Select `PLAIN` for the SASL mechanism
```

```
props.put(SaslConfigs.SASL_MECHANISM, "SCRAM-SHA-256");

// Set the consumer timeout period.
// If the consumer does not return a heartbeat message within the interval, the
props.put(ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG, 30000);
// Set the maximum time interval between two polls.
// Before v0.10.1.0, these two concepts were mixed and were both represented by
props.put(ConsumerConfig.MAX_POLL_INTERVAL_MS_CONFIG, 30000);
// Set the maximum number of messages that can be polled at a time.
// Do not set this parameter to an excessively large value. If polled messages
props.put(ConsumerConfig.MAX_POLL_RECORDS_CONFIG, 30);
// Set the method for deserializing messages.
props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
        "org.apache.kafka.common.serialization.StringDeserializer");
props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
        "org.apache.kafka.common.serialization.StringDeserializer");
// Set the consumer group for the current consumer instance. You need to apply
// The instances in the same consumer group consume messages in load balancing
props.put(ConsumerConfig.GROUP_ID_CONFIG, kafkaProperties.getProperty("group.id
// Construct a consumer object. This generates a consumer instance.
KafkaConsumer<String, String> consumer = new KafkaConsumer<String, String>(prop
// Set one or more topics to which the consumer group subscribes.
// We recommend that you configure consumer instances with the same `GROUP_ID_C
List<String> subscribedTopics = new ArrayList<String>();
// If you want to subscribe to multiple topics, add the topics here.
// You must create these topics in the console in advance.
String topicStr = kafkaProperties.getProperty("topic");
String[] topics = topicStr.split(",");
for (String topic : topics) {
   subscribedTopics.add(topic.trim());
}
consumer.subscribe(subscribedTopics);

// Consume messages in loop
while (true){
   try {
      ConsumerRecords<String, String> records = consumer.poll(1000);
      // All messages must be consumed before the next poll, and the total dura
      for (ConsumerRecord<String, String> record : records) {
         System.out.println(
                 String.format("Consume partition:%d offset:%d", record.partiti
                         record.offset()));
      }
   } catch (Exception e){
      System.out.println("consumer error!");
   }
}
```

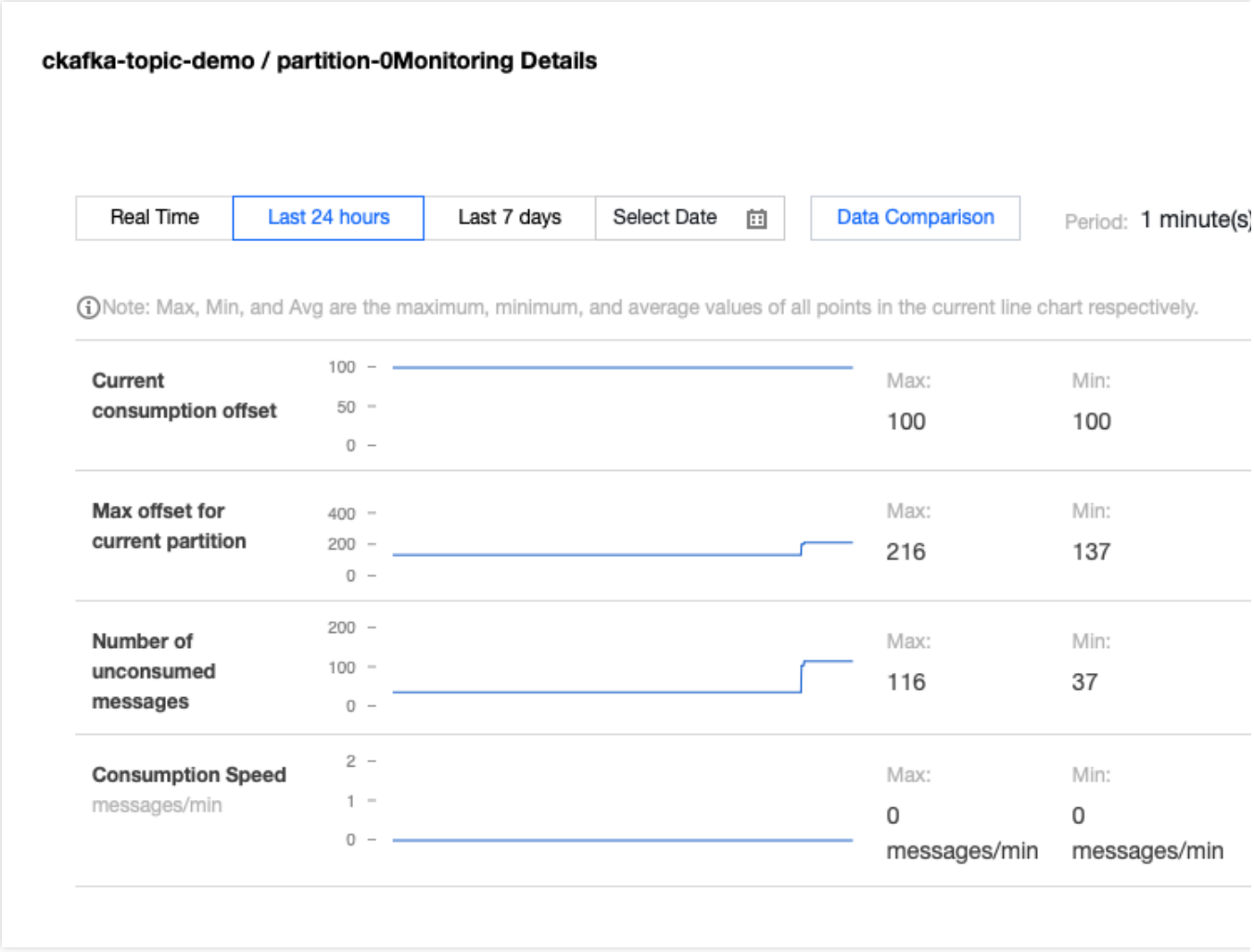
```
    }
}
```

2. Compile and run `KafkaSaslConsumerDemo.java` to consume the message.

3. View the execution result.

```
Consume partition:0 offset:298
Consume partition:0 offset:299
```

4. On the **Consumer Group** page in the Ckafka console, click the triangle icon on the left of the target consumer group name, enter the topic name in the search box, and click **View Details** to view the consumption details.



Public Network Access Through SASL_PLAINTEXT

Last updated：2024-01-09 15:00:33

Overview

This document describes how to access CKafka to receive/send messages with the SDK for Java through SASL_PLAINTEXT on the public network.

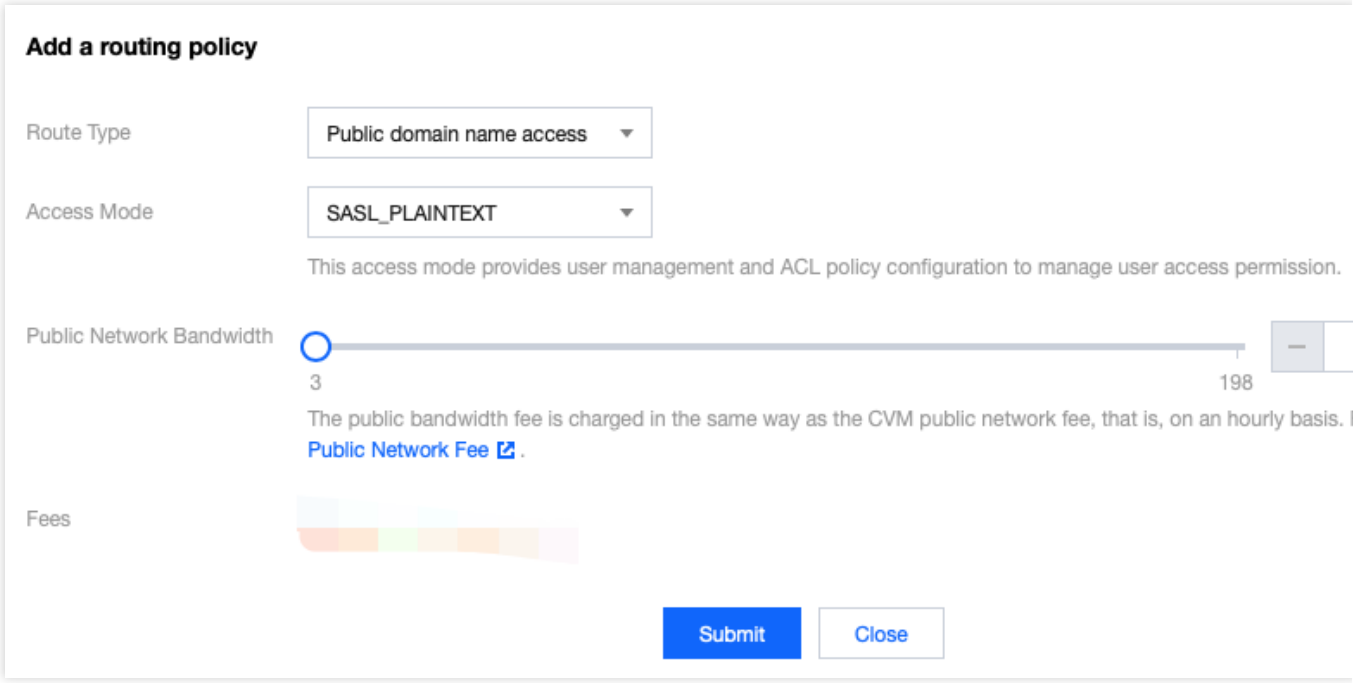
Prerequisites

You have installed JDK 1.8 or later
You have installed Maven 2.5 or later
You have configured an ACL policy
You have downloaded the demo

Directions

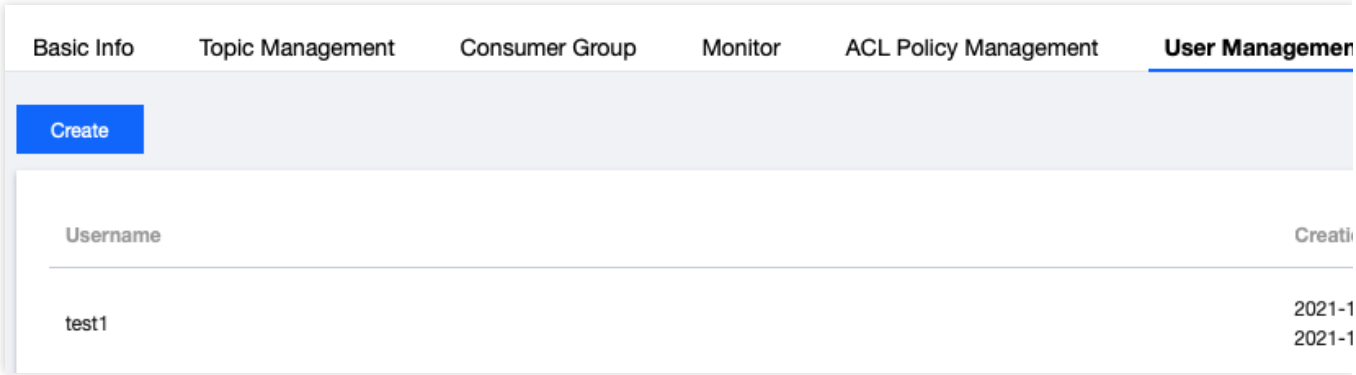
Step 1. Create resources in the console

1. Create an access point.
1.1 On the **Instance List** page in the CKafka console, click the target instance ID to enter the instance details page.
1.2 In **Basic Info** > **Access Mode**, click **Add a routing policy**. In the pop-up window, select `Route Type: Public domain name access` , `Access Mode: SASL_PLAINTEXT` .



2. Create a role.

On the **User Management** tab page, create a role and set the password.



3. Create a topic.

Create a topic on the **Topic Management** tab page as instructed in Topic Management.

Step 2. Add the configuration file

1. Add the following dependencies to the `pom.xml` file:

```
<dependencies>
   <dependency>
      <groupId>org.apache.kafka</groupId>
      <artifactId>kafka-clients</artifactId>
      <version>2.1.0</version>
   </dependency>
```

```
    <dependency>
       <groupId>org.slf4j</groupId>
       <artifactId>slf4j-api</artifactId>
       <version>1.7.5</version>
    </dependency>
    <dependency>
       <groupId>org.slf4j</groupId>
       <artifactId>slf4j-simple</artifactId>
       <version>1.6.4</version>
    </dependency>
</dependencies>
```

2. Create a JAAS configuration file named `ckafka_client_jaas.conf` and modify it with the user created on the **User Management** tab page.

```
KafkaClient {
org.apache.kafka.common.security.plain.PlainLoginModule required
username="yourinstance#yourusername"
password="yourpassword";
};
```

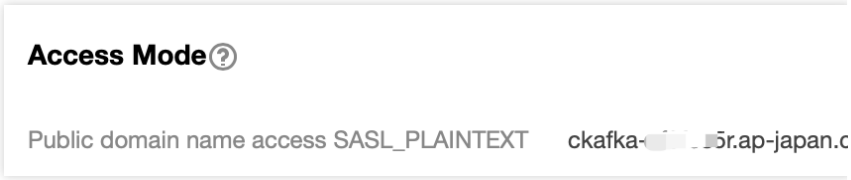
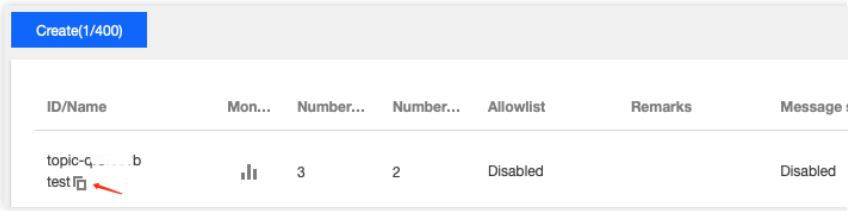
Note:

Set `username` to a value in the format of `instance ID` + `#` + `configured username`, and `password` to a configured password.

3. Create a Kafka configuration file named `kafka.properties`.

```
## Configure the accessed network by copying the information in the **Network** col
bootstrap.servers=ckafka-xxxxxxx
## Configure the topic by copying the information on the **Topic Management** page
topic=XXX
## Configure the consumer group as needed
group.id=XXX
## SASL configuration
java.security.auth.login.config.plain=/xxxx/ckafka_client_jaas.conf
```

Parameter	Description
bootstrap.servers	Accessed network, which can be copied in the **Network** column in the **Acce** page in the console.

	
topic	Topic name, which can be copied in **Topic Management** on the instance de 
group.id	You can customize it. After the demo runs successfully, you can see the cons
java.security.auth.login.config.plain	Enter the path of the JAAS configuration file `ckafka_client_jaas.con`

4. Create a configuration file loading program named `CKafkaConfigurer.java` .

```
public class CKafkaConfigurer {

    private static Properties properties;

    public static void configureSaslPlain() {
        //If you have used the `-D` parameter or another method to set the path, do
        if (null == System.getProperty("java.security.auth.login.config")) {
            // Replace `XXX` with your own path
            System.setProperty("java.security.auth.login.config",
                    getCKafkaProperties().getProperty("java.security.auth.login.con
        }
    }

    public synchronized static Properties getCKafkaProperties() {
        if (null != properties) {
            return properties;
        }
        //Obtain the content of the configuration file `kafka.properties`
        Properties kafkaProperties = new Properties();
        try {
            kafkaProperties.load(CKafkaProducerDemo.class.getClassLoader().getResou
        } catch (Exception e){
```

```
            System.out.println("getCKafkaProperties error");
        }
        properties = kafkaProperties;
        return kafkaProperties;
    }
}
```

Step 3. Send messages

1. Create a message sending program named `KafkaSaslProducerDemo.java` .

```
public class KafkaSaslProducerDemo {

public static void main(String[] args) {
    //Set the path to the JAAS configuration file
    CKafkaConfigurer.configureSaslPlain();

    //Load `kafka.properties`
    Properties kafkaProperties = CKafkaConfigurer.getCKafkaProperties();

    Properties props = new Properties();
    //Set the access point. Obtain the access point of the topic via the console
    props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
            kafkaProperties.getProperty("bootstrap.servers"));

    //
    //  Public network access through SASL_PLAINTEXT
    //
    props.put(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG, "SASL_PLAINTEXT");
    //  Select `PLAIN` for the SASL mechanism.
    props.put(SaslConfigs.SASL_MECHANISM, "PLAIN");

    //Set the method for serializing Kafka messages
    props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
            "org.apache.kafka.common.serialization.StringSerializer");
    props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
            "org.apache.kafka.common.serialization.StringSerializer");
    //Set the maximum request wait time
    props.put(ProducerConfig.MAX_BLOCK_MS_CONFIG, 30 * 1000);
    //Set the number of retries for the client
    props.put(ProducerConfig.RETRIES_CONFIG, 5);
    //Set the retry interval for the client
    props.put(ProducerConfig.RECONNECT_BACKOFF_MS_CONFIG, 3000);
    // If `ack` is 0, the producer will not wait for acknowledgment from the brok
    // If `ack` is 1, the broker leader will directly return `ack` without waitin
    // If `ack` is `all`, the broker leader will return `ack` only after receivin
```

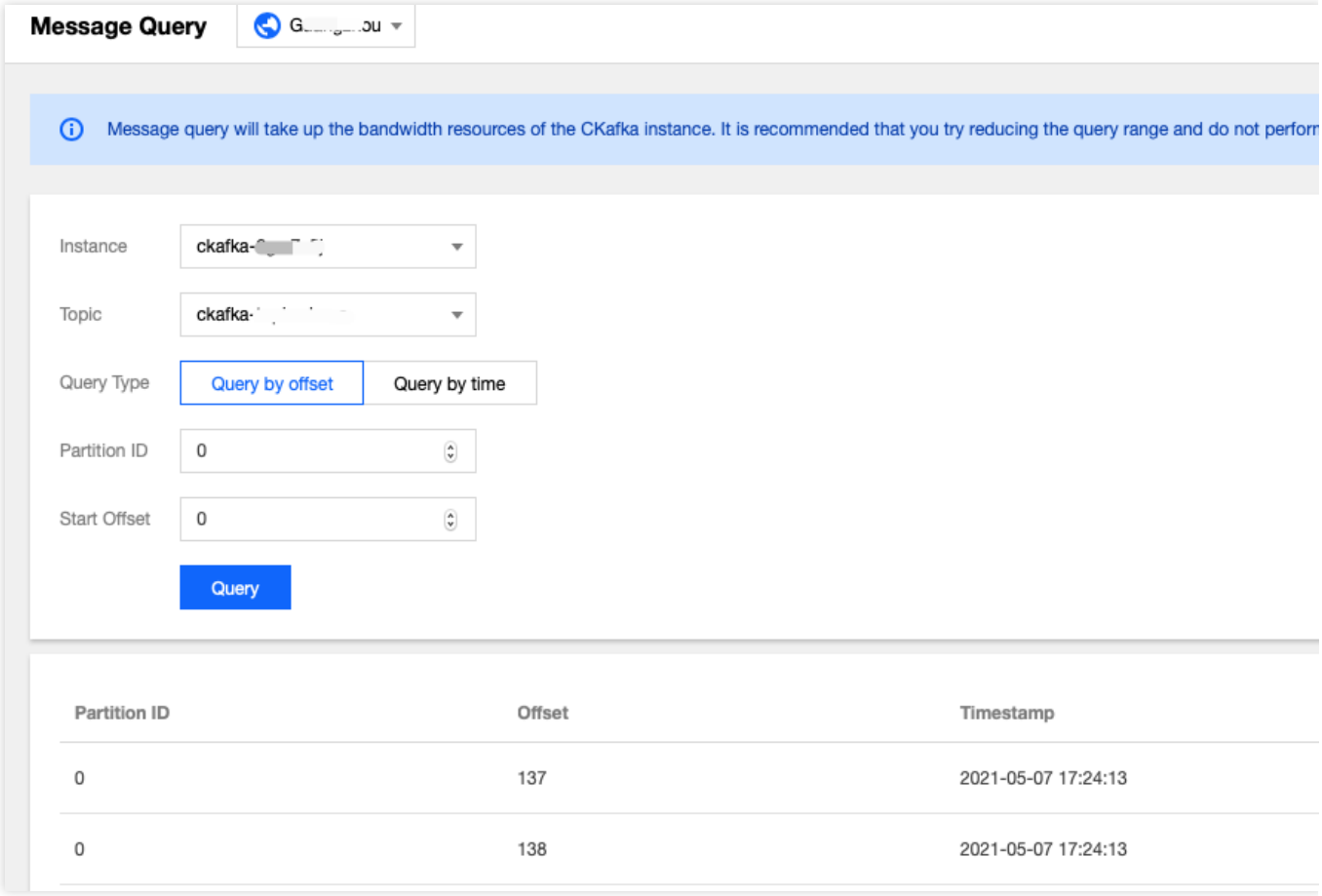
```
        props.put(ProducerConfig.ACKS_CONFIG, "all");
        //Create a producer object. Note: a producer object is thread-safe, and gener
        KafkaProducer<String, String> producer = new KafkaProducer<>(props);

        //Create a Kafka message
        String topic = kafkaProperties.getProperty("topic"); //Topic of the message.
        String value = "this is ckafka msg value"; //Content of the message.

        try {
            //Batch obtaining future objects can speed up the process, but the batch s
            List<Future<RecordMetadata>> futures = new ArrayList<>(128);
            for (int i = 0; i < 100; i++) {
                //Send the message and obtain a future object
                ProducerRecord<String, String> kafkaMessage = new ProducerRecord<>(topi
                        value + ": " + i);
                Future<RecordMetadata> metadataFuture = producer.send(kafkaMessage);
                futures.add(metadataFuture);

            }
            producer.flush();
            for (Future<RecordMetadata> future : futures) {
                //Sync the result that the future object is obtained
                RecordMetadata recordMetadata = future.get();
                System.out.println("Produce ok:" + recordMetadata.toString());
            }
        } catch (Exception e){
            //If the sending still fails after client internal retries, the system nee
            System.out.println("error occurred");
        }
    }
}
```

2. Compile and run `KafkaSaslProducerDemo.java` to send messages.

3. View the execution result (output).

```
Produce ok:ckafka-topic-demo-0@198
Produce ok:ckafka-topic-demo-0@199
```

4. On the **Topic Management** tab page on the instance details page in the CKafka console, select the target topic, and click **More** > **Message Query** to view the message just sent.



Step 4. Consume messages

1. Create a program named `KafkaSaslConsumerDemo.java` for a single consumer to subscribe to messages.

```
public class KafkaSaslConsumerDemo {

   public static void main(String[] args) {
      //Set the path to the JAAS configuration file
      CKafkaConfigurer.configureSaslPlain();

      //Load `kafka.properties`
      Properties kafkaProperties = CKafkaConfigurer.getCKafkaProperties();

      Properties props = new Properties();
      //Set the access point. Obtain the access point of the topic via the console.
      props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
              kafkaProperties.getProperty("bootstrap.servers"));

      //
      //  Public network access through SASL_PLAINTEXT
      //
      props.put(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG, "SASL_PLAINTEXT");
```

```
//  Select `PLAIN` for the SASL mechanism
props.put(SaslConfigs.SASL_MECHANISM, "PLAIN");

// Set the consumer timeout period
//If the consumer does not return a heartbeat message within the interval, th
props.put(ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG, 30000);
// Set the maximum time interval between two polls
// Before v0.10.1.0, these two concepts were mixed and both represented by `s
props.put(ConsumerConfig.MAX_POLL_INTERVAL_MS_CONFIG, 30000);
//Set the maximum number of messages that can be polled at a time
//Do not set this parameter to an excessively large value. If polled messages
props.put(ConsumerConfig.MAX_POLL_RECORDS_CONFIG, 30);
//Set the method for deserializing messages
props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
        "org.apache.kafka.common.serialization.StringDeserializer");
props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
        "org.apache.kafka.common.serialization.StringDeserializer");
//Set the consumer group of the current consumer instance after you apply for
//The instances in the same consumer group consume messages in load balancing
props.put(ConsumerConfig.GROUP_ID_CONFIG, kafkaProperties.getProperty("group.
//Create a consumer object, which means generating a consumer instance
KafkaConsumer<String, String> consumer = new KafkaConsumer<String, String>(pr
//Set one or more topics to which the consumer group subscribes
//You are advised to subscribe to the same topics if the values of GROUP_ID_C
List<String> subscribedTopics = new ArrayList<String>();
//If you want to subscribe to multiple topics, add the topics here
//You must create the topics in the console in advance
String topicStr = kafkaProperties.getProperty("topic");
String[] topics = topicStr.split(",");
for (String topic : topics) {
   subscribedTopics.add(topic.trim());
}
consumer.subscribe(subscribedTopics);

//Consume messages in loop
while (true) {
   try {
      ConsumerRecords<String, String> records = consumer.poll(1000);
      //All messages must be consumed before the next poll, and the total dur
      for (ConsumerRecord<String, String> record : records) {
         System.out.println(
                 String.format("Consume partition:%d offset:%d", record.parti
                         record.offset()));
      }
   } catch (Exception e){
      System.out.println("consumer error!");
   }
```

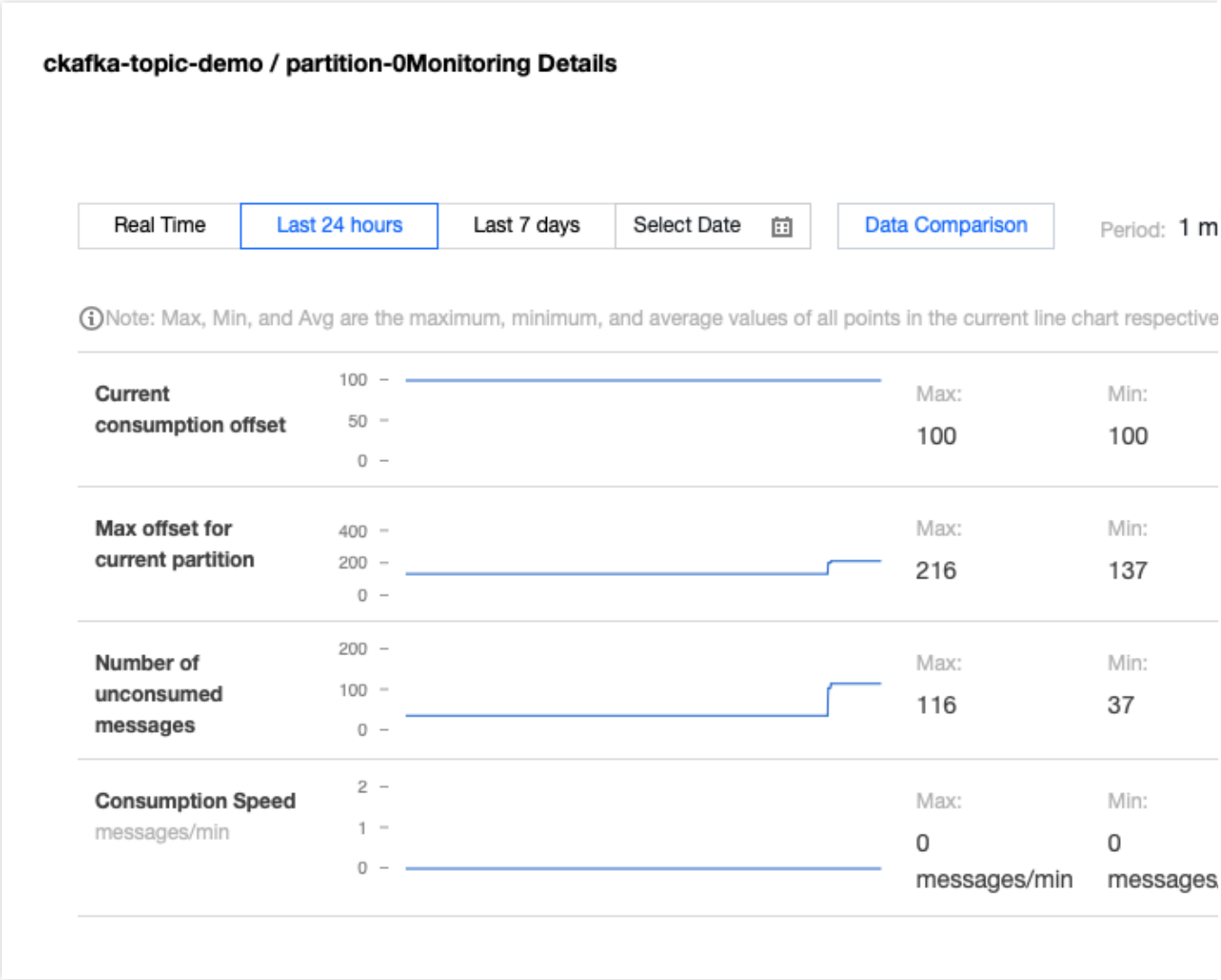
```
        }
    }
}
```

2. Compile and run `KafkaSaslConsumerDemo.java` to consume messages.

3. View the execution result (output).

```
Consume partition:0 offset:298
Consume partition:0 offset:299
```

4. On the **Consumer Group** page in the Ckafka console, select the consumer group name, enter the topic name, and click **Query Details** to view the consumption details.



Access Through SASL_SSL

Last updated：2024-09-20 11:13:01

Overview

This document describes how to access CKafka to send/receive messages with the SDK for Java through SASL_SSL on the public network.
An SSL certificate is mainly used to protect server-client communication. Once data is encrypted by the SSL certificate, it cannot be accessed via a private key but by the server.

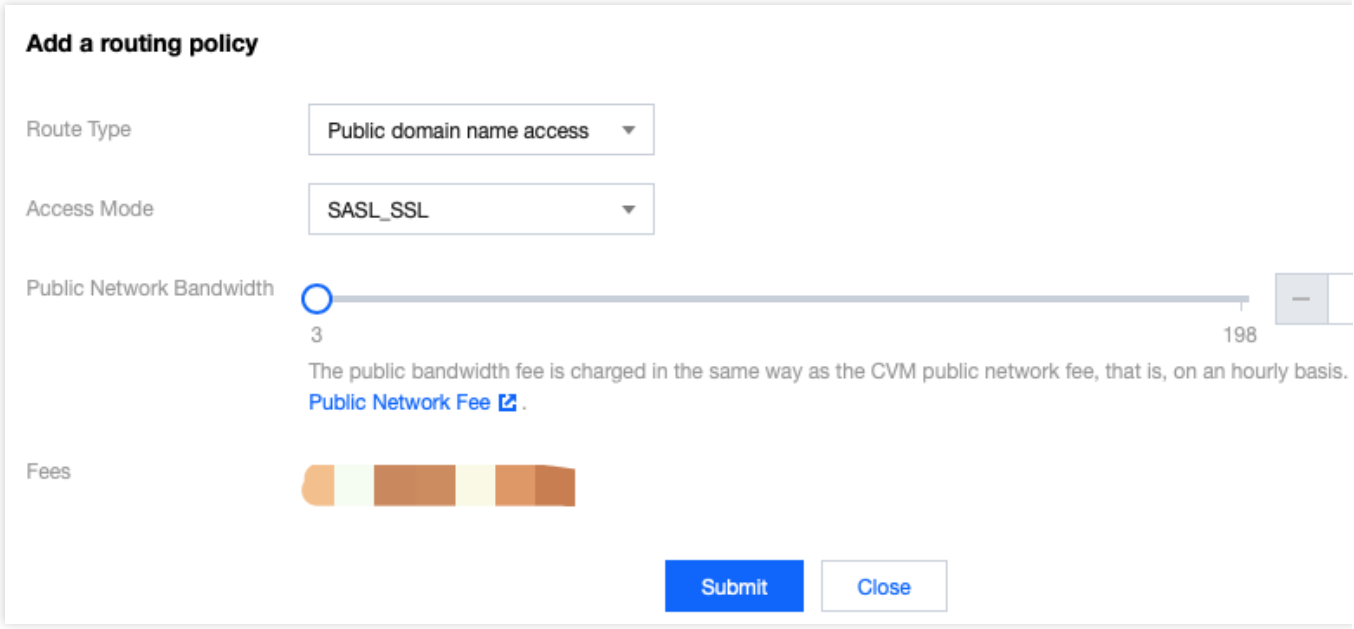
Prerequisites

You have installed JDK 1.8 or later as instructed in Java Downloads.
You have installed Maven 2.5 or later as instructed in Downloading Apache Maven 3.8.6.
You have configured an ACL policy as instructed in Configuring ACL Policy.
You have downloaded the demo from GitHub.
You have downloaded the SASL_SSL certificate here.

Directions

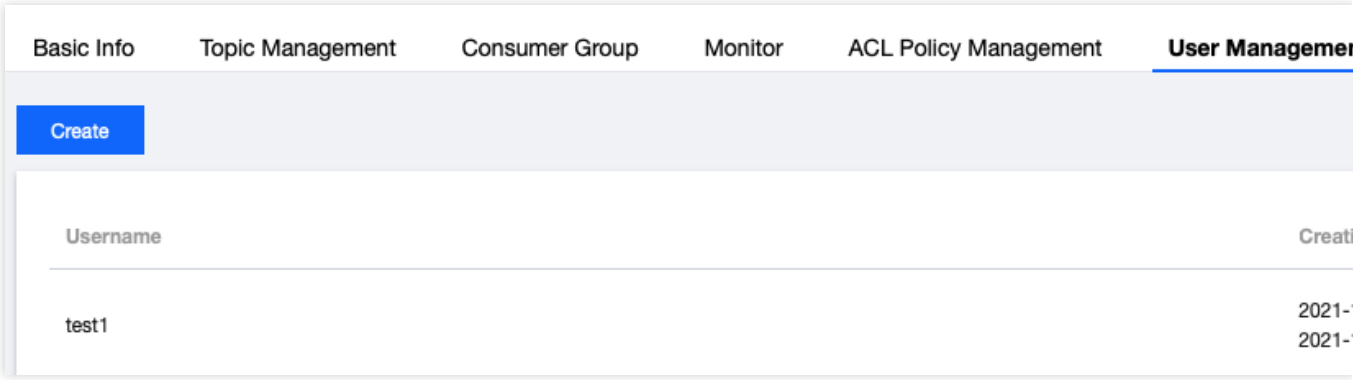
Step 1. Create resources in the console

1. Create an access point.
1.1 On the **Instance List** page in the CKafka console, click the target instance ID to enter the instance details page.
1.2 In **Basic Info** > **Access Mode**, click **Add a routing policy**. In the pop-up window, select `Route Type: Public domain name access` and `Access Mode: SASL_SSL`.



2. Create a role.

On the **User Management** tab, create a role and set the password.



3. Create a topic.

Create a topic on the **Topic Management** tab as instructed in Creating Topic.

Step 2. Add the configuration file

1. Add the following dependencies to the `pom.xml` file:

```
<dependencies>
   <dependency>
      <groupId>org.apache.kafka</groupId>
      <artifactId>kafka-clients</artifactId>
      <version>2.1.0</version>
   </dependency>
   <dependency>
```

```
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-api</artifactId>
        <version>1.7.5</version>
    </dependency>
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-simple</artifactId>
        <version>1.6.4</version>
    </dependency>
</dependencies>
```

2. Create a JAAS configuration file named `ckafka_client_jaas.conf` and modify it with the user created on the **User Management** tab page.

```
KafkaClient {
org.apache.kafka.common.security.plain.PlainLoginModule required
username="yourinstance#yourusername"
password="yourpassword";
};
```

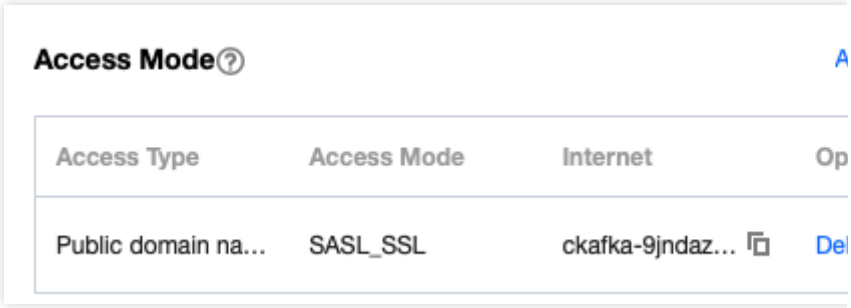
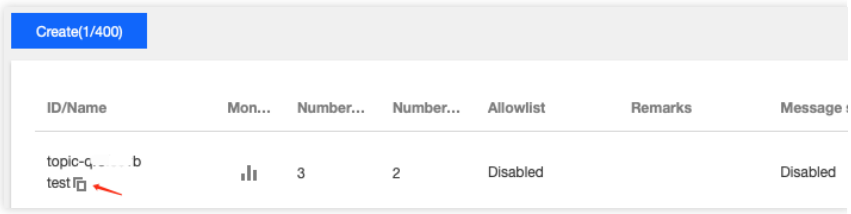
Note:

Set `username` to a value in the format of `instance ID` + `#` + `configured username`, and `password` to a configured password.

3. Create a CKafka configuration file named `kafka.properties`.

```
## Configure the accessed network by copying the information in the **Network** col
bootstrap.servers=ckafka-xxxxxxx
## Configure the topic by copying the information on the **Topic Management** page
topic=XXX
## Configure the consumer group as needed
group.id=XXX
## SASL configuration
java.security.auth.login.config.plain=/xxxx/ckafka_client_jaas.conf
## SSL certificate configuration, which takes effect when the access mode is specif
ssl.truststore.location=/xxxx/client.truststore.jks
ssl.truststore.password=5fi6R!M
ssl.endpoint.identification.algorithm=
```

Parameter	Description
bootstrap.servers	Accessed network, which can be copied in the **Network** column in the Acces page in the console.

	
topic	Topic name, which can be copied from the **Topic Management** page in the 
group.id	You can customize it. After the demo runs successfully, you can see the cons
java.security.auth.login.config.plain	Enter the path of the JAAS configuration file ckafka_client_jaas.conf.
client.truststore.jks	The required certificate path when SASL_SSL is used for access.

4. Create a configuration file loading program named `CKafkaConfigurer.java`.

```
public class CKafkaConfigurer {
    private static Properties properties;
    public static void configureSaslPlain() {
        // If you have used the `-D` parameter or another method to set the path, d
        if (null == System.getProperty("java.security.auth.login.config")) {
            // Replace `XXX` with your own path
            System.setProperty("java.security.auth.login.config",
                    getCKafkaProperties().getProperty("java.security.auth.login.con
        }
    }
    public synchronized static Properties getCKafkaProperties() {
        if (null != properties) {
            return properties;
        }
        // Get the content of the configuration file `kafka.properties`.
        Properties kafkaProperties = new Properties();
        try {
            kafkaProperties.load(CKafkaProducerDemo.class.getClassLoader().getResou
```

```
        } catch (Exception e) {
            System.out.println("getCKafkaProperties error");
        }
        properties = kafkaProperties;
        return kafkaProperties;
    }
}
```

Step 3. Send messages

1. Create a message sending program named `KafkaSaslProducerDemo.java` .

```
public class KafkaSaslProducerDemo {
public static void main(String[] args) {
    // Set the path of the JAAS configuration file.
    CKafkaConfigurer.configureSaslPlain();
    // Load `kafka.properties`.
    Properties kafkaProperties = CKafkaConfigurer.getCKafkaProperties();
    Properties props = new Properties();
    // Set the access point. Get the access point of the corresponding topic in t
    props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
            kafkaProperties.getProperty("bootstrap.servers"));
    //
    // Use SASL_SSL for public network access.
    //
    // Set the access protocol.
    props.put(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG, "SASL_SSL");
    // Use Plain mode for SASL.
    props.put(SaslConfigs.SASL_MECHANISM, "PLAIN");
    // Set SSL encryption.
    props.put(SslConfigs.SSL_TRUSTSTORE_LOCATION_CONFIG, kafkaProperties.getPrope
    props.put(SslConfigs.SSL_TRUSTSTORE_PASSWORD_CONFIG, kafkaProperties.getPrope
    props.put(SslConfigs.SSL_ENDPOINT_IDENTIFICATION_ALGORITHM_CONFIG,kafkaProper
    // Set the method for serializing Kafka messages.
    props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
            "org.apache.kafka.common.serialization.StringSerializer");
    props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
            "org.apache.kafka.common.serialization.StringSerializer");
    // Set the maximum request wait time.
    props.put(ProducerConfig.MAX_BLOCK_MS_CONFIG, 30 * 1000);
    // Set the number of retries for the client
    props.put(ProducerConfig.RETRIES_CONFIG, 5);
    // Set the internal retry interval for the client.
    props.put(ProducerConfig.RECONNECT_BACKOFF_MS_CONFIG, 3000);
    // If `ack` is 0, the producer will not wait for acknowledgment from the brok
    // If `ack` is 1, the broker leader will directly return `ack` without waitin
```

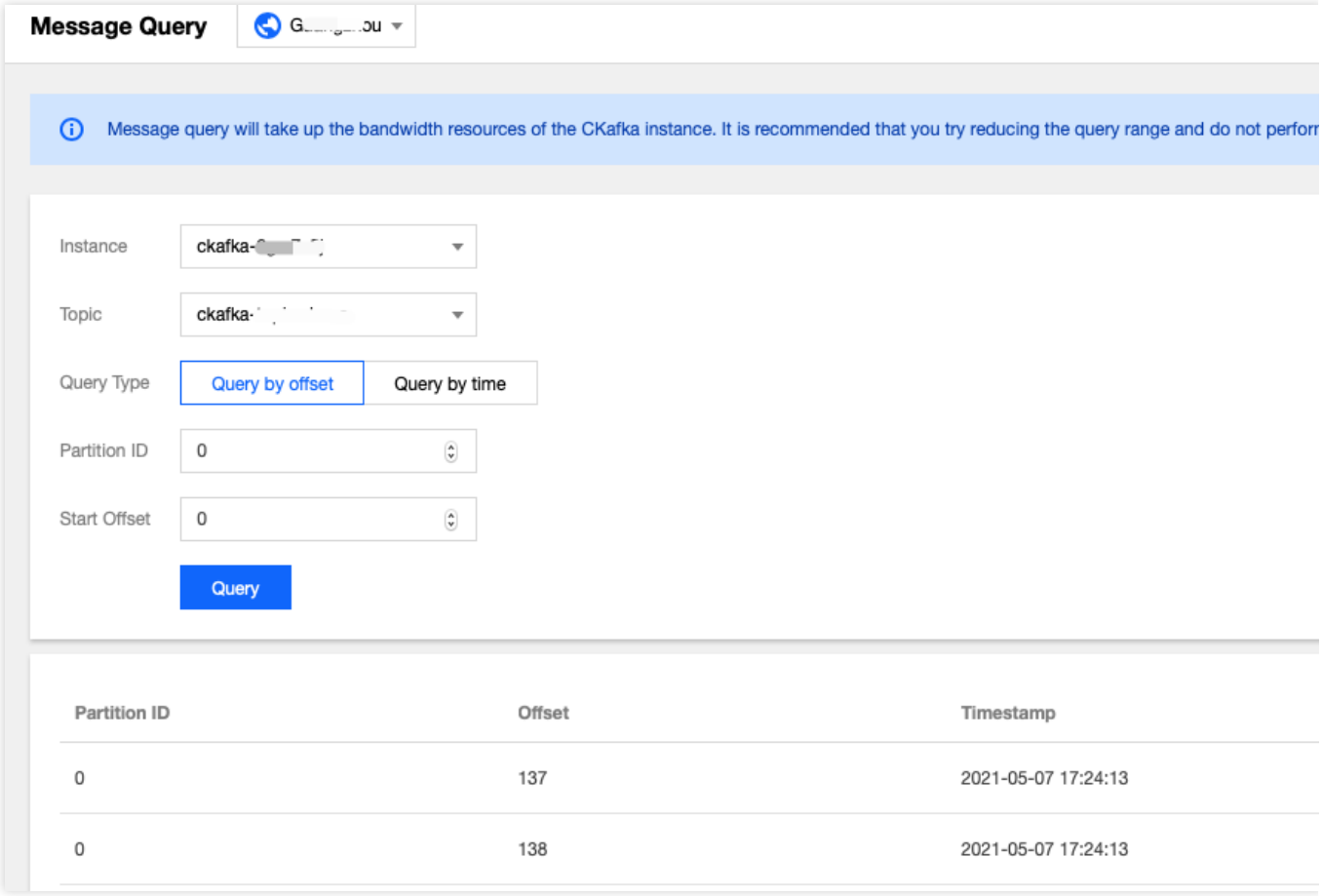
```
        // If `ack` is `all`, the broker leader will return `ack` only after receivin
        props.put(ProducerConfig.ACKS_CONFIG, "all");
        // Construct a producer object. Note: A producer object is thread-safe, and g
        KafkaProducer<String, String> producer = new KafkaProducer<>(props);
        // Construct a CKafka message.
        String topic = kafkaProperties.getProperty("topic"); // Topic of the message.
        String value = "this is ckafka msg value"; // Message content
        try {
            // Batch getting future objects can speed up the process. Note that the ba
            List<Future<RecordMetadata>> futures = new ArrayList<>(128);
            for (int i = 0; i < 100; i++) {
                // Send the message and get a future object.
                ProducerRecord<String, String> kafkaMessage = new ProducerRecord<>(topi
                        value + ": " + i);
                Future<RecordMetadata> metadataFuture = producer.send(kafkaMessage);
                futures.add(metadataFuture);
            }
            producer.flush();
            for (Future<RecordMetadata> future : futures) {
                // Sync the future object obtained.
                RecordMetadata recordMetadata = future.get();
                System.out.println("Produce ok:" + recordMetadata.toString());
            }
        } catch (Exception e) {
            // If the sending still fails after client internal retries, the system ne
            System.out.println("error occurred");
        }
    }
}
```

2. Compile and run `KafkaSaslProducerDemo.java` to send the message.

3. View the execution result (output).

```
Produce ok:ckafka-topic-demo-0@198
Produce ok:ckafka-topic-demo-0@199
```

4. On the **Topic Management** tab on the instance details page in the CKafka console, select the target topic and click **More** > **Message Query** to view the message just sent.



Step 4. Consume the message

1. Create a program named `KafkaSaslConsumerDemo.java` for a consumer to subscribe to messages.

```
public class KafkaSaslConsumerDemo {
    public static void main(String[] args) {
        // Set the path of the JAAS configuration file.
        CKafkaConfigurer.configureSaslPlain();
        // Load `kafka.properties`.
        Properties kafkaProperties = CKafkaConfigurer.getCKafkaProperties();
        Properties props = new Properties();
        // Set the access point. Get the access point of the corresponding topic in t
        props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
                kafkaProperties.getProperty("bootstrap.servers"));
        //
        // Use SASL_SSL for public network access.
        //
        // Set the access protocol.
        props.put(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG, "SASL_SSL");
        // Use Plain mode for SASL.
        props.put(SaslConfigs.SASL_MECHANISM, "PLAIN");
        // Set SSL encryption.
```

```
        props.put(SslConfigs.SSL_TRUSTSTORE_LOCATION_CONFIG, kafkaProperties.getPrope
        props.put(SslConfigs.SSL_TRUSTSTORE_PASSWORD_CONFIG, kafkaProperties.getPrope
        props.put(SslConfigs.SSL_ENDPOINT_IDENTIFICATION_ALGORITHM_CONFIG,kafkaProper
        // Set the consumer timeout period.
        // If the consumer does not return a heartbeat message within the interval, t
        props.put(ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG, 30000);
        // Set the maximum time interval between two polls.
        // Before v0.10.1.0, these two concepts were mixed and both represented by `s
        props.put(ConsumerConfig.MAX_POLL_INTERVAL_MS_CONFIG, 30000);
        // Set the maximum number of messages that can be polled at a time.
        // Do not set this parameter to an excessively large value. If polled message
        props.put(ConsumerConfig.MAX_POLL_RECORDS_CONFIG, 30);
        // Set the method for deserializing messages.
        props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
                "org.apache.kafka.common.serialization.StringDeserializer");
        props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
                "org.apache.kafka.common.serialization.StringDeserializer");
        // Set the consumer group of the current consumer instance after you apply fo
        // The instances in the same consumer group consume messages in load balancin
        props.put(ConsumerConfig.GROUP_ID_CONFIG, kafkaProperties.getProperty("group.
        // Construct a consumer object. This generates a consumer instance.
        KafkaConsumer<String, String> consumer = new KafkaConsumer<String, String>(pr
        // Set one or more topics to which the consumer group subscribes.
        // We recommend you configure consumer instances with the same `GROUP_ID_CONF
        List<String> subscribedTopics = new ArrayList<String>();
        // If you want to subscribe to multiple topics, add the topics here.
        // You must create the topics in the console in advance.
        String topicStr = kafkaProperties.getProperty("topic");
        String[] topics = topicStr.split(",");
        for (String topic : topics) {
           subscribedTopics.add(topic.trim());
        }
        consumer.subscribe(subscribedTopics);
        // Consume messages in loop.
        while (true) {
           try {
              ConsumerRecords<String, String> records = consumer.poll(1000);
              // All messages must be consumed before the next poll, and the total du
              for (ConsumerRecord<String, String> record : records) {
                 System.out.println(
                         String.format("Consume partition:%d offset:%d", record.parti
                                 record.offset()));
              }
           } catch (Exception e) {
              System.out.println("consumer error!");
           }
        }
```

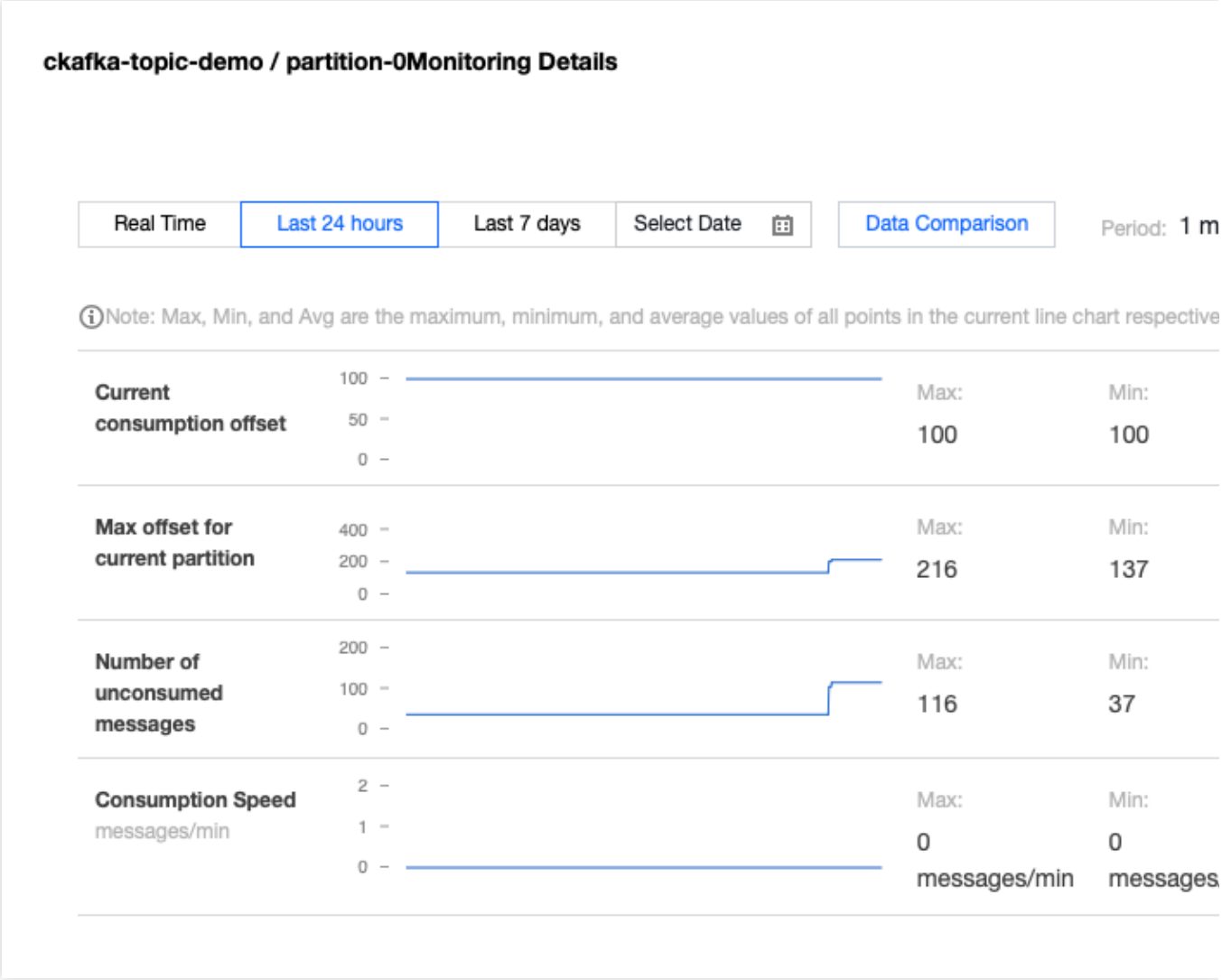
```
    }
}
```

2. Compile and run `KafkaSaslConsumerDemo.java` to consume the message.

3. View the execution result.

```
Consume partition:0 offset:298
Consume partition:0 offset:299
```

4. On the **Consumer Group** tab in the CKafka console, select the corresponding consumer group, enter the topic name, and click **View Details** to view the consumption details.



SDK for Python VPC Access

Last updated：2024-01-09 15:00:32

Overview

This document describes how to access CKafka to send/receive messages with the SDK for Python in a VPC.

Prerequisites

Install Python
Install pip
Download the demo

Directions

Upload the `pythonkafkademo` in the downloaded demo to the Linux server, log in to the server, and enter the `pythonkafkademo` directory.

Step 1. Add the Python dependency library

Run the following command to install:

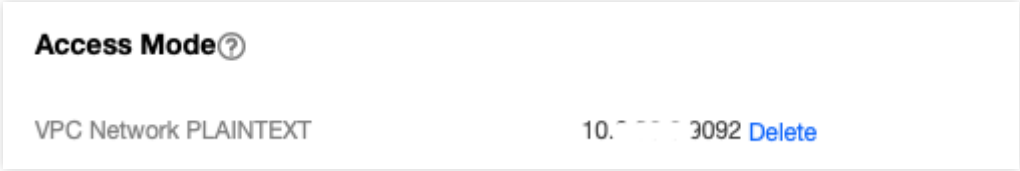
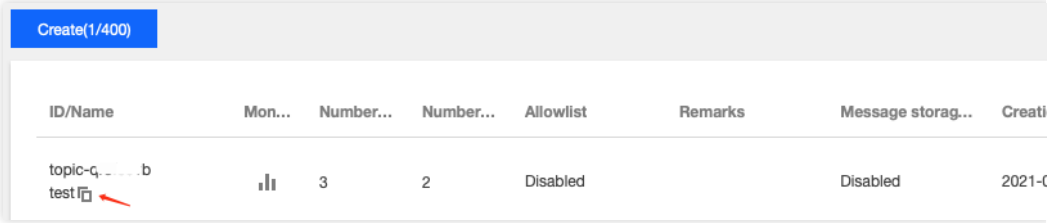
```
pip install kafka-python
```

Step 2. Produce a message

1. Modify the configuration parameters in the message production program `producer.py` .

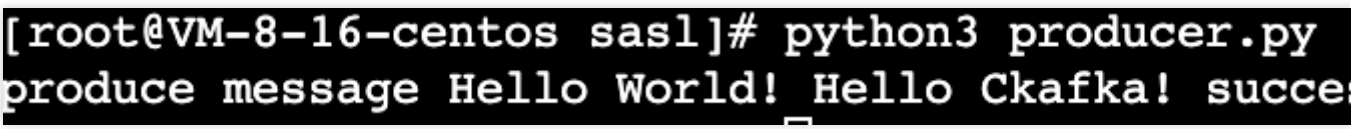
```
#coding:utf8
from kafka import KafkaProducer
import json
producer = KafkaProducer(
   bootstrap_servers = ['$domainName:$port'],
   api_version = (0,10,0)
)
message = "Hello World! Hello Ckafka!"
msg = json.dumps(message).encode()
```

```
producer.send('topic_name',value = msg)
print("produce message " + message + " success.")
producer.close()
```

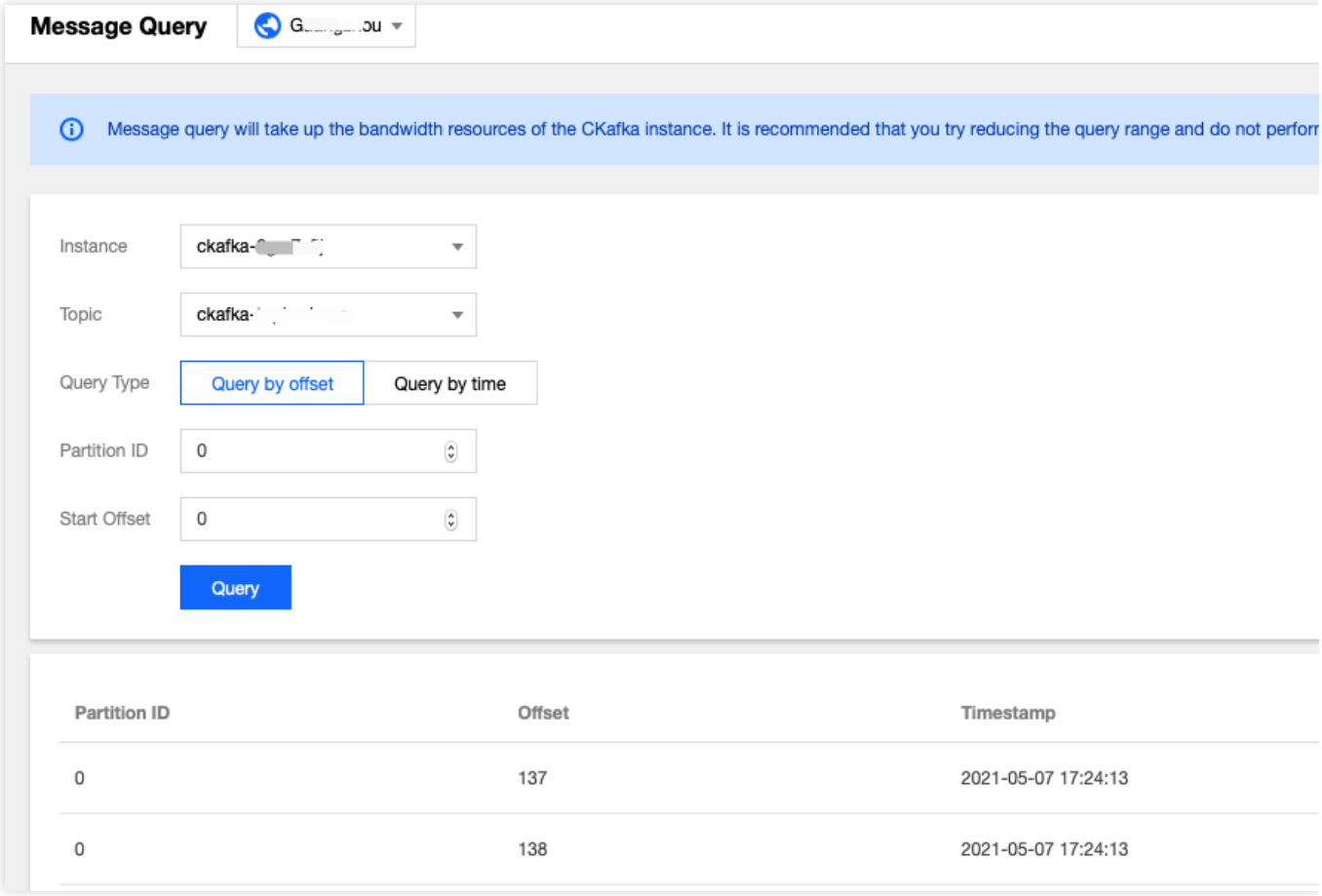
Parameter	Description
bootstrap_servers	Accessed network, which can be copied in the **Network** column in the Access Mode section console. 
topic_name	Topic name, which can be copied from the **Topic Management** page in the console. 

2. Compile and run `producer.py` .

3. View the operation result.



4. On the **Topic Management** tab on the instance details page in the CKafka console, select the target topic and click **More** > **Message Query** to view the message just sent.

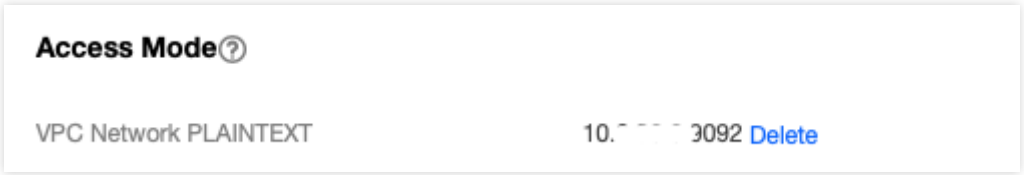
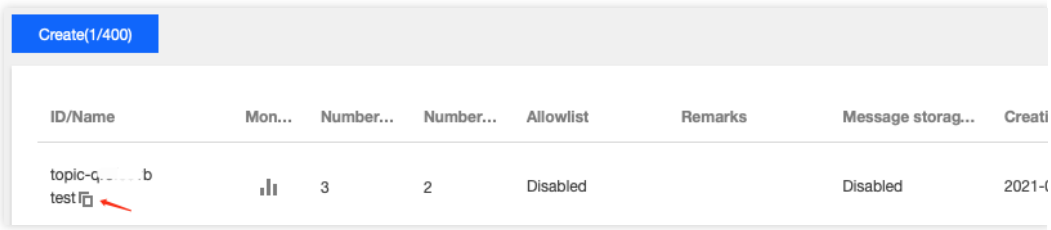


Step 3. Consume the message

1. Modify the configuration parameters in the message consumption program `consumer.py` .

```
#coding:utf8
from kafka import KafkaConsumer

consumer = KafkaConsumer(
    '$topic_name',
    group_id = "$group_id",
    bootstrap_servers = ['$domainName:$port'],
    api_version = (0,10,0)
)

for message in consumer:
    print ("Topic:[%s] Partition:[%d] Offset:[%d] Value:[%s]" % (message.topic, mes
```

Parameter	Description
bootstrap_servers	Accessed network, which can be copied in the **Network** column in the Access Mode section console.

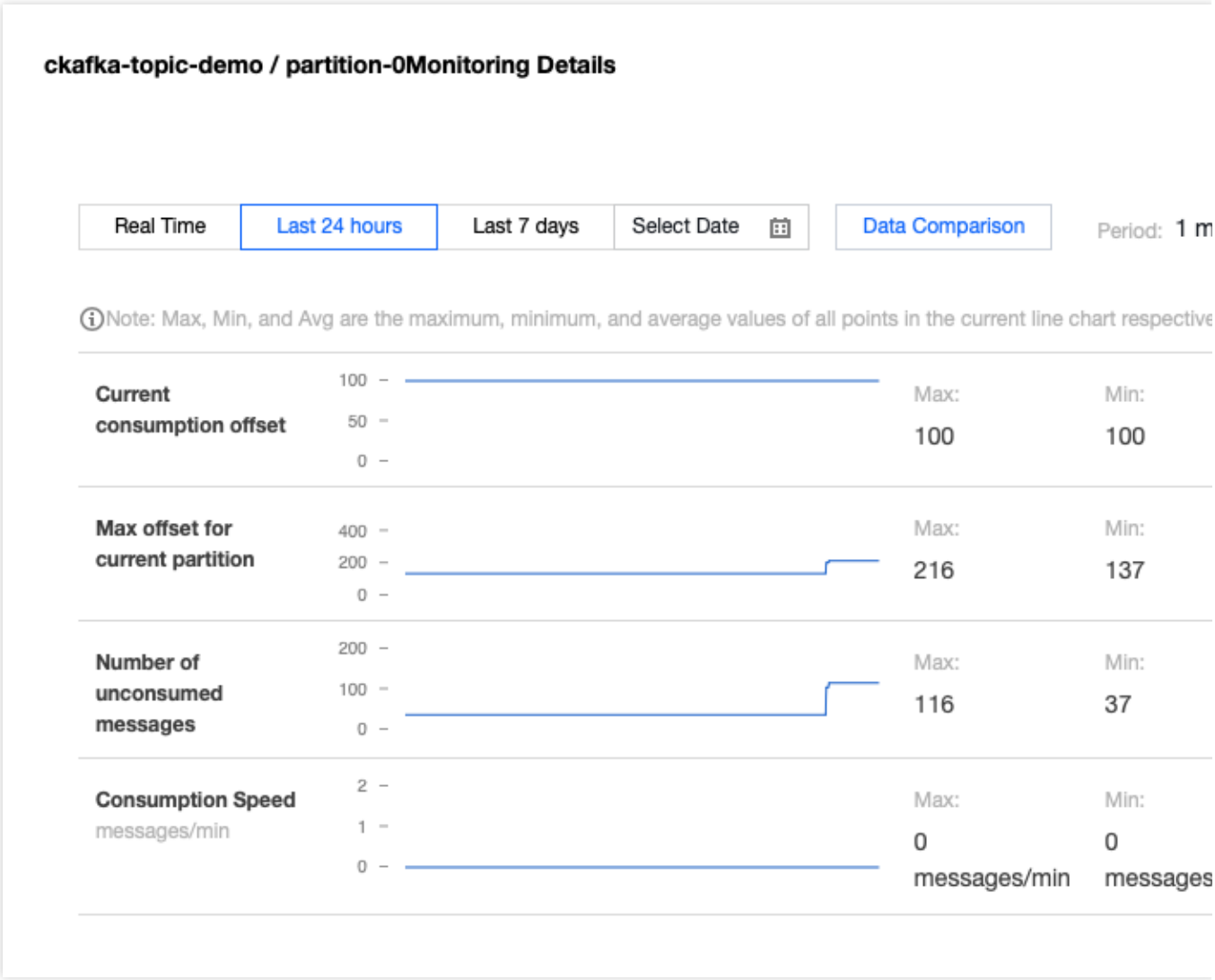
	
group_id	Consumer group ID, which can be customized according to the business needs
topic_name	Topic name, which can be copied from the **Topic Management** page in the console. 

2. Compile and run `consumer.py`.

3. View the operation result.

```
[root@VM-8-16-centos sasl]# python3 consumer.py
Topic:[test] Partition:[0] Offset:[2011] Value:[b'"Hello Wor
```

4. On the **Consumer Group** tab in the CKafka console, select the corresponding consumer group, enter the topic name, and click View Details to view the consumption details.



Public Network Access Through SASL_PLAINTEXT

Last updated：2024-01-09 15:00:32

Overview

This document describes how to access CKafka to receive/send messages with the SDK for Python through SASL_PLAINTEXT over public network.

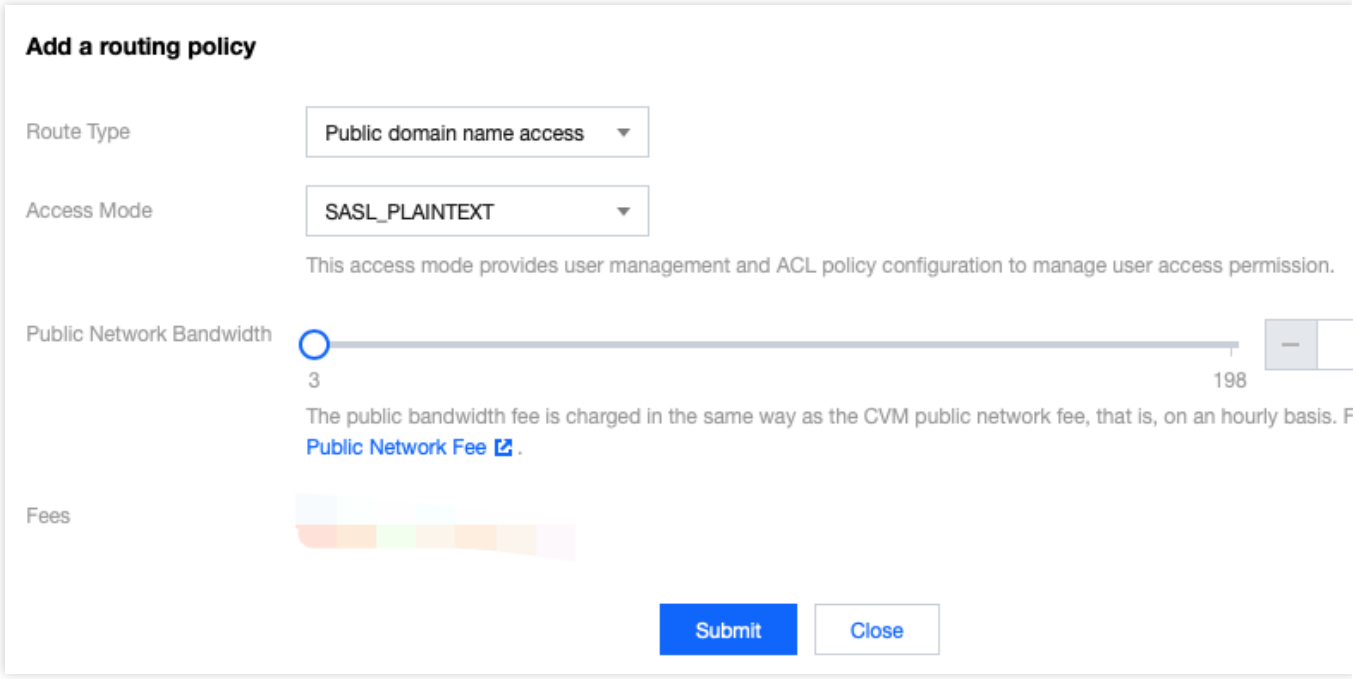
Prerequisites

You have installed Python.
You have installed pip.
You have configured an ACL policy.
You have downloaded the demo.

Directions

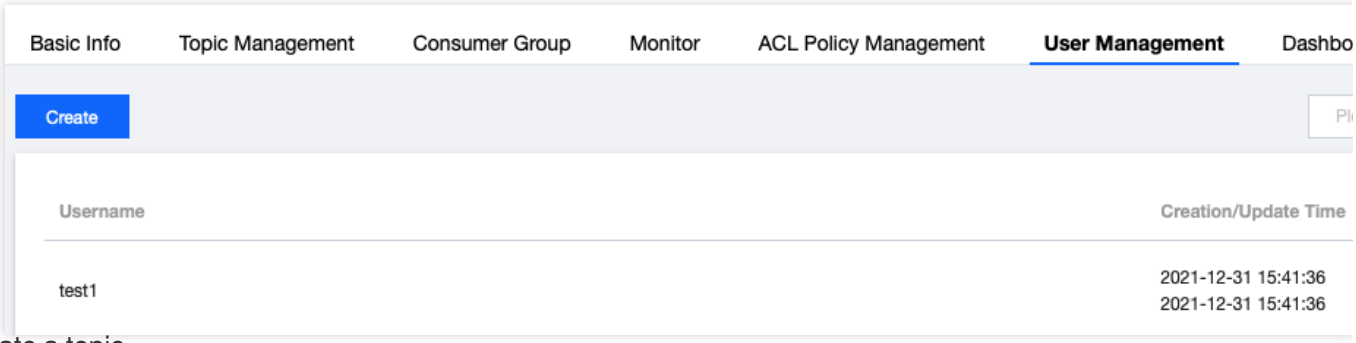
Step 1. Make preparations

1. Create an access point.
1.1 On the Instance List page in the CKafka console, click the target instance ID to enter the instance details page.
1.2 In **Basic Info** > **Access Mode**, click **Add a routing policy**. In the pop-up window, select `Route Type: Public domain name access` , `Access Mode: SASL_PLAINTEXT` .



2. Create a role.

On the **User Management** tab page, create a role and set the password.



3. Create a topic.

Create a topic on the **Topic Management** tab page as instructed in Topic Management.

4. Add the Python dependent library.

Run the following command to install the dependent library:

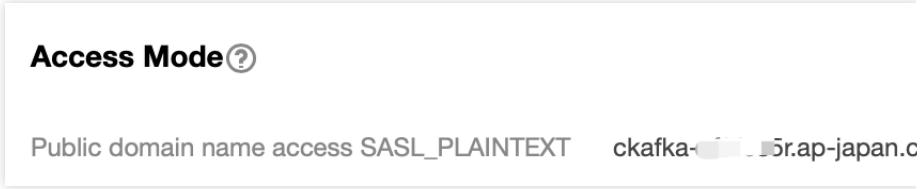
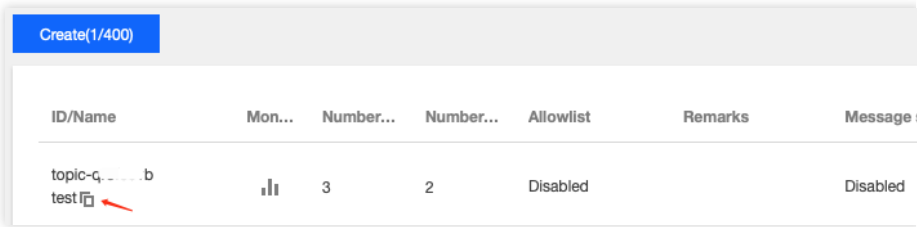
```
pip install kafka-python
```

Step 2. Produce messages

1. Modify the configuration parameters in the message production program `producer.py` .

```
producer = KafkaProducer(
 bootstrap_servers = ['xx.xx.xx.xx:port'],
 api_version = (1, 1),

 #
```

```
 # Public network access through SASL_PLAINTEXT
 #
 security_protocol = "SASL_PLAINTEXT",
 sasl_mechanism = "PLAIN",
 sasl_plain_username = "instanceId#username",
 sasl_plain_password = "password",
)

message = "Hello World! Hello Ckafka!"
msg = json.dumps(message, ensure_ascii=False).encode()
producer.send('topic_name', value = msg)
print("produce message " + message + " success.")
producer.close()
```

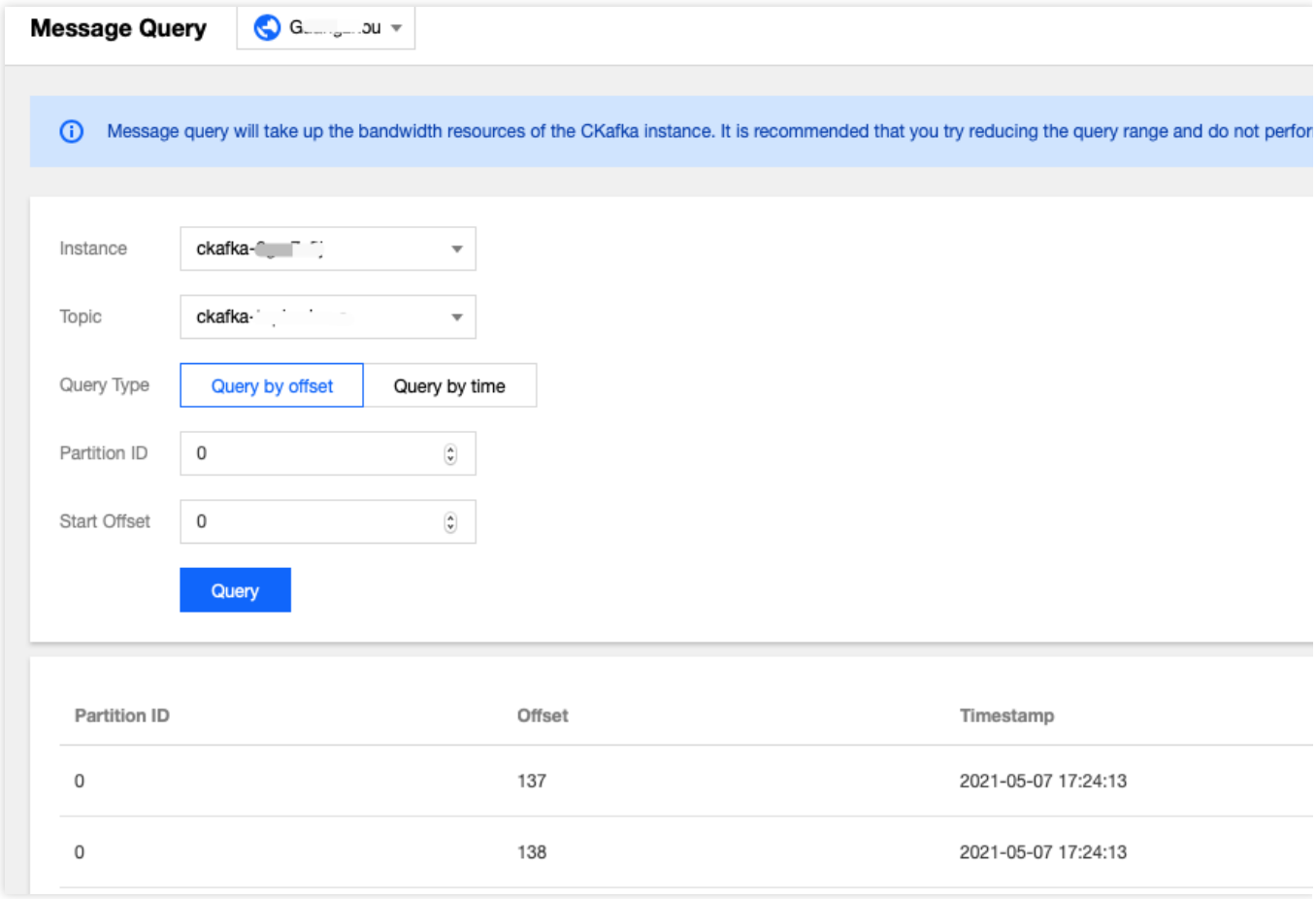
Parameter	Description
`bootstrap_servers`	Accessed network, which can be copied from the **Network** column in the **Access** details page in the console. 
`sasl_plain_username`	Username in the format of `instance ID` + `#` + `username`. The instance instance details page in the CKafka console, and the username is set when the use
`sasl_plain_password`	User password, which is set when the user is created in **User Management**on the
`topic_name`	Topic name, which can be copied in **Topic Management** in the console. 

2. Compile and run `producer.py`.

3. View the execution result.

```
[root@VM-8-16-centos sasl]# python3 producer.py
produce message Hello World! Hello Ckafka! success.
```

4. Go to the **Topic Management** tab on the instance details page in the CKafka console, select the target topic, and click **More** > **Message Query** to view the message just sent.

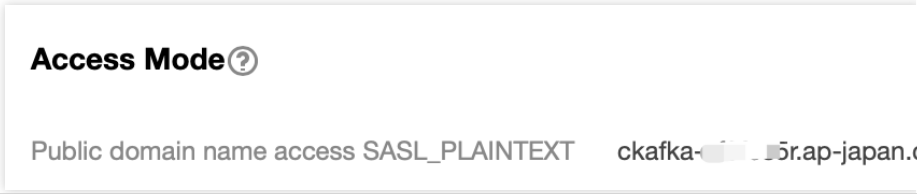
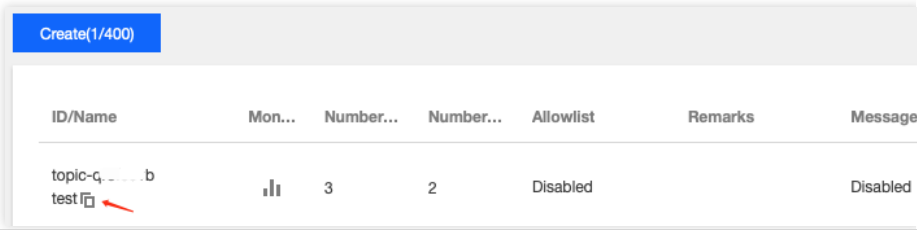


Step 3. Consume messages

1. Modify the configuration parameters in the message consumption program `consumer.py` .

```
consumer = KafkaConsumer(
 'topic_name',
 group_id = "group_id",
 bootstrap_servers = ['xx.xx.xx.xx:port'],
 api_version = (1,1),

 #
 # Public network access through SASL_PLAINTEXT
 #
```

```
 security_protocol = "SASL_PLAINTEXT",
 sasl_mechanism = 'PLAIN',
 sasl_plain_username = "instanceId#username",
 sasl_plain_password = "password",
)

for message in consumer:
 print ("Topic:[%s] Partition:[%d] Offset:[%d] Value:[%s]" %
  (message.topic, message.partition, message.offset, message.value))
```

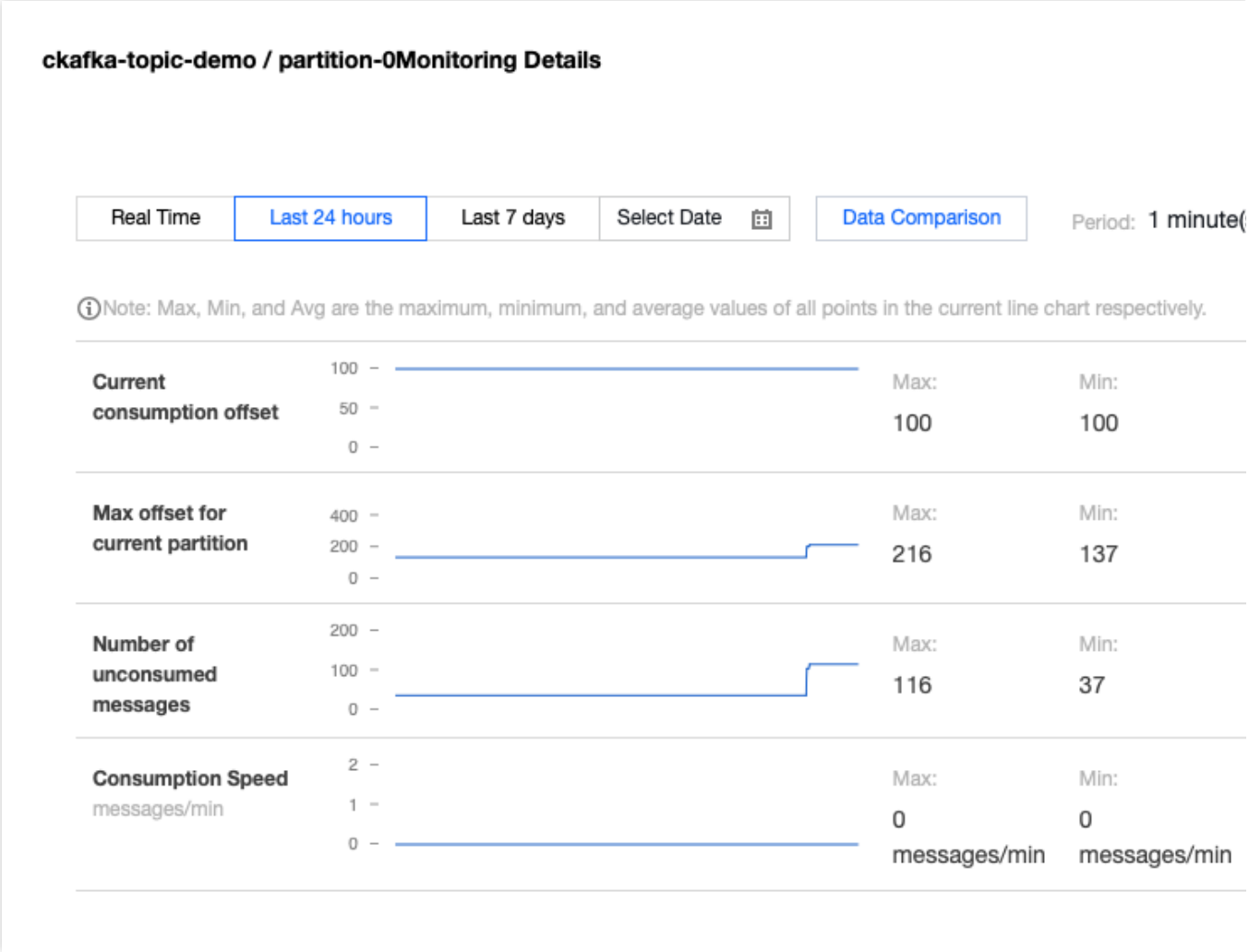
Parameter	Description
`bootstrap_servers`	Accessed network, which can be copied from the **Network** column in the **Access** details page in the console. 
`group_id`	Consumer group ID, which can be customized based on business requirements.
`sasl_plain_username`	Username in the format of `instance ID` + `#` + `username`. The instance instance details page in the CKafka console, and the username is set when the user is created in **User Management**.
`sasl_plain_password`	User password, which is set when the user is created in **User Management**on the
`topic_name`	Topic name, which can be copied in **Topic Management** in the console. 

2. Compile and run `consumer.py`.

3. View the execution result.

```
[root@VM-8-16-centos sasl]# python3 consumer.py
Topic:[test] Partition:[0] Offset:[2011] Value:[b'"Hello World! Hello Ckafka!"']
```

4. On the **Consumer Group** tab page in the CKafka console, select the corresponding consumer group, enter the topic name, and click **View Details** to view the consumption details.



Public Network Access Through SASL_SSL

Last updated：2024-01-09 15:00:32

Overview

This document describes how to access CKafka to send/receive messages with the SDK for Python through SASL_SSL over public network.

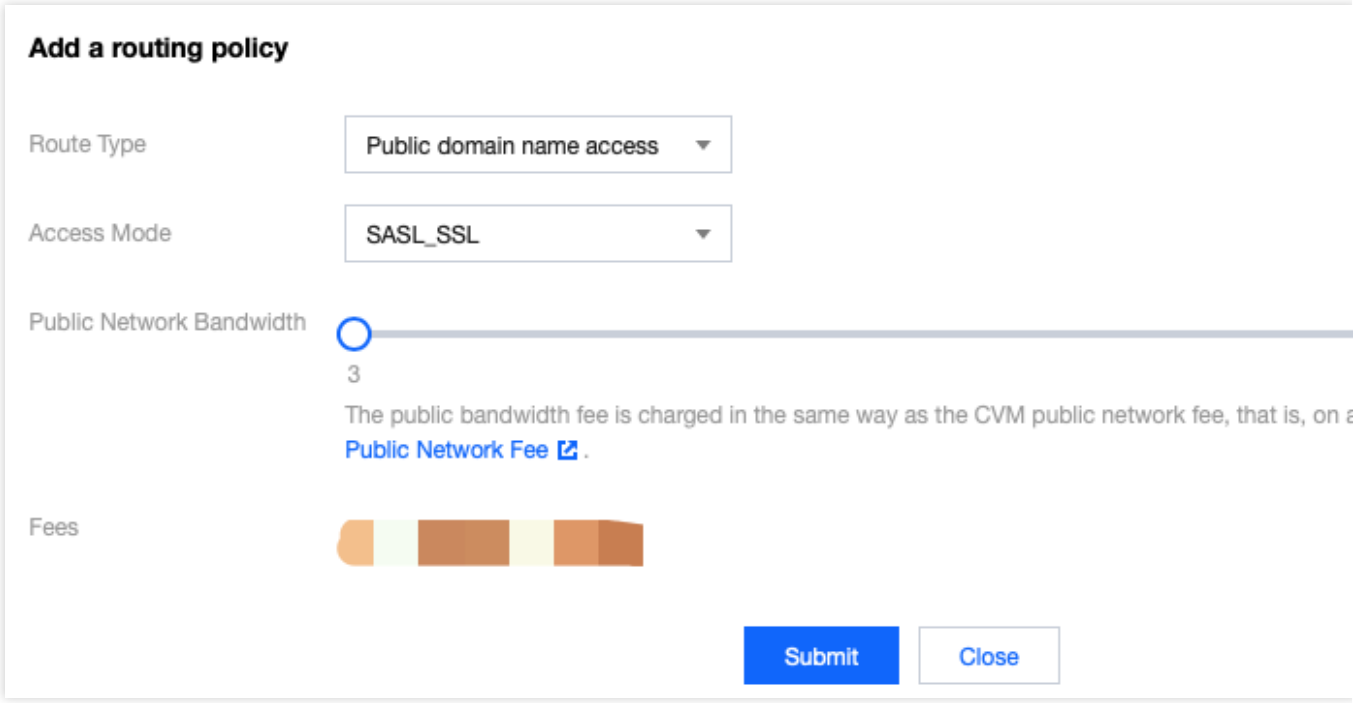
Prerequisites

Install Python
Install pip
Configure an ACL policy
Download demo
Download the SASL_SSL certificate

Directions

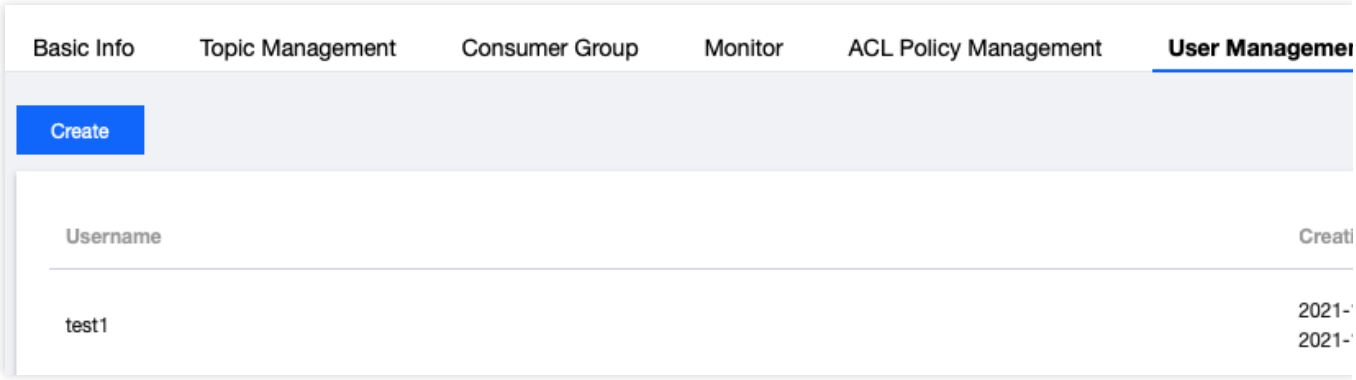
Step 1. Make preparations

1. Create an access point.
1.1 On the **Instance List** page in the CKafka console, click the target instance ID to enter the instance details page.
1.2 In **Basic Info** > **Access Mode**, click **Add a routing policy**. In the pop-up window, select `Route Type: Public domain name access` , `Access Mode: SASL_SSL` .



2. Create a role.

On the **User Management** tab page, create a role and set the password.



3. Create a topic.

Create a topic on the **Topic Management** tab page as instructed in Creating a topic.

4. Add the Python dependent library.

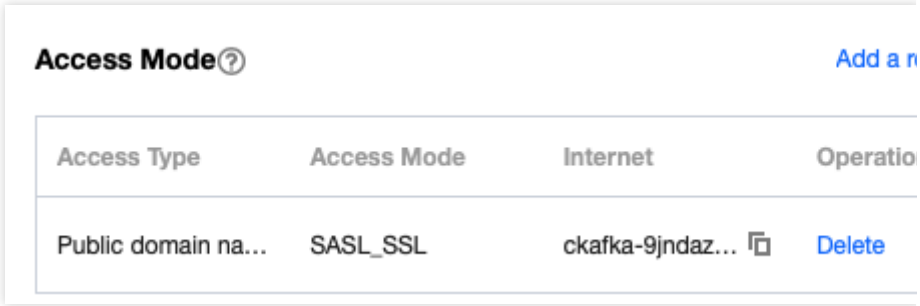
Run the following command to install the dependent library:

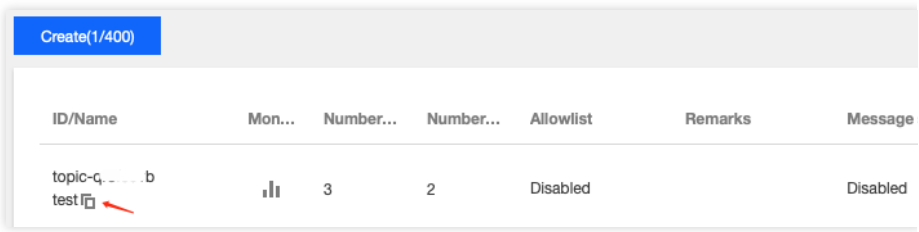
```
pip install kafka-python
```

Step 2. Produce messages

1. Modify the configuration parameters in the message production program `producer.py` .

```
producer = KafkaProducer(
 bootstrap_servers = ['xx.xx.xx.xx:port'],
 api_version = (1, 1),

 #
 # Public network access through SASL_SSL
 #
 security_protocol = "SASL_SSL",
 sasl_mechanism = "PLAIN",
 sasl_plain_username = "instanceId#username",
 sasl_plain_password = "password",
 ssl_cafile = "CARoot.pem",
 ssl_check_hostname = False,
)

message = "Hello World! Hello Ckafka!"
msg = json.dumps(message).encode()
producer.send('topic_name', value = msg)
print("produce message " + message + " success.")
producer.close()
```

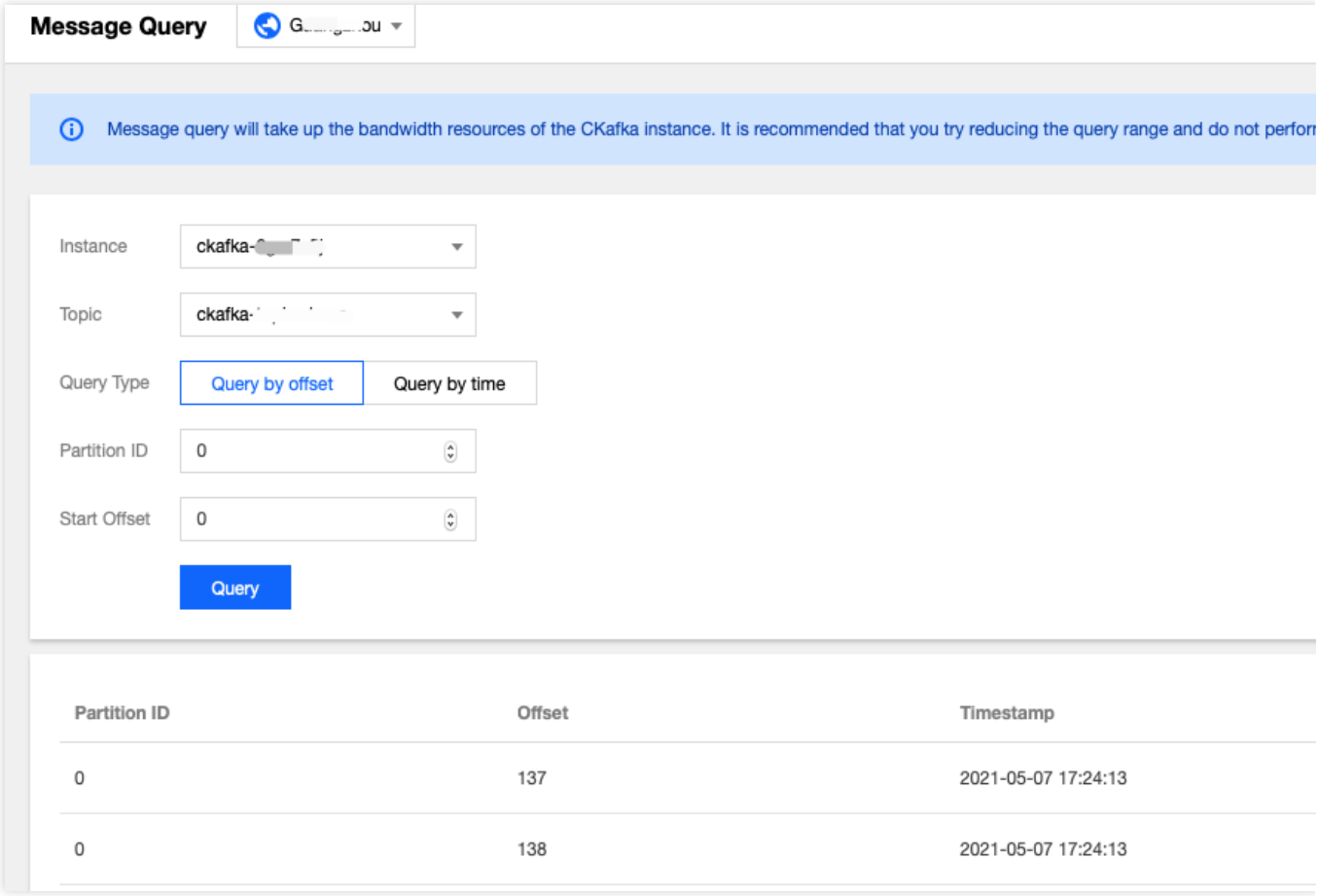
Parameter	Description
`bootstrap_servers`	Accessed network, which can be copied from the **Network** column in the **Access** details page in the console. 
`sasl_plain_username`	Username in the format of `instance ID` + `#` + `username`. The instance instance details** page in the CKafka console, and the username is set when the u
`sasl_plain_password`	User password, which is set when the user is created in **User Management** on th
`topic_name`	Topic name, which can be copied from the **Topic Management** page in the consc

	
`CARoot.pem`	Certificate path that is required when the access mode is `SASL_SSL` .

2. Compile and run `producer.py` .

3. View the execution result.

4. On the **Topic Management** tab page on the instance details page in the CKafka console, select the target topic, and click **More** > **Message Query** to view the message just sent.

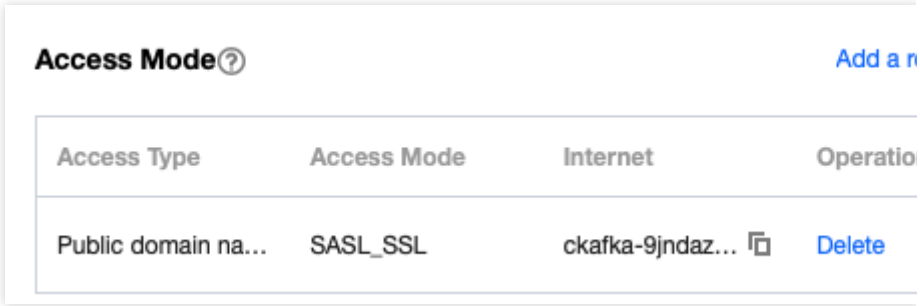
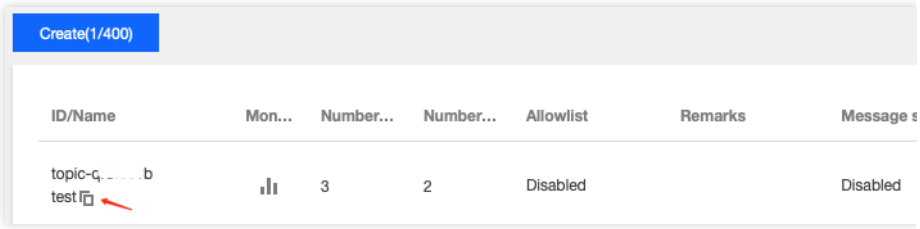


Step 3. Consume messages

1. Modify the configuration parameters in the message consumption program `consumer.py` .

```
consumer = KafkaConsumer(
 'topic_name',
 group_id = "group_id",
 bootstrap_servers = ['xx.xx.xx.xx:port'],
 api_version = (1,1),

 #
 # Public network access through SASL_SSL
 #
 security_protocol = "SASL_SSL",
 sasl_mechanism = 'PLAIN',
 sasl_plain_username = "instanceId#username",
 sasl_plain_password = "password",
 ssl_cafile = "CARoot.pem",
 ssl_check_hostname = False,

)
```

```
for message in consumer:
 print ("Topic:[%s] Partition:[%d] Offset:[%d] Value:[%s]" %
 (message.topic, message.partition, message.offset, message.value))
```

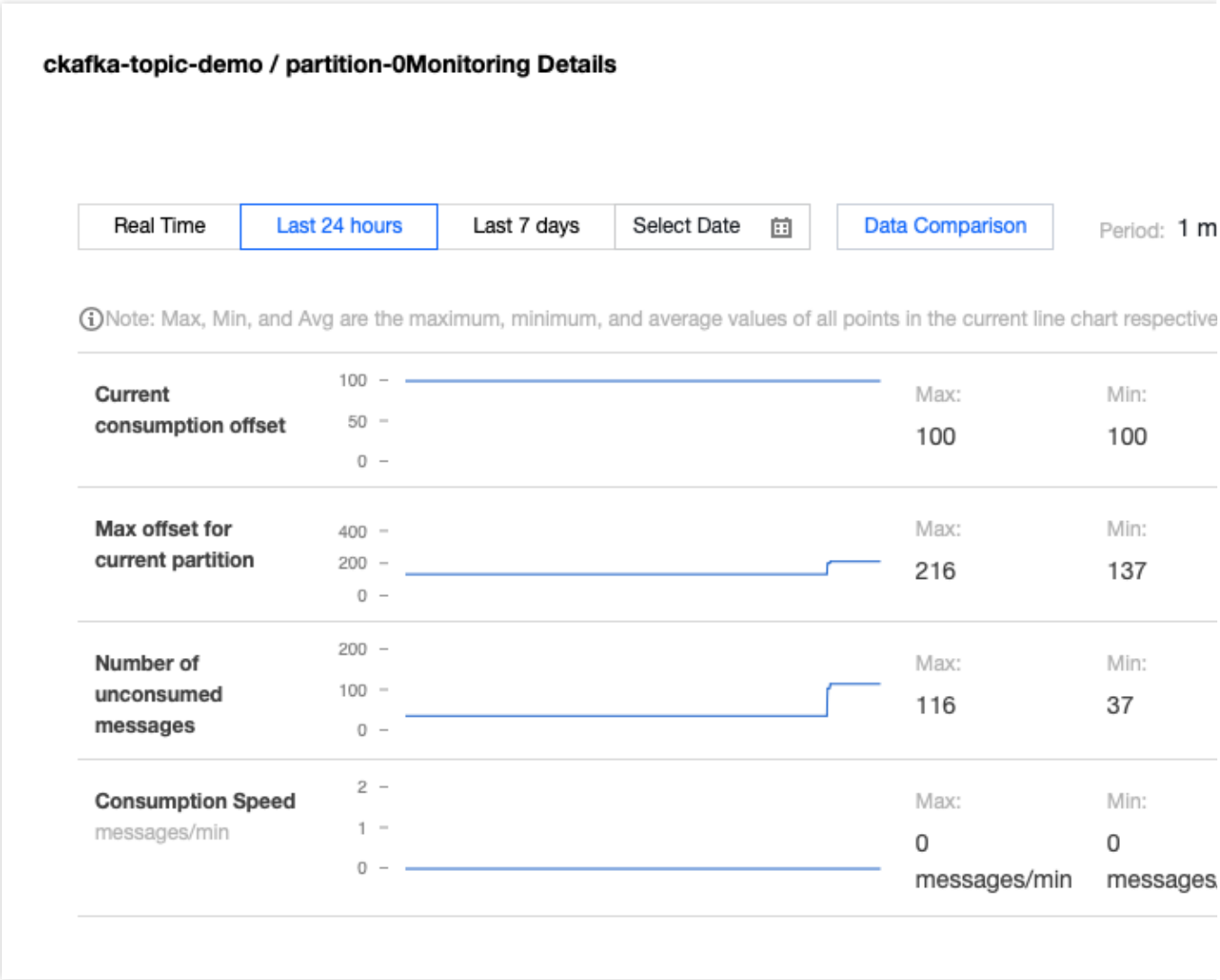
Parameter	Description
`bootstrap_servers`	Accessed network, which can be copied from the **Network** column in the **Access** details page in the console. 
`group_id`	Consumer group ID, which can be customized based on business requirements.
`sasl_plain_username`	Username in the format of `instance ID` + `#` + `username` . The instance instance details** page in the CKafka console, and the username is set when the u
`sasl_plain_password`	User password, which is set when the user is created in **User Management** on th
`topic_name`	Topic name, which can be copied from the **Topic Management** page in the consc 
`CARoot.pem`	Certificate path that is required when the access mode is `SASL_SSL` .

2. Compile and run `consumer.py` .

3. View the execution result.

```
[root@VM-8-16-centos sasl]# python3 consumer.py
Topic:[test] Partition:[0] Offset:[2011] Value:[b'"Hello Wor
```

4. On the **Consumer Group** tab page in the CKafka console, select the corresponding consumer group name, enter the topic name, and click **View Details** to view the consumption details.



SDK for Go VPC Access

Last updated：2022-05-20 15:51:45

Overview

This document describes how to access CKafka to send/receive messages with the SDK for Go in a VPC.

Prerequisites

- You have installed Go
- You have downloaded the demo

Directions

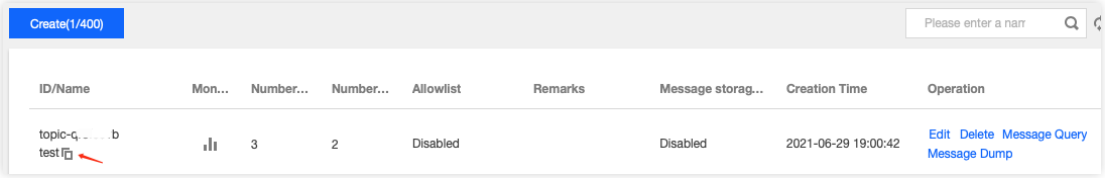
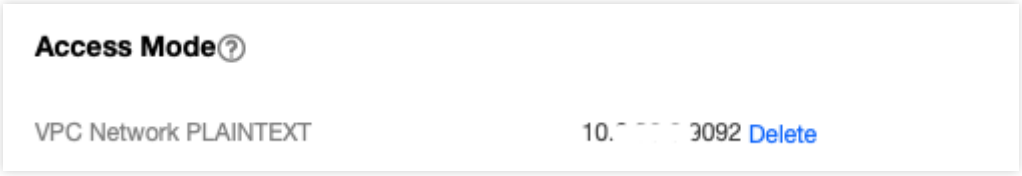
Step 1. Prepare configurations

1. Upload the `gokafkademo` in the downloaded demo to the Linux server.
2. Log in to the Linux server, enter the `gokafkademo` directory, and run the following command to add the dependency library.

```
go get -v gopkg.in/confluentinc/confluent-kafka-go.v1/kafka
```

3. Modify the configuration file `kafka.json` .

```
{
"topic": [
"test"
],
"bootstrapServers": [
"xx.xx.xx.xx:xxxx"
],
"consumerGroupId": "yourConsumerId"
}
```

Parameter	Description
topic	Topic name, which can be copied in **Topic Management** on the instance details page in the console. 
bootstrapServers	Accessed network, which can be copied from the **Network** column in the **Access Mode** section in **Basic Info** on the instance details page in the console. 
consumerGroupId	You can customize it. After the demo runs successfully, you can see the consumer in **Consumer Group** on the instance details page.

Step 2. Send messages

1. Write a message production program.

```
package main
import (
"fmt"
"gokafkademo/config"
"log"
"strings"
"github.com/confluentinc/confluent-kafka-go/kafka"
)
func main() {
cfg, err := config.ParseConfig("../config/kafka.json")
if err != nil {
log.Fatal(err)
}
p, err := kafka.NewProducer(&kafka.ConfigMap{
// Set the access point of the topic, which can be obtained in the console
"bootstrap.servers": strings.Join(cfg.Servers, ","),
// If you do not configure this parameter, the default value will be 1. You ca
n customize this according to your business requirements.
"acks": 1,
```

```
// Number of retries upon request error. It is recommended that you set the pa
rameter to a value greater than 0 to enable retries and guarantee that message
s are not lost to the greatest extent possible.
"retries": 0,
// Retry interval upon request failure
"retry.backoff.ms": 100,
// Timeout duration of a producer network request
"socket.timeout.ms": 6000,
// Set the interval between retries for the client
"reconnect.backoff.max.ms": 3000,
})
if err != nil {
log.Fatal(err)
}
defer p.Close()
// Deliver the produced messages to the report processor
go func() {
for e := range p.Events() {
switch ev := e.(type) {
case *kafka.Message:
if ev.TopicPartition.Error != nil {
fmt.Printf("Delivery failed: %v\n", ev.TopicPartition)
} else {
fmt.Printf("Delivered message to %v\n",ev.TopicPartition)
}
}
}
}()
// Send messages in async mode
topic := cfg.Topic[0]
for _, word := range []string{"Confluent-Kafka", "Golang Client Message"} {
_ = p.Produce(&kafka.Message{
TopicPartition: kafka.TopicPartition{Topic: &topic, Partition: kafka.Partition
Any},
Value: []byte(word),
}, nil)
}
// Wait for message delivery
p.Flush(10 * 1000)
}
```

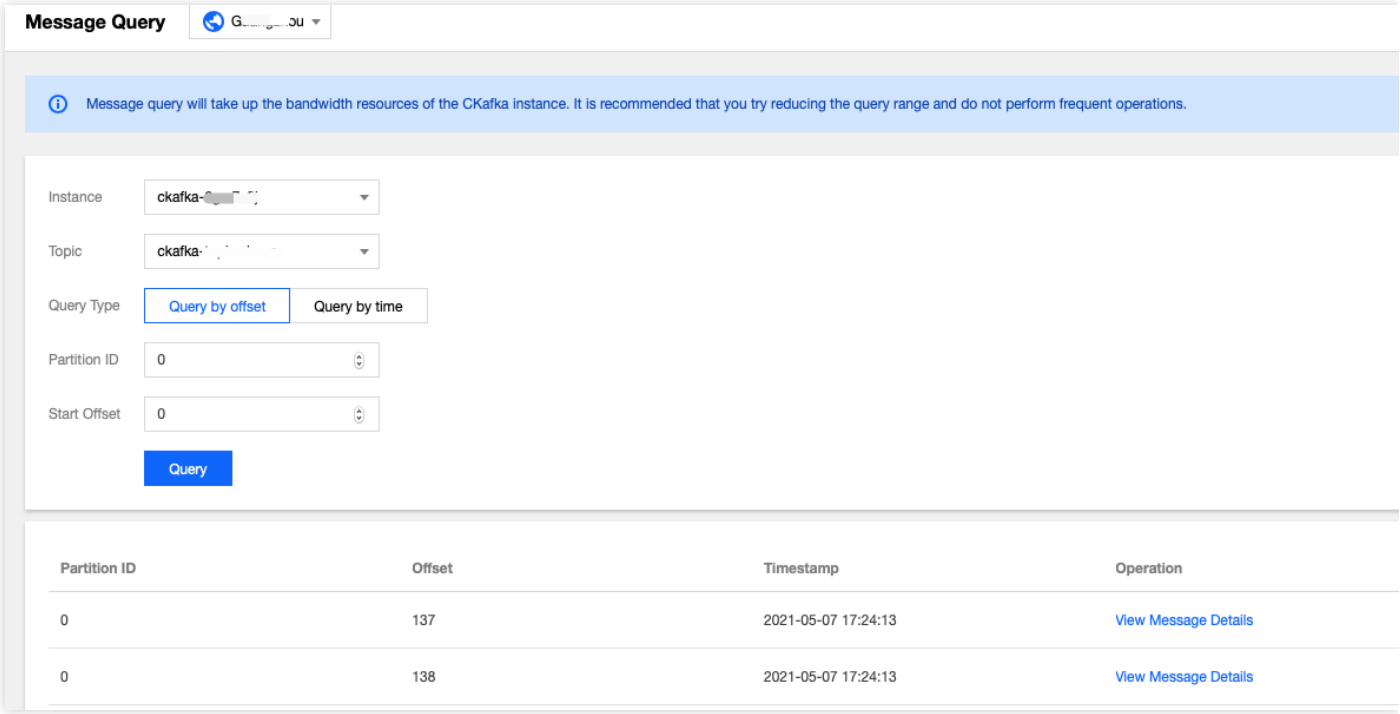
2. Compile and run the program to send messages.

```
go run main.go
```

3. View the execution result. Below is a sample:

```
Delivered message to test[0]@628
Delivered message to test[0]@629
```

4. On the **Topic Management** tab page on the instance details page in the CKafka console, select the topic, and click **More** > **Message Query** to view the messages just sent.



Step 3. Consume messages

1. Write the message consumption program.

```
package main
import (
"fmt"
"gokafkademo/config"
"log"
"strings"
"github.com/confluentinc/confluent-kafka-go/kafka"
)
func main() {
cfg, err := config.ParseConfig("../config/kafka.json")
if err != nil {
log.Fatal(err)
}
```

```
c, err := kafka.NewConsumer(&kafka.ConfigMap{
// Set the access point of the topic, which can be obtained in the console
"bootstrap.servers": strings.Join(cfg.Servers, ","),
// The set message consumer group
"group.id": cfg.ConsumerGroupId,
"auto.offset.reset": "earliest",

// Consumer timeout period when the Kafka consumer grouping mechanism is used.
If the broker does not receive the heartbeat of the consumer within this perio
d, the consumer will be considered to have failed and the broker will initiate
rebalance.
// Currently, this value must be configured in the broker between 6000 (value
of group.min.session.timeout.ms) and 300000 (value of group.max.session.timeou
t.ms).
"session.timeout.ms": 10000,
})

if err != nil {
log.Fatal(err)
}
// List of subscribed message topics
err = c.SubscribeTopics(cfg.Topic, nil)
if err != nil {
log.Fatal(err)
}

for {
msg, err := c.ReadMessage(-1)
if err == nil {
fmt.Printf("Message on %s: %s\n", msg.TopicPartition, string(msg.Value))
} else {
// The client will automatically try to recover all errors
fmt.Printf("Consumer error: %v (%v)\n", err, msg)
}
}

c.Close()
}
```

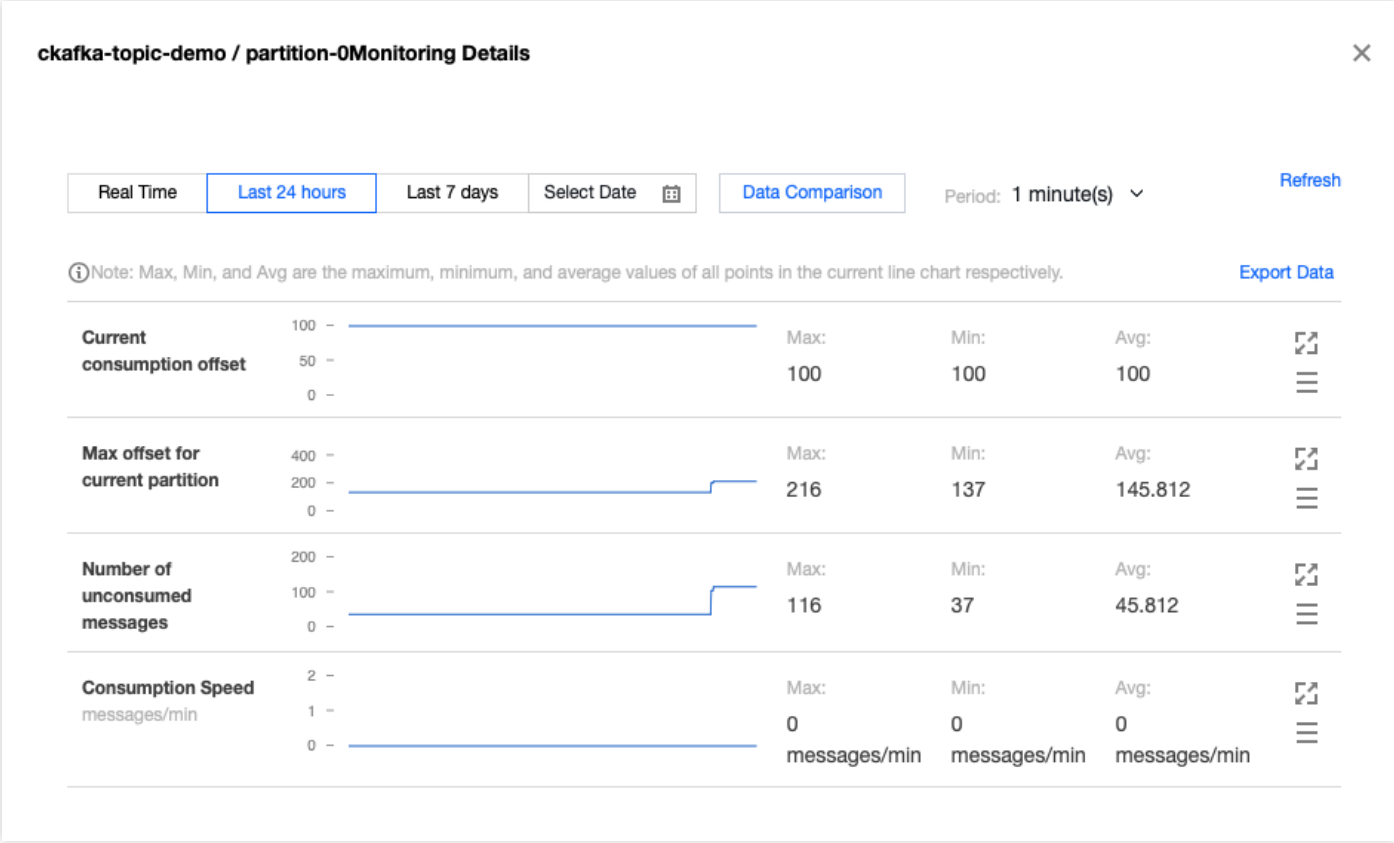
2. Compile and run the program to consume messages.

```
go run main.go
```

3. View the execution result. Below is a sample:

```
Message on test[0]@628: Confluent-Kafka
Message on test[0]@629: Golang Client Message
```

4. On the **Consumer Group** tab page in the CKafka console, select the consumer group name, enter the topic name, and click **View Details** to view the consumption details.



Public Network Access Through SASL_PLAINTEXT

Last updated：2024-01-09 15:00:32

Overview

This document describes how to access CKafka to send/receive messages with the SDK for Go through SASL_PLAINTEXT over public network.

Prerequisites

Install Go
Configure an ACL policy
Download demo

Directions

Step 1. Prepare the Go dependent library

Download the demo, enter the `gokafkademo` directory, and run the following command to install it.

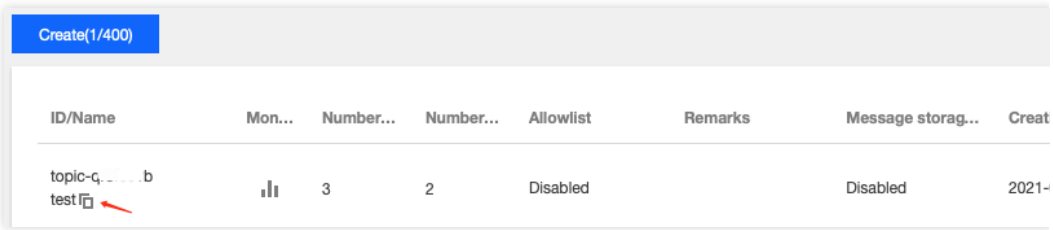
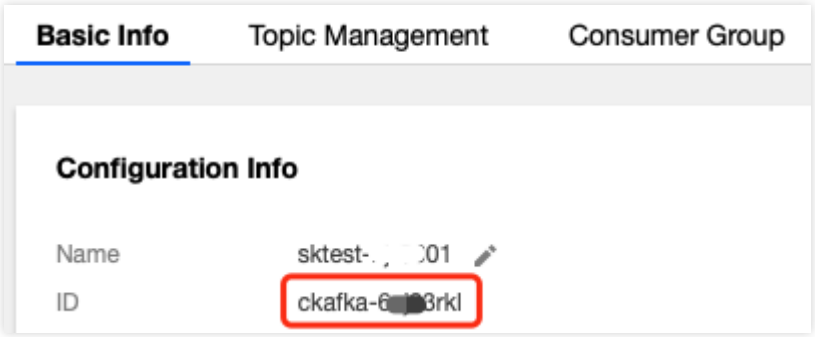
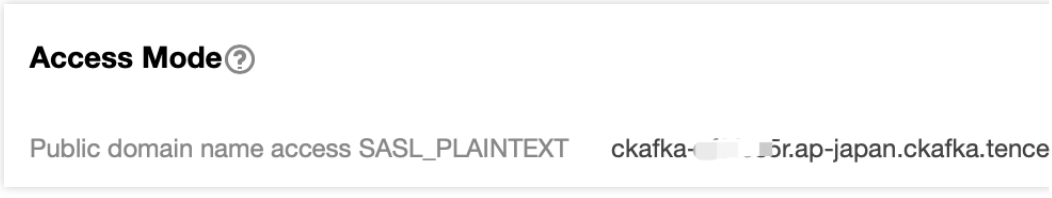
```
go get -v gopkg.in/confluentinc/confluent-kafka-go.v1/kafka
```

Step 2. Prepare configurations

Create a configuration file named `kafka.json` .

```
{
  "topic": [
      "xxxx"
  ],
  "sasl": {
      "username": "yourUserName",
      "password": "yourPassword",
      "instanceId":"instanceId"
  },
  "bootstrapServers": [
      "xxx.ap-changsha-ec.ckafka.tencentcloudmq.com:6000"
```

```
  ],
  "consumerGroupId": "yourConsumerId"
}
```

Parameter	Description
topic	Topic name, which can be copied in **Topic Management** on the instance details page in the 
sasl.username	Username, which is set when the user is created in **User Management** on the instance detai
sasl.password	Password, which is set when the user is created in **User Management** on the instance detail
sasl.instanceId	Instance ID, which can be obtained in **Basic Info** on the instance details page in the console. 
bootstrapServers	Accessed network, which can be copied from the **Network** column in the **Access Mode** sect details page in the console. 
consumerGroupId	You can customize it. After the demo runs successfully, you can see the consumer on the **Cor**

Step 3. Send messages

1. Write a message production program.

```
package main

import (
    "fmt"
    "gokafkademo/config"
    "log"
    "strings"

    "github.com/confluentinc/confluent-kafka-go/kafka"
)

func main() {

    cfg, err := config.ParseConfig("../config/kafka.json")
    if err != nil {
        log.Fatal(err)
    }

    p, err := kafka.NewProducer(&kafka.ConfigMap{
        // Set the access point of the corresponding topic, which can be obtained
        "bootstrap.servers": strings.Join(cfg.Servers, ","),
        // The SASL authentication mechanism is PLAIN by default
        "sasl.mechanism": "PLAIN",
        // Configure an ACL policy locally
        "security.protocol": "SASL_PLAINTEXT",
        // Set “username” to a value in the format of “instance ID + # + configur
        "sasl.username": fmt.Sprintf("%s#%s", cfg.SASL.InstanceId, cfg.SASL.Usern
        "sasl.password": cfg.SASL.Password,

        // Kafka producer supports 3 ACK mechanisms:
        // -1 or all: the broker responds to the producer and continues to send t
        // This configuration provides the highest data reliability, as messages
        // It can be used together with the “min.insync.replicas” parameter at th
        // 0: the producer continues to send the next message or next batch of me
        // 1: the producer sends the next message or next batch of messages after
        // If you do not configure this parameter, the default value will be 1. Y
        "acks": 1,
        // Number of retries upon request error. It is recommended that you set t
        "retries": 0,
        // Retry interval upon request failure
        "retry.backoff.ms": 100,
        // Timeout duration of a producer network request
        "socket.timeout.ms": 6000,
```

```
        // Set the interval between retries for the client
        "reconnect.backoff.max.ms": 3000,
    })
    if err != nil {
        log.Fatal(err)
    }

    defer p.Close()

    // Deliver the produced messages to the report processor
    go func() {
        for e := range p.Events() {
            switch ev := e.(type) {
            case *kafka.Message:
                if ev.TopicPartition.Error != nil {
                    fmt.Printf("Delivery failed: %v\\n", ev.TopicPartition)
                } else {
                    fmt.Printf("Delivered message to %v\\n", ev.TopicPartition)
                }
            }
        }
    }()

    // Send messages in async mode
    topic := cfg.Topic
    for _, word := range []string{"Confluent-Kafka", "Golang Client Message"} {
        _ = p.Produce(&kafka.Message{
            TopicPartition: kafka.TopicPartition{Topic: &topic, Partition: kafka.
            Value:          []byte(word),
        }, nil)
    }

    // Wait for message delivery
    p.Flush(10 * 1000)
```

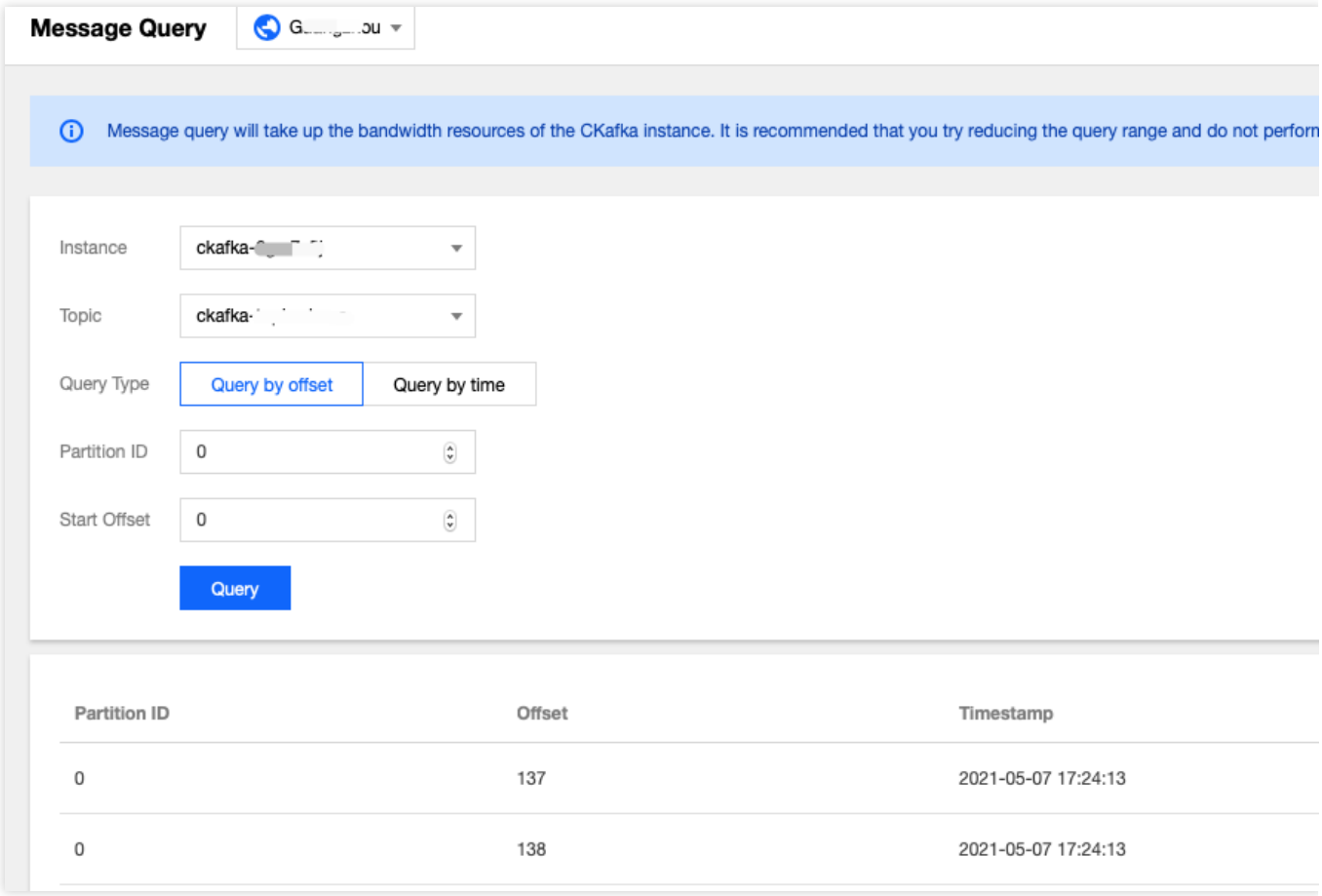
2. Compile and run the program to send messages.

```
go run main.go
```

3. View the execution result. Below is a sample:

```
Delivered message to test[0]@628
Delivered message to test[0]@629
```

4. On the **Topic Management** tab page on the instance details page in the CKafka console, select the corresponding topic, and click **More** > **Message Query** to view the messages just sent.



Step 4. Consume messages

1. Write the message consumption program.

```
package main

import (
    "fmt"
    "gokafkademo/config"
    "log"
    "strings"

    "github.com/confluentinc/confluent-kafka-go/kafka"
)

func main() {

    cfg, err := config.ParseConfig("../config/kafka.json")
    if err != nil {
        log.Fatal(err)
    }
```

```
    c, err := kafka.NewConsumer(&kafka.ConfigMap{
        // Set the access point of the corresponding topic, which can be obtained
        "bootstrap.servers": strings.Join(cfg.Servers, ","),
        // The SASL authentication mechanism is PLAIN by default
        "sasl.mechanism": "PLAIN",
        // Configure an ACL policy locally
        "security.protocol": "SASL_PLAINTEXT",
        // Set “username” to a value in the format of “instance ID + # + configur
        "sasl.username": fmt.Sprintf("%s#%s", cfg.SASL.InstanceId, cfg.SASL.Usern
        "sasl.password": cfg.SASL.Password,
        // The set message consumer group
        "group.id":          cfg.ConsumerGroupId,
        "auto.offset.reset": "earliest",

        // Consumer timeout period when the Kafka consumer grouping mechanism is
        // Currently, this value must be configured in the broker between 6000 (v
        "session.timeout.ms": 10000,
    })

    if err != nil {
        log.Fatal(err)
    }
    // List of subscribed message topics
    err = c.SubscribeTopics([]string{"test", "test-topic"}, nil)
    if err != nil {
        log.Fatal(err)
    }

    for  {
        msg, err := c.ReadMessage(-1)
        if err == nil {
            fmt.Printf("Message on %s: %s\\n", msg.TopicPartition, string(msg.Val
        } else {
            // The client will automatically try to recover all errors
            fmt.Printf("Consumer error: %v (%v)\\n", err, msg)
        }
    }

    c.Close()
}
```

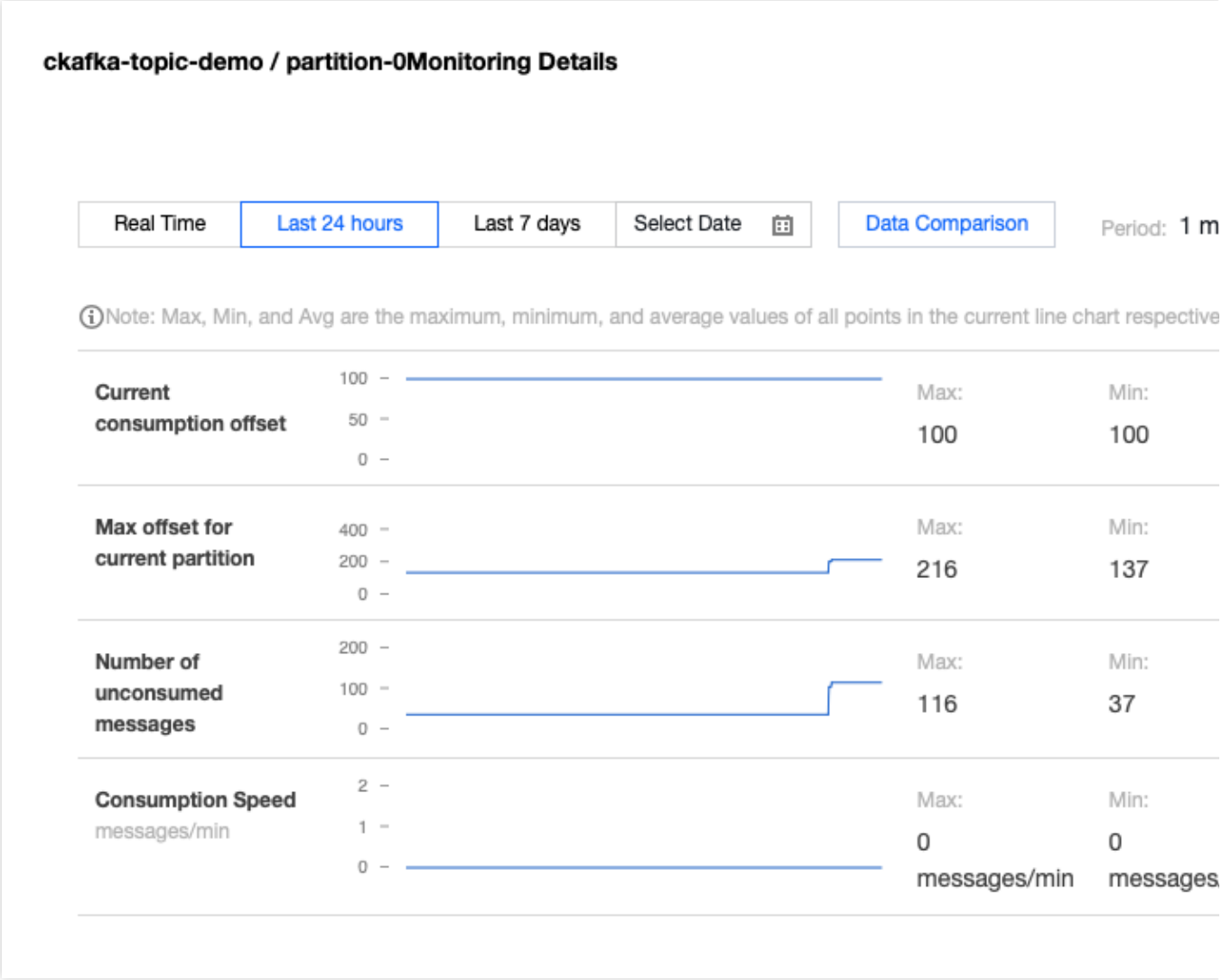
2. Compile and run the program to consume messages.

```
go run main.go
```

3. View the execution result. Below is a sample:

```
Message on test[0]@628: Confluent-Kafka
Message on test[0]@629: Golang Client Message
```

4. On the **Consumer Group** tab page on the instance details page in the CKafka console, select the corresponding consumer group, enter the topic name, and click **View Details** to view the consumption details.



SDK for PHP
VPC Access

Last updated：2024-01-09 15:00:32

Overview

This document describes how to access CKafka to send/receive messages with the SDK for PHP in a VPC.

Prerequisites

Install librdkafka
Install PHP 5.6 or above
Install PEAR
Download the demo

Directions

Step 1. Add the Rdkafka extension

1. Find the latest version of Rdkafka extension package for PHP here.

Note:

Different package versions require different PHP versions. 4.1.2 is used here as an example.

2. Log in to the Linux server and install the Rdkafka extension.

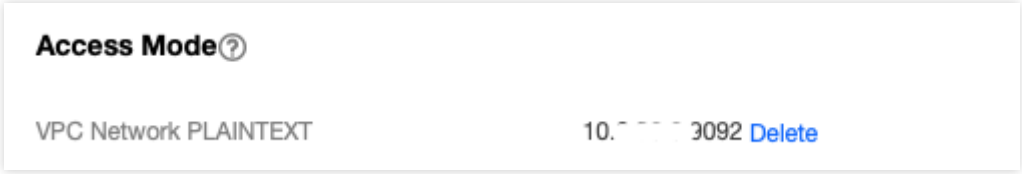
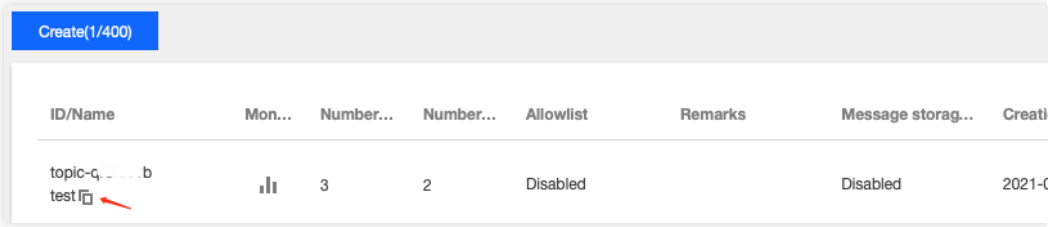
```
wget --no-check-certificate https://pecl.php.net/get/rdkafka-4.1.2.tgz
pear install rdkafka-4.1.2.tgz
# If the installation succeeds, the system will prompt "install ok" and "You should
# If the installation fails, please follow the prompts to troubleshoot
# After successful installation, add `extension=rdkafka.so` to `php.ini`
# After `php --ini` is executed, `Loaded Configuration File:` shows the location of
echo 'extension=rdkafka.so' >> /etc/php.ini
```

Step 2. Prepare configurations

1. Upload the `phpkafkademo` in the downloaded demo to the Linux server.

2. Log in to the Linux server, enter the `phpkafkademo` directory, and modify the `CKafkaSetting.php` configuration file.

```
<?php
return [
 'bootstrap_servers' => 'bootstrap_servers1:port,bootstrap_servers2:port',
 'topic_name' => 'topic_name',
 'group_id' => 'php-demo',
];
```

Parameter	Description
bootstrap_servers	Accessed network, which can be copied from the **Network** column in the **Access Mode** sect in the console. 
topic_name	Topic name, which can be copied from the **Topic Management** page in the console. 
group_id	Consumer group ID. You can customize it. After the demo runs successfully, you can see the **Group** page.

Step 3. Send a message

1. Write the message production program `Producer.php` .

```
<?php

$setting = require __DIR__ . '/CKafkaSetting.php';

$conf = new RdKafka\\Conf();
// To set the entry service, please get the corresponding service address in the co
```

```
$conf->set('bootstrap.servers', $setting['bootstrap_servers']);
// There are three ack mechanisms for a Kafka producer as described below:
// -1 or all: the broker responds to the producer and continues to send the next me
// This configuration provides the highest data reliability, and messages will neve
// It can be used together with the `min.insync.replicas` parameter at the topic le
// 0: the producer does not wait for acknowledgment of sync completion from the bro
// (Data loss may occur when the server fails. If the leader is dead but the produc
// 1: the producer sends the next message or next batch of messages after it receiv
// (Messages may be lost if the leader is dead but has not been replicated yet.)
// If you do not explicitly configure this, the value of 1 will be used by default.
$conf->set('acks', '1');
// Number of retries upon request error. We recommend you set the value to be great
$conf->set('retries', '0');
// The time between when a request fails and the next time the request is retried
$conf->set('retry.backoff.ms', 100);
// Timeout period of the producer network request
$conf->set('socket.timeout.ms', 6000);
$conf->set('reconnect.backoff.max.ms', 3000);

// Register the callback for message sending
$conf->setDrMsgCb(function ($kafka, $message) {
  echo '[Producer] sent a message: message=' . var_export($message, true) . "\\n";
});
// Register the callback for message sending error
$conf->setErrorCb(function ($kafka, $err, $reason) {
  echo "[Producer] run into an error while sending the message: err=$err reason=$re
});

$producer = new RdKafka\\Producer($conf);
// Please set `LOG_DEBUG` when debugging
//$producer->setLogLevel(LOG_DEBUG);
$topicConf = new RdKafka\\TopicConf();
$topic = $producer->newTopic($setting['topic_name'], $topicConf);
// Produce a message and send it
for ($i = 0; $i < 5; $i++) {
  // `RD_KAFKA_PARTITION_UA` lets Kafka select a partition freely
  $topic->produce(RD_KAFKA_PARTITION_UA, 0, "Message $i");
  $producer->poll(0);
}

while ($producer->getOutQLen() > 0) {
  $producer->poll(50);
}

echo "[Producer] sent the message successfully\\n";
```

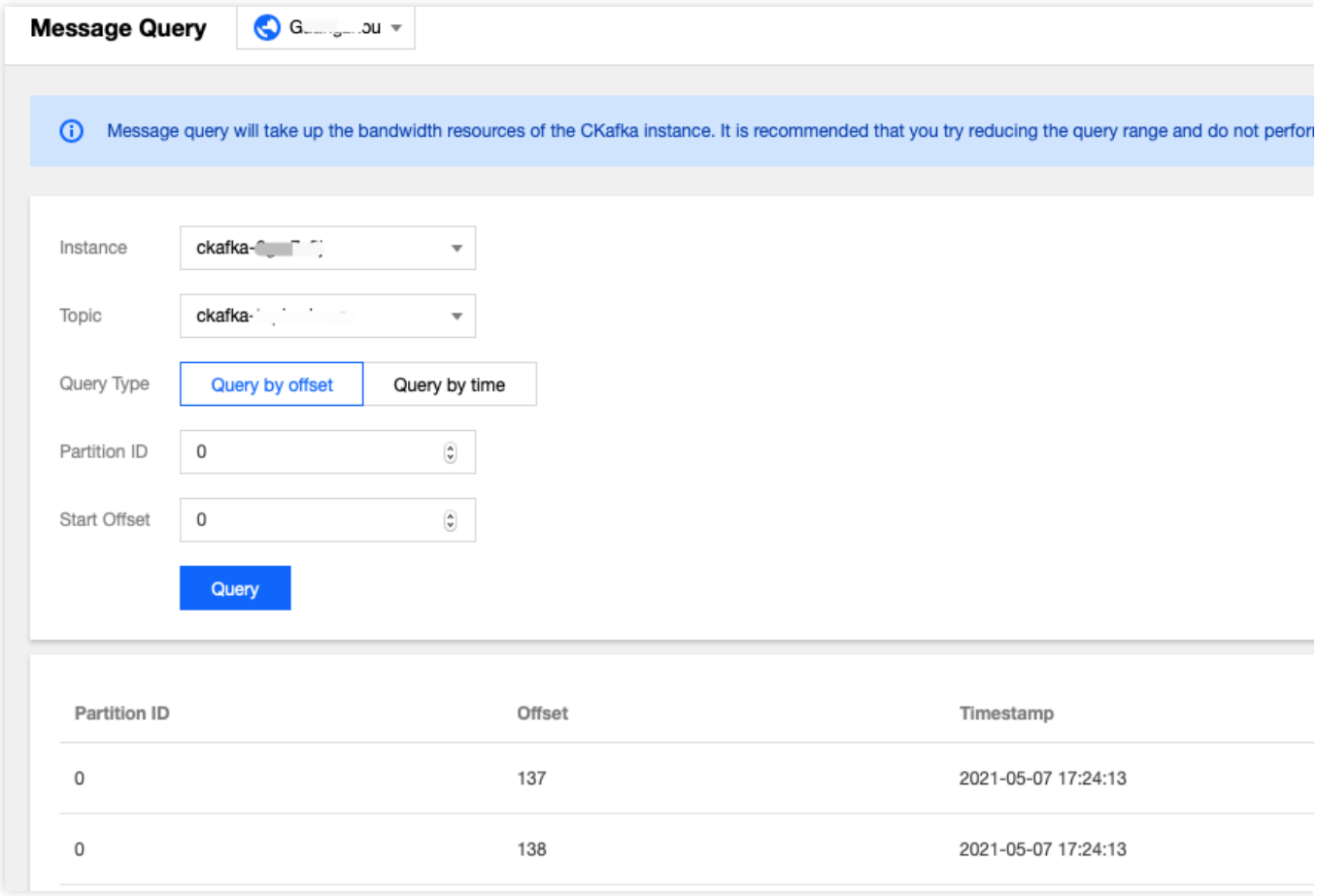
2. Run `Producer.php` to send the message.

```
php Producer.php
```

3. View the execution result. Below is a sample:

```
>[Producer] sent a message: message=RdKafka\\Message::__set_state(array(
>   'err' => 0,
>   'topic_name' => 'topic_name',
>   'timestamp' => 1618800895159,
>   'partition' => 0,
>   'payload' => 'Message 0',
>   'len' => 9,
>   'key' => NULL,
>   'offset' => 0,
>   'headers' => NULL,
>))
>[Producer] sent a message: message=RdKafka\\Message::__set_state(array(
>   'err' => 0,
>   'topic_name' => 'topic_name',
>   'timestamp' => 1618800895159,
>   'partition' => 0,
>   'payload' => 'Message 1',
>   'len' => 9,
>   'key' => NULL,
>   'offset' => 1,
>   'headers' => NULL,
>))

...

>[Producer] sent the message successfully
```

4. On the **Topic Management** page in the CKafka console, select the corresponding topic and click **More** > **Message Query** to view the just sent message.



Step 4. Consume the message

1. Write the subscribe message consumer program `Consumer.php` .

```
<?php

$setting = require __DIR__ . '/CKafkaSetting.php';

$conf = new RdKafka\\Conf();
$conf->set('group.id', $setting['group_id']);
// To set the entry service, please get the corresponding service address in the co
$conf->set('bootstrap.servers', $setting['bootstrap_servers']);
// Consumer timeout period when the Kafka consumer grouping mechanism is used. If t
// the consumer will be considered to have failed and the broker will initiate reba
$conf->set('session.timeout.ms', 10000);
// Client request timeout period. If no response is received after this time elapse
$conf->set('request.timeout.ms', 305000);
// Set the internal retry interval of the client
$conf->set('reconnect.backoff.max.ms', 3000);

$topicConf = new RdKafka\\TopicConf();
#$topicConf->set('auto.commit.interval.ms', 100);
```

```
// Offset reset policy. Please set as appropriate according to the business scenari
$topicConf->set('auto.offset.reset', 'earliest');
$conf->setDefaultTopicConf($topicConf);

$consumer = new RdKafka\\KafkaConsumer($conf);
// Please set `LOG_DEBUG` when debugging
//$consumer->setLogLevel(LOG_DEBUG);
$consumer->subscribe([$setting['topic_name']]);

$isConsuming = true;
while ($isConsuming) {
  $message = $consumer->consume(10 * 1000);
  switch ($message->err) {
      case RD_KAFKA_RESP_ERR_NO_ERROR:
          echo "[Consumer] received a message:" . var_export($message, true) . "\\n
          break;
      case RD_KAFKA_RESP_ERR__PARTITION_EOF:
          echo "[Consumer] is waiting for messages\\n";
          break;
      case RD_KAFKA_RESP_ERR__TIMED_OUT:
          echo "[Consumer] waiting timed out\\n";
          $isConsuming = false;
          break;
      default:
          throw new \\Exception($message->errstr(), $message->err);
          break;
  }
}
```

2. Run `Consumer.php` to consume the message.

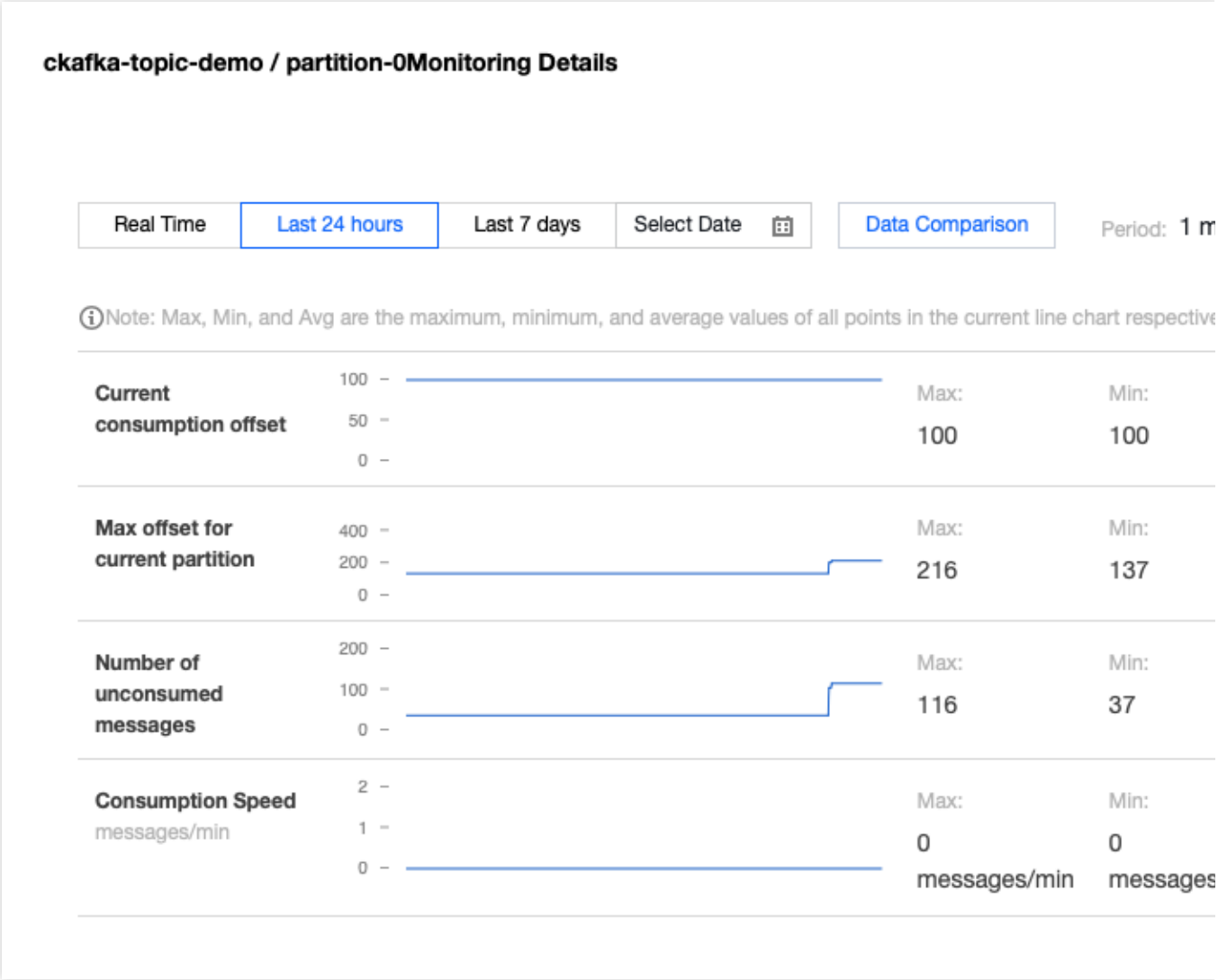
```
php Consumer.php
```

3. View the operation result.

```
>[Consumer] received a message: RdKafka\\Message::__set_state(array(
>   'err' => 0,
>   'topic_name' => 'topic_name',
>   'timestamp' => 1618800895159,
>   'partition' => 0,
>   'payload' => 'Message 0',
>   'len' => 9,
>   'key' => NULL,
>   'offset' => 0,
>   'headers' => NULL,
>))
>[Consumer] received a message: RdKafka\\Message::__set_state(array(
```

```
>   'err' => 0,
>   'topic_name' => 'topic_name',
>   'timestamp' => 1618800895159,
>   'partition' => 0,
>   'payload' => 'Message 1',
>   'len' => 9,
>   'key' => NULL,
>   'offset' => 1,
>   'headers' => NULL,
>))

...
```

4. On the **Consumer Group** page in the CKafka console, select the corresponding consumer group, enter the topic name in **Topic Name**, and click **Query Details** to view the consumption details.



Public Network Access Through SASL_PLAINTEXT

Last updated：2024-01-09 15:00:32

Overview

This document describes how to access CKafka to send/receive messages with the SDK for PHP through SASL_PLAINTEXT over public network.

Prerequisites

Install librdkafka
Install PHP 5.6 or later
Install PEAR
Configure an ACL policy
Download demo

Directions

Step 1. Add rdkafka extension

1. On the rdkafka official website, find the PHP-rdkafka extension package of the latest version.

Note:

PHP-rdkafka extension packages of different versions have different requirements on the PHP version. This procedure takes the PHP-rdkafka extension package of version 4.1.2 as an example.

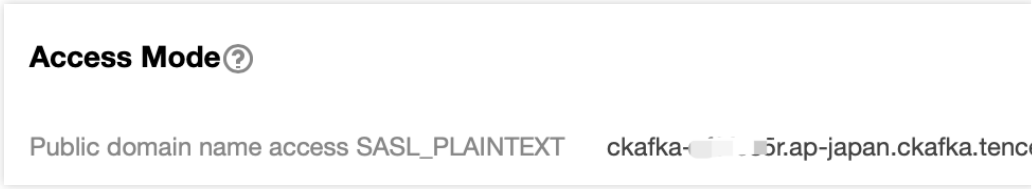
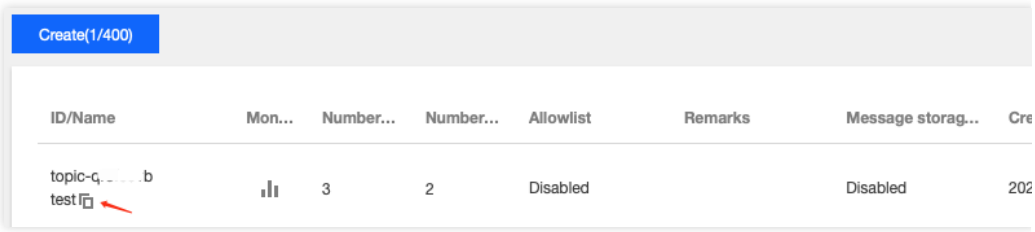
2. Install the rdkafka extension.

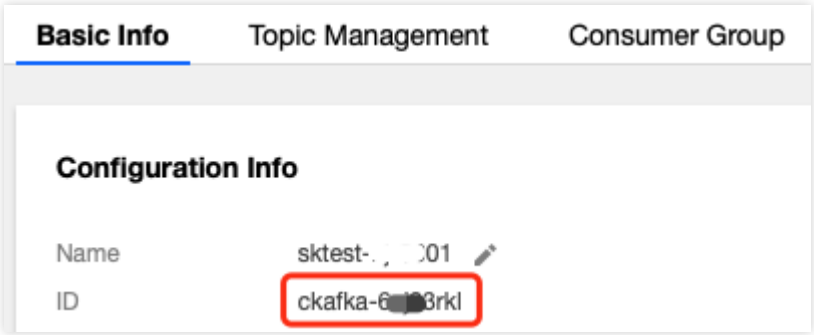
```
wget --no-check-certificate https://pecl.php.net/get/rdkafka-4.1.2.tgz
pear install rdkafka-4.1.2.tgz
# If the installation is successful, “install ok” and “You should add “extension=rd
# If the installation fails and the system prompts “could not extract the package.x
# For other errors, please follow the prompts to troubleshoot
# After successful installation, add “extension=rdkafka.so” to “php.ini”
# After you run “php --ini”, the value of “Loaded Configuration File” is the locati
echo 'extension=rdkafka.so' >> /etc/php.ini
```

Step 2. Prepare configurations

Create a configuration file named `CKafkaSetting.php` .

```
<?php
return [
    'bootstrap_servers' => 'bootstrap_servers1:port,bootstrap_servers2:port',
    'topic_name' => 'topic_name',
    'group_id' => 'php-demo',
    'ckafka_instance_id' => 'ckafka_instance_id',
    'sasl_username' => 'username',
    'sasl_password' => 'password'
];
```

Parameter	Description
bootstrap_servers	Accessed network, which can be copied from the **Network** column in the **Access Mode** se details page in the console. 
topic_name	Topic name, which can be copied in **Topic Management** on the instance details page in th 
group_id	Consumer group ID. You can customize it according to your business needs. After the demo consumer on the **Consumer Group** page.
ckafka_instance_id	Instance ID, which can be obtained in **Basic Info** on the instance details page in the CKafka

	
sasl.username	Username, which is set when the user is created in **User Management** on the instance deta
sasl_password	Password, which is set when the user is created in **User Management** on the instance deta

Step 3. Send messages

1. Write a message production program named `Producer.php` .

```
<?php

$setting = require __DIR__ . '/CKafkaSetting.php';

$conf = new RdKafka\\Conf();
// Set the ingress service. Obtain the corresponding service address in the console
$conf->set('bootstrap.servers', $setting['bootstrap_servers']);
// ---------- Set this part if SASL authentication is enabled ----------
// The SASL authentication mechanism is PLAIN by default
$conf->set('sasl.mechanism', 'PLAIN');
// Set the username. Format: instance ID + # + username specified on the **User Man
$conf->set('sasl.username', $setting['ckafka_instance_id'] . '#' . $setting['sasl_u
// Set the password, which is set on the **User Management** tab page in the consol
$conf->set('sasl.password', $setting['sasl_password']);
// Configure an ACL policy locally
$conf->set('security.protocol', 'SASL_PLAINTEXT');
// ---------- Set this part if SASL authentication is enabled ----------
// Kafka producer supports 3 ACK mechanisms:
// -1 or all: the broker responds to the producer and continues to send the next me
// This configuration provides the highest data reliability, as messages will never
// It can be used together with the “min.insync.replicas” parameter at the topic le
// 0: the producer continues to send the next message or next batch of messages wit
// (Data may get lost when the server fails. If the leader is dead but the producer
// 1: the producer sends the next message or next batch of messages after it receiv
// (Messages may get lost if the leader is dead but has not been replicated yet.)
// If you do not configure this parameter, the default value will be 1. You can cus
$conf->set('acks', '1');
// Number of retries upon request error. It is recommended that you set the paramet
```

```
$conf->set('retries', '0');
// Retry interval upon request failure
$conf->set('retry.backoff.ms', 100);
// Timeout duration of a producer network request
$conf->set('socket.timeout.ms', 6000);
$conf->set('reconnect.backoff.max.ms', 3000);

// Register a callback for message sending
$conf->setDrMsgCb(function ($kafka, $message) {
 echo '**Producer** sent a message: message=' . var_export($message, true) . "\\n";
});
// Register a callback for message sending errors
$conf->setErrorCb(function ($kafka, $err, $reason) {
 echo "**Producer** message sending error: err=$err reason=$reason \\n";
});

$producer = new RdKafka\\Producer($conf);
// Set the field to LOG_DEBUG for debugging
//$producer->setLogLevel(LOG_DEBUG);
$topicConf = new RdKafka\\TopicConf();
$topic = $producer->newTopic($setting['topic_name'], $topicConf);
// Produce and send a message
for ($i = 0; $i < 5; $i++) {
 // RD_KAFKA_PARTITION_UA: allow Kafka to choose any partition
 $topic->produce(RD_KAFKA_PARTITION_UA, 0, "Message $i");
 $producer->poll(0);
}

while ($producer->getOutQLen() > 0) {
 $producer->poll(50);
}

echo "**Producer** successfully sent the message\\n";
```

2. Run `Producer.php` to send messages.

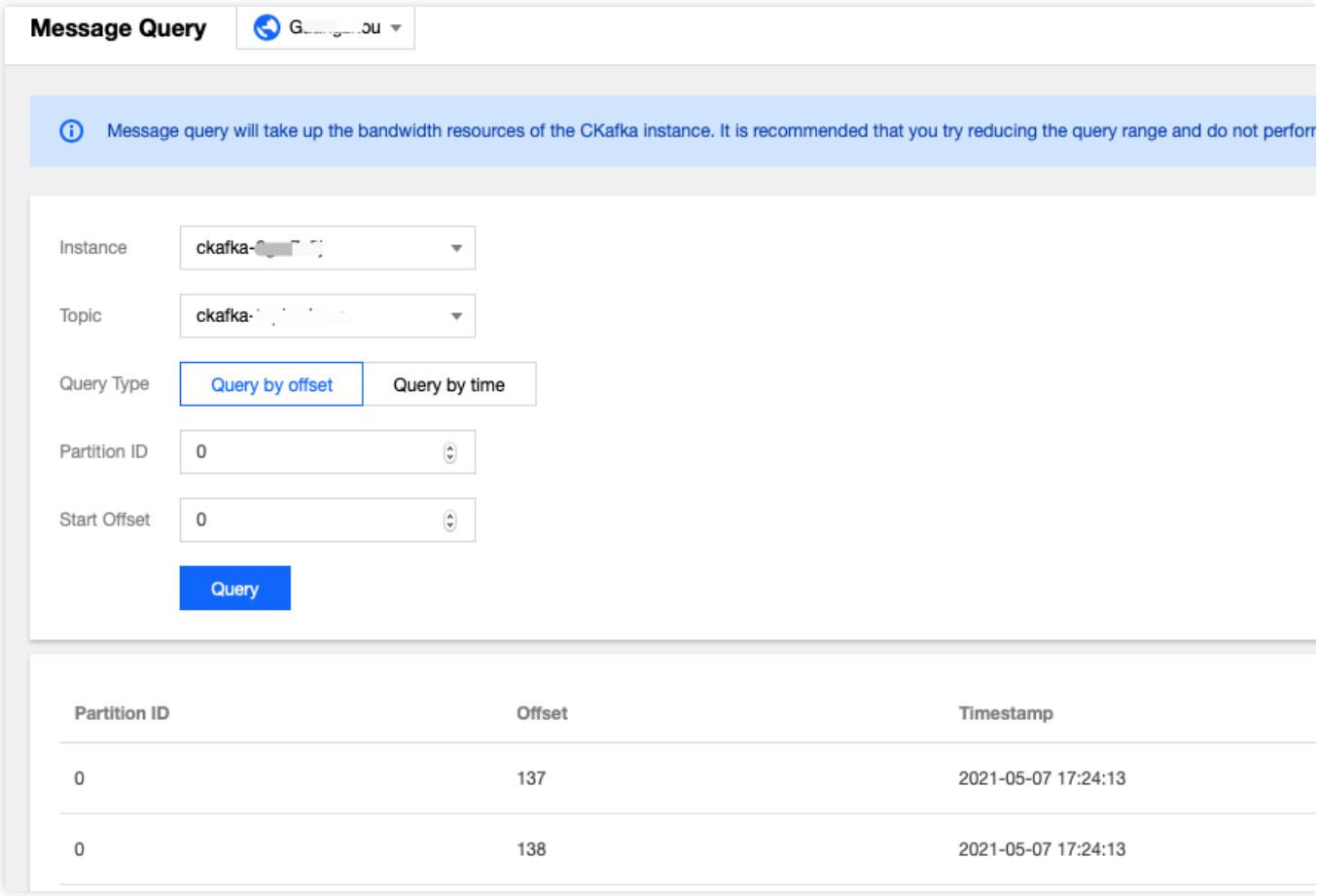
```
php Producer.php
```

3. View the execution result.

```
>**Producer** sent a message: message=RdKafka\\Message::__set_state(array(
>   'err' => 0,
>   'topic_name' => 'topic_name',
>   'timestamp' => 1618800895159,
>   'partition' => 0,
>   'payload' => 'Message 0',
>   'len' => 9,
```

```
>   'key' => NULL,
>   'offset' => 0,
>   'headers' => NULL,
>))
>**Producer** sent a message: message=RdKafka\\Message::__set_state(array(
>   'err' => 0,
>   'topic_name' => 'topic_name',
>   'timestamp' => 1618800895159,
>   'partition' => 0,
>   'payload' => 'Message 1',
>   'len' => 9,
>   'key' => NULL,
>   'offset' => 1,
>   'headers' => NULL,
>))

...

>**Producer** successfully sent the message
```

4. On the **Topic Management** tab page on the instance details page in the CKafka console, select the target topic, and click **More** > **Message Query** to view the message just sent.



Step 4. Consume messages

1. Write a message consumption program named `Consumer.php` .

```
<?php

$setting = require __DIR__ . '/CKafkaSetting.php';

$conf = new RdKafka\\Conf();
$conf->set('group.id', $setting['group_id']);
// Set the ingress service. Obtain the corresponding service address in the console
$conf->set('bootstrap.servers', $setting['bootstrap_servers']);
// ---------- Set this part if SASL authentication is enabled ----------
// The SASL authentication mechanism is PLAIN by default
$conf->set('sasl.mechanism', 'PLAIN');
// Set the username. Format: instance ID + # + username specified on the **User Man
$conf->set('sasl.username', $setting['ckafka_instance_id'] . '#' . $setting['sasl_u
// Set the password, which is set on the **User Management** tab page in the consol
$conf->set('sasl.password', $setting['sasl_password']);
// Configure an ACL policy locally
$conf->set('security.protocol', 'SASL_PLAINTEXT');
// ---------- Set this part if SASL authentication is enabled ----------
```

```
// Consumer timeout period when the Kafka consumer grouping mechanism is used. If t
// the consumer will be considered to have failed and the broker will initiate reba
$conf->set('session.timeout.ms', 10000);
// Client request timeout duration. If no response is received within this duration
$conf->set('request.timeout.ms', 305000);
// Set the interval between retries for the client
$conf->set('reconnect.backoff.max.ms', 3000);

$topicConf = new RdKafka\\TopicConf();
#$topicConf->set('auto.commit.interval.ms', 100);
// Offset resetting policy. Set it according to the business scenario. Improper set
$topicConf->set('auto.offset.reset', 'earliest');
$conf->setDefaultTopicConf($topicConf);

$consumer = new RdKafka\\KafkaConsumer($conf);
// Set the field to LOG_DEBUG for debugging
//$consumer->setLogLevel(LOG_DEBUG);
$consumer->subscribe([$setting['topic_name']]);

$isConsuming = true;
while ($isConsuming) {
 $message = $consumer->consume(10 * 1000);
 switch ($message->err) {
   case RD_KAFKA_RESP_ERR_NO_ERROR:
       echo "**Consumer** received a message:" . var_export($message, true) . "\\n"
       break;
   case RD_KAFKA_RESP_ERR__PARTITION_EOF:
       echo "**Consumer** is waiting for messages\\n";
       break;
   case RD_KAFKA_RESP_ERR__TIMED_OUT:
       echo "**Consumer** wait timed out\\n";
       $isConsuming = false;
       break;
   default:
       throw new \\Exception($message->errstr(), $message->err);
       break;
 }
}
```

2. Run `Consumer.php` to consume messages.

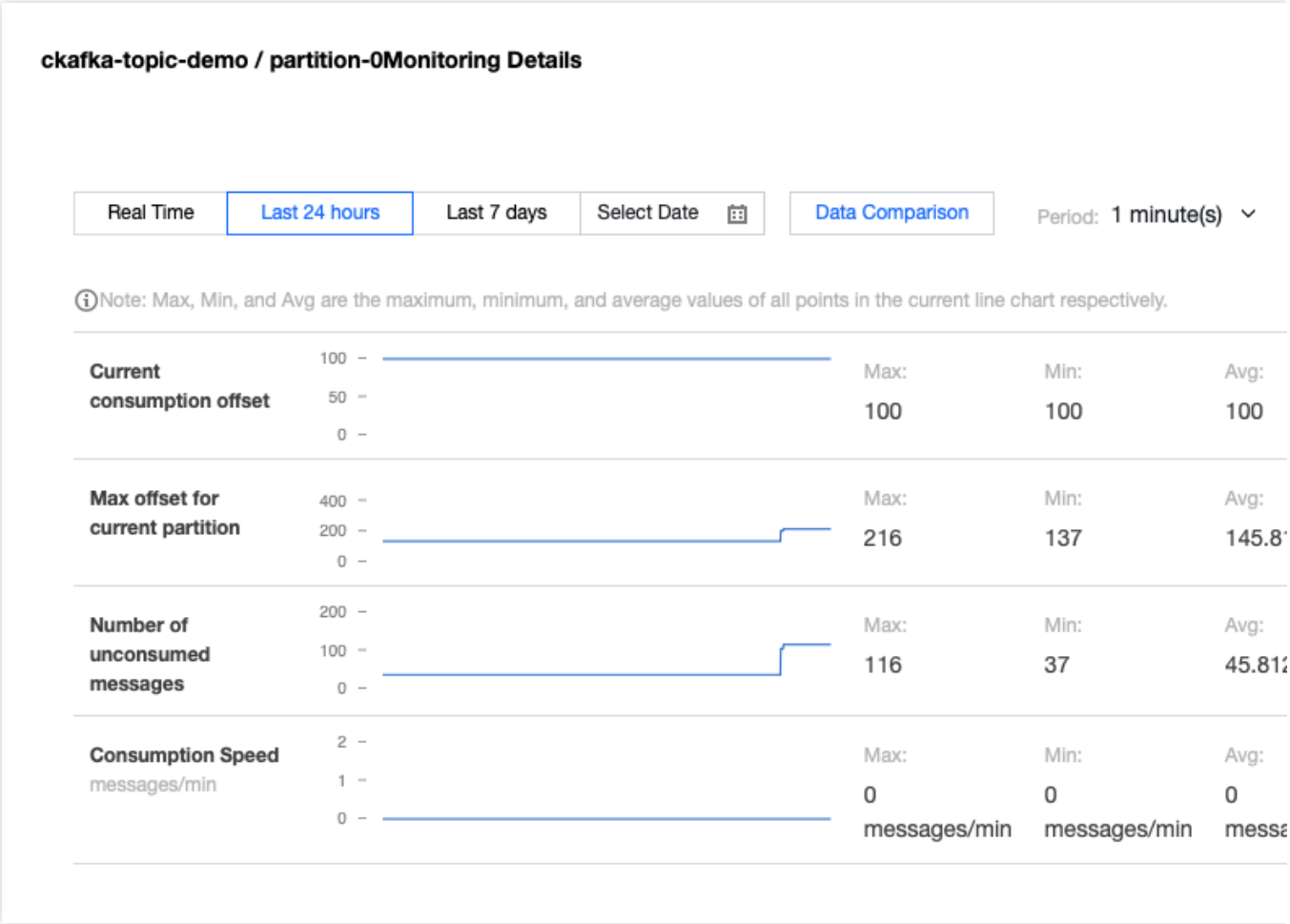
```
php Consumer.php
```

3. View the execution result.

```
>**Consumer** received a message: RdKafka\\Message::__set_state(array(
>   'err' => 0,
```

```
>    'topic_name' => 'topic_name',
>    'timestamp' => 1618800895159,
>    'partition' => 0,
>    'payload' => 'Message 0',
>    'len' => 9,
>    'key' => NULL,
>    'offset' => 0,
>    'headers' => NULL,
>))
>**Consumer** received a message: RdKafka\\Message::__set_state(array(
>    'err' => 0,
>    'topic_name' => 'topic_name',
>    'timestamp' => 1618800895159,
>    'partition' => 0,
>    'payload' => 'Message 1',
>    'len' => 9,
>    'key' => NULL,
>    'offset' => 1,
>    'headers' => NULL,
>))

...
```

4. On the **Consumer Group** tab page on the instance details page in the CKafka console, select the corresponding consumer group, enter the topic name, and click **Query Details** to view the consumption details.



SDK for C++ VPC Access

Last updated：2024-01-09 15:00:32

Overview

This document describes how to access CKafka to send/receive messages with the SDK for C++ in a VPC.

Prerequisites

Install GCC
Download the demo

Directions

Step 1. Install the C/C++ dependency library

1. Upload the `cppkafkademo` in the downloaded demo to the Linux server.
2. Log in to the Linux server and install librdkafka.

Step 2. Send a message

1. Create the `producer.c` file.

```
/*
 * librdkafka - Apache Kafka C library
 *
 * Copyright (c) 2017, Magnus Edenhill
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
 *
```

```
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

/**
 * Simple Apache Kafka producer
 * using the Kafka driver from librdkafka
 * (https://github.com/edenhill/librdkafka)
 */

#include <stdio.h>
#include <signal.h>
#include <string.h>


#include <librdkafka/rdkafka.h>


static volatile sig_atomic_t run = 1;

/**
 * @brief Signal termination of program
 */
static void stop (int sig) {
    run = 0;
    fclose(stdin); /* abort fgets() */
}


/**
 * @brief Message delivery report callback.
 *
 * This callback is called exactly once per message, indicating if
 * the message was succesfully delivered
 * (rkmessage->err == RD_KAFKA_RESP_ERR_NO_ERROR) or permanently
 * failed delivery (rkmessage->err != RD_KAFKA_RESP_ERR_NO_ERROR).
 *
 * The callback is triggered from rd_kafka_poll() and executes on
```

```
 * the application's thread.
 */
static void dr_msg_cb (rd_kafka_t *rk,
                       const rd_kafka_message_t *rkmessage, void *opaque) {
    if (rkmessage->err)
        fprintf(stderr, "%% Message delivery failed: %s\\n",
                rd_kafka_err2str(rkmessage->err));
    else
        fprintf(stderr,
                "%% Message delivered (%zd bytes, "
                "partition %"PRId32")\\n",
            rkmessage->len, rkmessage->partition);

    /* The rkmessage is destroyed automatically by librdkafka */
}



int main (int argc, char **argv) {
    rd_kafka_t *rk;         /* Producer instance handle */
    rd_kafka_conf_t *conf;  /* Temporary configuration object */
    char errstr[512];       /* librdkafka API error reporting buffer */
    char buf[512];          /* Message value temporary buffer */
    const char *brokers;    /* Argument: broker list */
    const char *topic;      /* Argument: topic to produce to */
    const char *user;       /* Argument: sasl username */
    const char *passwd;     /* Argument: sasl password */

    /*
     * Argument validation
     */
    if (argc < 3) {
        fprintf(stderr, "%% Usage: %s <broker> <topic> <username> <password>  \\n O
        return 1;
    }

    brokers = argv[1];
    topic   = argv[2];

    if(argc == 5) {
        user = argv[3];
        passwd = argv[4];
    }


    /*
     * Create Kafka client configuration place-holder
```

```
     */
    conf = rd_kafka_conf_new();

    /* Set bootstrap broker(s) as a comma-separated list of
     * host or host:port (default port 9092).
     * librdkafka will use the bootstrap brokers to acquire the full
     * set of brokers from the cluster. */
    if (rd_kafka_conf_set(conf, "bootstrap.servers", brokers,
                          errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK) {
        fprintf(stderr, "%s\\n", errstr);
        return 1;
    }

    /* Set sasl config*/
    if( user && passwd ) {
        if(rd_kafka_conf_set(conf,"security.protocol","sasl_plaintext",errstr,sizeo
            fprintf(stderr,"%s\\n",errstr);
            return 1;
        }
        if(rd_kafka_conf_set(conf, "sasl.mechanisms", "PLAIN", errstr, sizeof(errst
            fprintf(stderr,"%s\\n",errstr);
            return 1;
        }
        if(rd_kafka_conf_set(conf,"sasl.username",user,errstr,sizeof(errstr)) !=RD_
            fprintf(stderr,"%s\\n",errstr);
            return 1;

        }
        if(rd_kafka_conf_set(conf,"sasl.password",passwd,errstr,sizeof(errstr)) !=R
            fprintf(stderr,"%s\\n",errstr);
            return 1;
        }

    }
    /* Tencent Cloud recommended configuration parameters
      * https://cloud.tencent.com/document/product/597/30203
      */
    //if(rd_kafka_conf_set(conf,"debug","none",errstr,sizeof(errstr))!= RD_KAFKA_CO
        //fprintf(stderr,"%s\\n",errstr);
        //return 1;
    //}

    if(rd_kafka_conf_set(conf,"acks","1",errstr,sizeof(errstr)) != RD_KAFKA_CONF_OK
        fprintf(stderr,"%s\\n",errstr);
        return 1;
    }
    if(rd_kafka_conf_set(conf,"request.timeout.ms","30000",errstr,sizeof(errstr)) !
```

```
        fprintf(stderr,"%s\\n",errstr);
        return 1;
    }
    if(rd_kafka_conf_set(conf,"retries","3",errstr,sizeof(errstr)) != RD_KAFKA_CONF
        fprintf(stderr,"%s\\n",errstr);
        return 1;
    }

    if(rd_kafka_conf_set(conf,"retry.backoff.ms","1000",errstr,sizeof(errstr)) != R
        fprintf(stderr,"%s\\n",errstr);
        return 1;
    }

    /* Set the delivery report callback.
     * This callback will be called once per message to inform
     * the application if delivery succeeded or failed.
     * See dr_msg_cb() above.
     * The callback is only triggered from rd_kafka_poll() and
     * rd_kafka_flush(). */
    rd_kafka_conf_set_dr_msg_cb(conf, dr_msg_cb);

    /*
     * Create producer instance.
     *
     * NOTE: rd_kafka_new() takes ownership of the conf object
     *       and the application must not reference it again after
     *       this call.
     */
    rk = rd_kafka_new(RD_KAFKA_PRODUCER, conf, errstr, sizeof(errstr));
    if (!rk) {
        fprintf(stderr,
                "%% Failed to create new producer: %s\\n", errstr);
        return 1;
    }

    /* Signal handler for clean shutdown */
    signal(SIGINT, stop);

    fprintf(stderr,
            "%% Type some text and hit enter to produce message\\n"
            "%% Or just hit enter to only serve delivery reports\\n"
            "%% Press Ctrl-C or Ctrl-D to exit\\n");

    while (run && fgets(buf, sizeof(buf), stdin)) {
        size_t len = strlen(buf);
        rd_kafka_resp_err_t err;
```

```
        if (buf[len-1] == '\\n') /* Remove newline */
            buf[--len] = '\\0';

        if (len == 0) {
            /* Empty line: only serve delivery reports */
            rd_kafka_poll(rk, 0/*non-blocking */);
            continue;
        }

        /*
         * Send/Produce message.
         * This is an asynchronous call, on success it will only
         * enqueue the message on the internal producer queue.
         * The actual delivery attempts to the broker are handled
         * by background threads.
         * The previously registered delivery report callback
         * (dr_msg_cb) is used to signal back to the application
         * when the message has been delivered (or failed).
         */
        retry:
        err = rd_kafka_producev(
                /* Producer handle */
                rk,
                /* Topic name */
                RD_KAFKA_V_TOPIC(topic),
                /* Make a copy of the payload. */
                RD_KAFKA_V_MSGFLAGS(RD_KAFKA_MSG_F_COPY),
                /* Message value and length */
                RD_KAFKA_V_VALUE(buf, len),
                /* Per-Message opaque, provided in
                 * delivery report callback as
                 * msg_opaque. */
                RD_KAFKA_V_OPAQUE(NULL),
                /* End sentinel */
                RD_KAFKA_V_END);

        if (err) {
            /*
             * Failed to *enqueue* message for producing.
             */
            fprintf(stderr,
                    "%% Failed to produce to topic %s: %s\\n",
                    topic, rd_kafka_err2str(err));

            if (err == RD_KAFKA_RESP_ERR__QUEUE_FULL) {
                /* If the internal queue is full, wait for
                 * messages to be delivered and then retry.
```

```
                 * The internal queue represents both
                 * messages to be sent and messages that have
                 * been sent or failed, awaiting their
                 * delivery report callback to be called.
                 *
                 * The internal queue is limited by the
                 * configuration property
                 * queue.buffering.max.messages */
                rd_kafka_poll(rk, 1000/*block for max 1000ms*/);
                goto retry;
            }
        } else {
            fprintf(stderr, "%% Enqueued message (%zd bytes) "
                            "for topic %s\\n",
                    len, topic);
        }


        /* A producer application should continually serve
         * the delivery report queue by calling rd_kafka_poll()
         * at frequent intervals.
         * Either put the poll call in your main loop, or in a
         * dedicated thread, or call it after every
         * rd_kafka_produce() call.
         * Just make sure that rd_kafka_poll() is still called
         * during periods where you are not producing any messages
         * to make sure previously produced messages have their
         * delivery report callback served (and any other callbacks
         * you register). */
        rd_kafka_poll(rk, 0/*non-blocking*/);
    }


    /* Wait for final messages to be delivered or fail.
     * rd_kafka_flush() is an abstraction over rd_kafka_poll() which
     * waits for all messages to be delivered. */
    fprintf(stderr, "%% Flushing final messages..\\n");
    rd_kafka_flush(rk, 10*1000 /* wait for max 10 seconds */);

    /* If the output queue is still not empty there is an issue
     * with producing messages to the clusters. */
    if (rd_kafka_outq_len(rk) > 0)
        fprintf(stderr, "%% %d message(s) were not delivered\\n",
                rd_kafka_outq_len(rk));

    /* Destroy the producer instance */
    rd_kafka_destroy(rk);
```

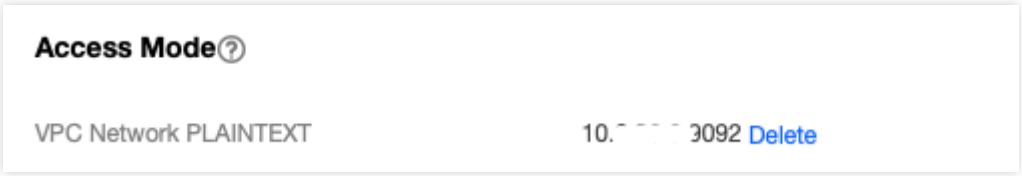
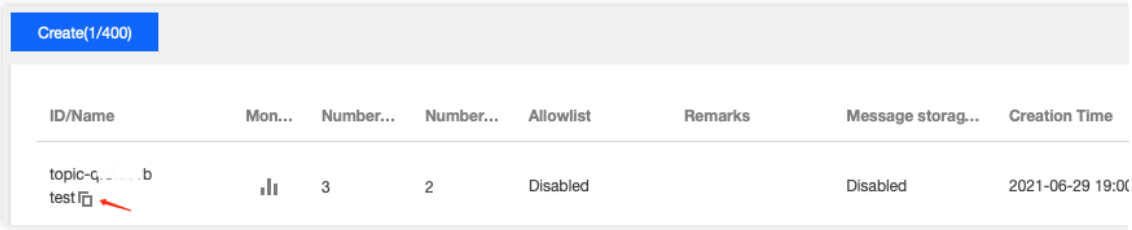
```
    return 0;
}
```

2. Run the following command to compile `producer.c` .

```
gcc -lrdkafka ./producer.c -o producer
```

3. Run the following command to send the message.

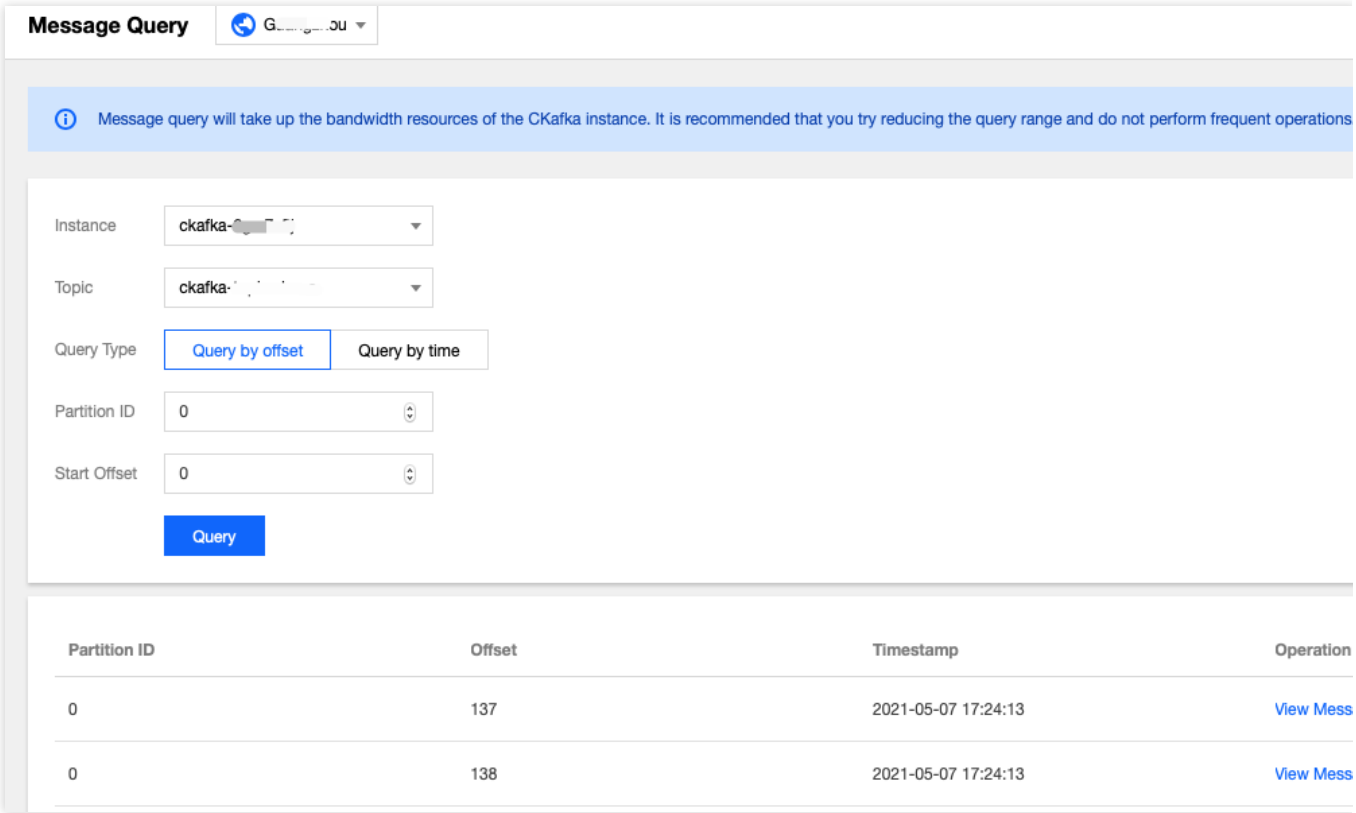
```
./produce <broker> <topic>
```

Parameter	Description
broker	Accessed network, which can be copied from the **Network** column in the **Access Mode** section on the console. 
topic	Topic name, which can be copied from the **Topic Management** page in the console. 

The execution result is as follows:

```
[root@VM_1_28_centos ~]# ./produce [redacted]:7779
% Type some text and hit enter to produce message
% Or just hit enter to only serve delivery reports
% Press Ctrl-C or Ctrl-D to exit
test
% Enqueued message (4 bytes) for topic test
test123
% Enqueued message (7 bytes) for topic test
% Message delivered (4 bytes, partition 1)
^C% Flushing final messages..
% Message delivered (7 bytes, partition 1)
```

4. On the **Topic Management** page in the CKafka console, select the corresponding topic and click **More** > **Message Query** to view the just sent message.



Step 3. Consume the message

1. Create the `consumer.c` file.


```
/*
 * librdkafka - Apache Kafka C library
 *
 * Copyright (c) 2019, Magnus Edenhill
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

/**
 * Simple high-level balanced Apache Kafka consumer
 * using the Kafka driver from librdkafka
 * (https://github.com/edenhill/librdkafka)
 */

#include <stdio.h>
#include <signal.h>
#include <string.h>
#include <ctype.h>


#include <librdkafka/rdkafka.h>


static volatile sig_atomic_t run = 1;

/**
 * @brief Signal termination of program
```

```
 */
static void stop (int sig) {
    run = 0;
}



/**
 * @returns 1 if all bytes are printable, else 0.
 */
static int is_printable (const char *buf, size_t size) {
    size_t i;

    for (i = 0 ; i < size ; i++)
        if (!isprint((int)buf[i]))
            return 0;

    return 1;
}


int main (int argc, char **argv) {
    rd_kafka_t *rk;          /* Consumer instance handle */
    rd_kafka_conf_t *conf;   /* Temporary configuration object */
    rd_kafka_resp_err_t err; /* librdkafka API error code */
    char errstr[512];        /* librdkafka API error reporting buffer */
    const char *brokers;     /* Argument: broker list */
    const char *groupid;     /* Argument: Consumer group id */
    char **topics;           /* Argument: list of topics to subscribe to */
    int topic_cnt;           /* Number of topics to subscribe to */
    rd_kafka_topic_partition_list_t *subscription; /* Subscribed topics */
    int i;

    /*
     * Argument validation
     */
    if (argc < 4) {
        fprintf(stderr,
                "%% Usage: "
                "%s <broker> <group.id> <topic1> <topic2>..\\n",
                argv[0]);
        return 1;
    }

    brokers   = argv[1];
    groupid   = argv[2];
    topics    = &argv[3];
```

```
    topic_cnt = argc - 3;


    /*
     * Create Kafka client configuration place-holder
     */
    conf = rd_kafka_conf_new();

    /* Set bootstrap broker(s) as a comma-separated list of
     * host or host:port (default port 9092).
     * librdkafka will use the bootstrap brokers to acquire the full
     * set of brokers from the cluster. */
    if (rd_kafka_conf_set(conf, "bootstrap.servers", brokers,
                          errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK) {
        fprintf(stderr, "%s\\n", errstr);
        rd_kafka_conf_destroy(conf);
        return 1;
    }

    /* Set the consumer group id.
     * All consumers sharing the same group id will join the same
     * group, and the subscribed topic' partitions will be assigned
     * according to the partition.assignment.strategy
     * (consumer config property) to the consumers in the group. */
    if (rd_kafka_conf_set(conf, "group.id", groupid,
                          errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK) {
        fprintf(stderr, "%s\\n", errstr);
        rd_kafka_conf_destroy(conf);
        return 1;
    }

    /* If there is no previously committed offset for a partition
     * the auto.offset.reset strategy will be used to decide where
     * in the partition to start fetching messages.
     * By setting this to earliest the consumer will read all messages
     * in the partition if there was no previously committed offset. */
    if (rd_kafka_conf_set(conf, "auto.offset.reset", "earliest",
                          errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK) {
        fprintf(stderr, "%s\\n", errstr);
        rd_kafka_conf_destroy(conf);
        return 1;
    }
    /* Tencent Cloud recommended configuration parameters
      * https://cloud.tencent.com/document/product/597/30203
      */
    if(rd_kafka_conf_set(conf,"debug","all",errstr,sizeof(errstr))!= RD_KAFKA_CONF_
        fprintf(stderr,"%s\\n",errstr);
```

```
        return 1;
    }

    if(rd_kafka_conf_set(conf,"session.timeout.ms","10000",errstr,sizeof(errstr)) !
        fprintf(stderr,"%s\\n",errstr);
        return 1;
    }
    if(rd_kafka_conf_set(conf,"heartbeat.interval.ms","3000",errstr,sizeof(errstr))
        fprintf(stderr,"%s\\n",errstr);
        return 1;
    }

    /*
     * Create consumer instance.
     *
     * NOTE: rd_kafka_new() takes ownership of the conf object
     *       and the application must not reference it again after
     *       this call.
     */
    rk = rd_kafka_new(RD_KAFKA_CONSUMER, conf, errstr, sizeof(errstr));
    if (!rk) {
        fprintf(stderr,
                "%% Failed to create new consumer: %s\\n", errstr);
        return 1;
    }

    conf = NULL; /* Configuration object is now owned, and freed,
                      * by the rd_kafka_t instance. */


    /* Redirect all messages from per-partition queues to
     * the main queue so that messages can be consumed with one
     * call from all assigned partitions.
     *
     * The alternative is to poll the main queue (for events)
     * and each partition queue separately, which requires setting
     * up a rebalance callback and keeping track of the assignment:
     * but that is more complex and typically not recommended. */
    rd_kafka_poll_set_consumer(rk);


    /* Convert the list of topics to a format suitable for librdkafka */
    subscription = rd_kafka_topic_partition_list_new(topic_cnt);
    for (i = 0 ; i < topic_cnt ; i++)
        rd_kafka_topic_partition_list_add(subscription,
                                          topics[i],
                /* the partition is ignored
```

```
                 * by subscribe() */
                                        RD_KAFKA_PARTITION_UA);

    /* Subscribe to the list of topics */
    err = rd_kafka_subscribe(rk, subscription);
    if (err) {
        fprintf(stderr,
                "%% Failed to subscribe to %d topics: %s\\n",
                subscription->cnt, rd_kafka_err2str(err));
        rd_kafka_topic_partition_list_destroy(subscription);
        rd_kafka_destroy(rk);
        return 1;
    }

    fprintf(stderr,
            "%% Subscribed to %d topic(s), "
            "waiting for rebalance and messages...\\n",
            subscription->cnt);

    rd_kafka_topic_partition_list_destroy(subscription);


    /* Signal handler for clean shutdown */
    signal(SIGINT, stop);

    /* Subscribing to topics will trigger a group rebalance
     * which may take some time to finish, but there is no need
     * for the application to handle this idle period in a special way
     * since a rebalance may happen at any time.
     * Start polling for messages. */

    while (run) {
        rd_kafka_message_t *rkm;

        rkm = rd_kafka_consumer_poll(rk, 100);
        if (!rkm)
            continue; /* Timeout: no message within 100ms,
                               *  try again. This short timeout allows
                               *  checking for `run` at frequent intervals.
                               */

        /* consumer_poll() will return either a proper message
         * or a consumer error (rkm->err is set). */
        if (rkm->err) {
            /* Consumer errors are generally to be considered
             * informational as the consumer will automatically
             * try to recover from all types of errors. */
```

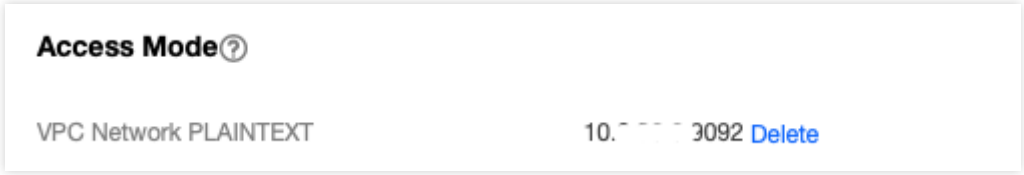
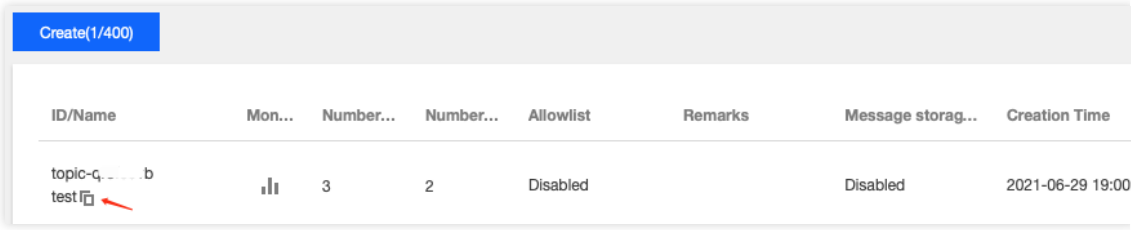
```
            fprintf(stderr,
                    "%% Consumer error: %s\\n",
                    rd_kafka_message_errstr(rkm));
            rd_kafka_message_destroy(rkm);
            continue;
        }

        /* Proper message. */
        printf("Message on %s [%"PRId32"] at offset %"PRId64":\\n",
               rd_kafka_topic_name(rkm->rkt), rkm->partition,
               rkm->offset);

        /* Print the message key. */
        if (rkm->key && is_printable(rkm->key, rkm->key_len))
            printf(" Key: %.*s\\n",
                   (int)rkm->key_len, (const char *)rkm->key);
        else if (rkm->key)
            printf(" Key: (%d bytes)\\n", (int)rkm->key_len);

        /* Print the message value/payload. */
        if (rkm->payload && is_printable(rkm->payload, rkm->len))
            printf(" Value: %.*s\\n",
                   (int)rkm->len, (const char *)rkm->payload);
        else if (rkm->payload)
            printf(" Value: (%d bytes)\\n", (int)rkm->len);

        rd_kafka_message_destroy(rkm);
    }


    /* Close the consumer: commit final offsets and leave the group. */
    fprintf(stderr, "%% Closing consumer\\n");
    rd_kafka_consumer_close(rk);


    /* Destroy the consumer */
    rd_kafka_destroy(rk);

    return 0;
}
```

2. Run the following command to compile `consumer.c` .

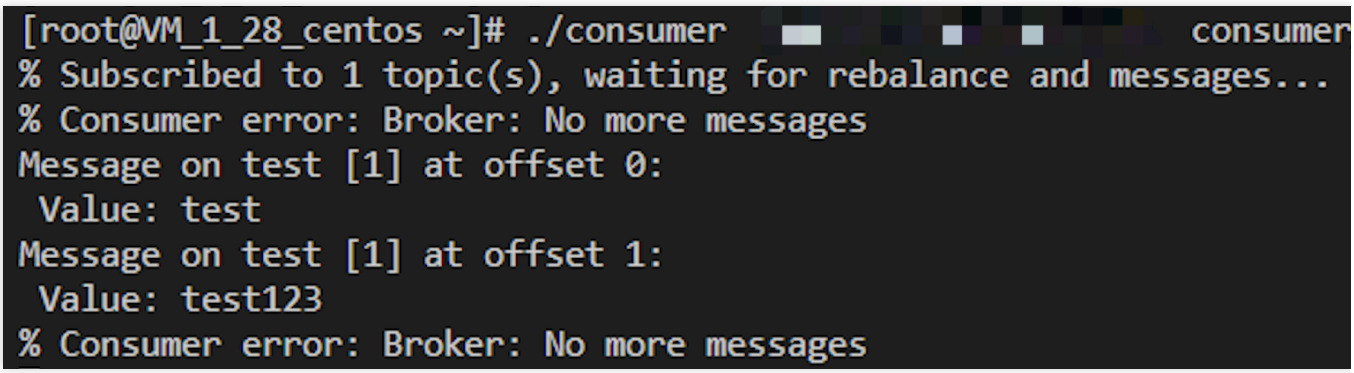
```
gcc -lrdkafka ./consumer.c -o consumer
```

3. Run the following command to send the message.

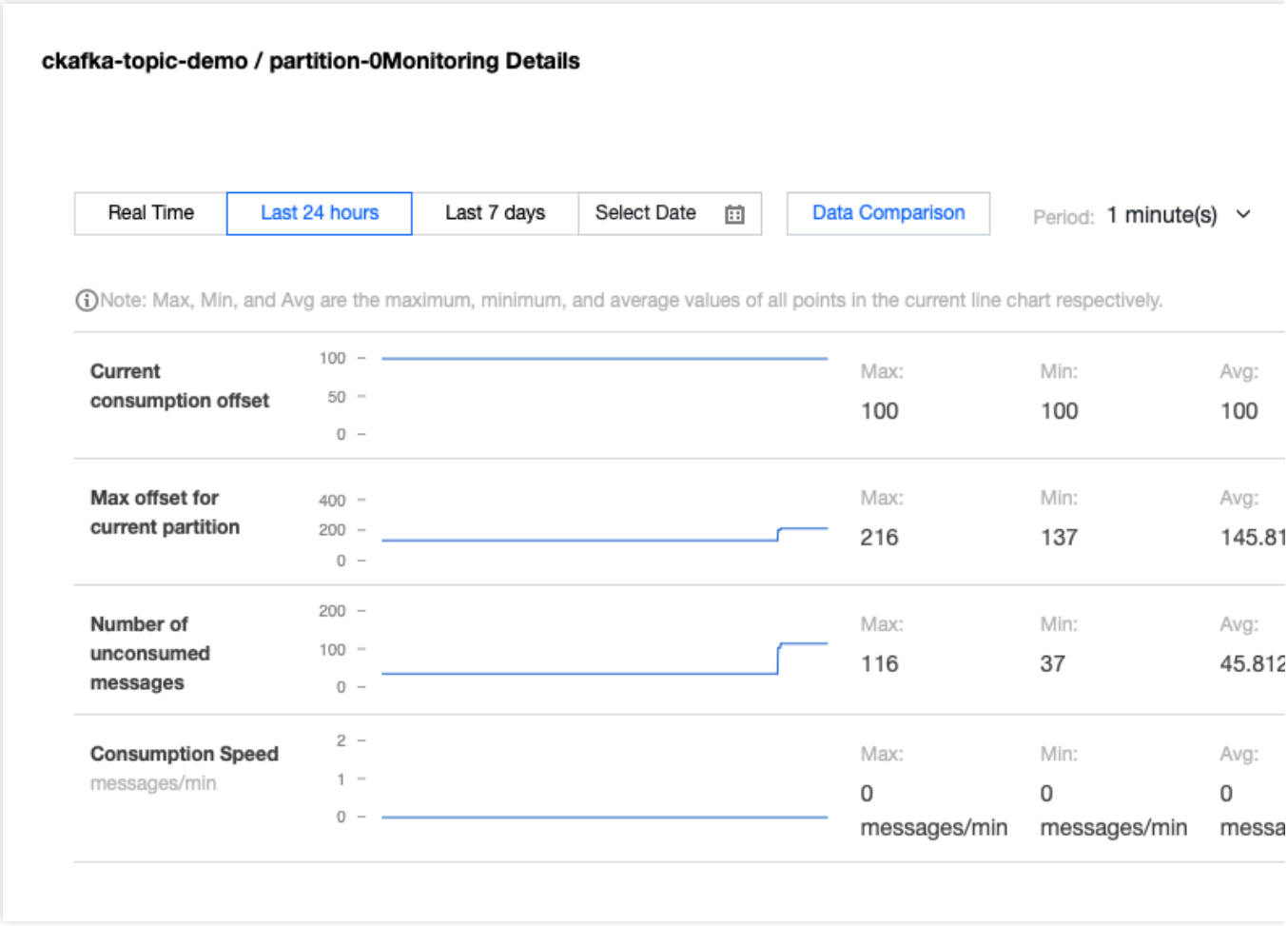
```
./consumer <broker> <group.id> <topic1> <topic2>..
```

Parameter	Description
broker	Accessed network, which can be copied from the **Network** column in the **Access Mode** section on in the console. 
group.id	Consumer group name. You can customize it. After the demo runs successfully, you can see the con **Group** page.
topic1 topic2..	Topic name, which can be copied from the **Topic Management** page in the console. 

The execution result is as follows:



4. On the **Consumer Group** page in the CKafka console, select the corresponding consumer group, enter the topic name in **Topic Name**, and click **Query Details** to view the consumption details.



Public Network Access Through SASL_PLAINTEXT

Last updated：2024-01-09 15:00:32

Overview

This document describes how to access CKafka to send/receive messages with the SDK for C++ through SASL_PLAINTEXT over public network.

Prerequisites

Install GCC
Configure an ACL policy
Download demo

Directions

Step 1. Install the C/C++ dependent library

For more information, please see here.

Step 2. Install SSL/SASL dependencies

```
yum install openssl openssl-devel
yum install cyrus-sasl{,-plain}
```

Step 3. Send messages

1. Create the `producer.c` file.

```
/*
 * librdkafka - Apache Kafka C library
 *
 * Copyright (c) 2017, Magnus Edenhill
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
```

```
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

/**
 * Simple Apache Kafka producer
 * using the Kafka driver from librdkafka
 * (https://github.com/edenhill/librdkafka)
 */

#include <stdio.h>
#include <signal.h>
#include <string.h>


#include <librdkafka/rdkafka.h>


static volatile sig_atomic_t run = 1;

/**
 * @brief Signal termination of program
 */
static void stop (int sig) {
    run = 0;
    fclose(stdin); /* abort fgets() */
}


/**
 * @brief Message delivery report callback.
```

```
 *
 * This callback is called exactly once per message, indicating if
 * the message was successfully delivered
 * (rkmessage->err == RD_KAFKA_RESP_ERR_NO_ERROR) or permanently
 * failed delivery (rkmessage->err != RD_KAFKA_RESP_ERR_NO_ERROR).
 *
 * The callback is triggered from rd_kafka_poll() and executes on
 * the application's thread.
 */
static void dr_msg_cb (rd_kafka_t *rk,
                       const rd_kafka_message_t *rkmessage, void *opaque) {
    if (rkmessage->err)
        fprintf(stderr, "%% Message delivery failed: %s\\n",
                rd_kafka_err2str(rkmessage->err));
    else
        fprintf(stderr,
                "%% Message delivered (%zd bytes, "
                "partition %"PRId32")\\n",
            rkmessage->len, rkmessage->partition);

    /* The rkmessage is destroyed automatically by librdkafka */
}



int main (int argc, char **argv) {
    rd_kafka_t *rk;         /* Producer instance handle */
    rd_kafka_conf_t *conf;  /* Temporary configuration object */
    char errstr[512];       /* librdkafka API error reporting buffer */
    char buf[512];          /* Message value temporary buffer */
    const char *brokers;    /* Argument: broker list */
    const char *topic;      /* Argument: topic to produce to */
    const char *user;      /* Argument: sasl username */
    const char *passwd;     /* Argument: sasl password */

    /*
     * Argument validation
     */
    if (argc < 3) {
        fprintf(stderr, "%% Usage: %s <broker> <topic> <username> <password>  \\n O
        return 1;
    }

    brokers   = argv[1];
    topic   = argv[2];

    if(argc == 5) {
```

```
        user = argv[3];
        passwd = argv[4];
    }


    /*
     * Create Kafka client configuration place-holder
     */
    conf = rd_kafka_conf_new();

    /* Set bootstrap broker(s) as a comma-separated list of
     * host or host:port (default port 9092).
     * librdkafka will use the bootstrap brokers to acquire the full
     * set of brokers from the cluster. */
    if (rd_kafka_conf_set(conf, "bootstrap.servers", brokers,
                          errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK) {
        fprintf(stderr, "%s\\n", errstr);
        return 1;
    }

    /* Set sasl config*/
    if( user && passwd ) {
        if(rd_kafka_conf_set(conf,"security.protocol","sasl_plaintext",errstr,sizeo
            fprintf(stderr,"%s\\n",errstr);
            return 1;
        }
        if(rd_kafka_conf_set(conf, "sasl.mechanisms", "PLAIN", errstr, sizeof(errst
            fprintf(stderr,"%s\\n",errstr);
            return 1;
        }
        if(rd_kafka_conf_set(conf,"sasl.username",user,errstr,sizeof(errstr)) !=RD_
            fprintf(stderr,"%s\\n",errstr);
            return 1;

        }
        if(rd_kafka_conf_set(conf,"sasl.password",passwd,errstr,sizeof(errstr)) !=R
            fprintf(stderr,"%s\\n",errstr);
            return 1;
        }

    }
    /* Tencent Cloud recommended configuration parameters
      * https://cloud.tencent.com/document/product/597/30203
      */
    //if(rd_kafka_conf_set(conf,"debug","none",errstr,sizeof(errstr))!= RD_KAFKA_CO
      //  fprintf(stderr,"%s\\n",errstr);
     //   return 1;
```

```
//  }

  if(rd_kafka_conf_set(conf,"acks","1",errstr,sizeof(errstr)) != RD_KAFKA_CONF_OK
      fprintf(stderr,"%s\\n",errstr);
      return 1;
  }
  if(rd_kafka_conf_set(conf,"request.timeout.ms","30000",errstr,sizeof(errstr)) !
      fprintf(stderr,"%s\\n",errstr);
      return 1;
  }
  if(rd_kafka_conf_set(conf,"retries","3",errstr,sizeof(errstr)) != RD_KAFKA_CONF
      fprintf(stderr,"%s\\n",errstr);
      return 1;
  }

  if(rd_kafka_conf_set(conf,"retry.backoff.ms","1000",errstr,sizeof(errstr)) != R
      fprintf(stderr,"%s\\n",errstr);
      return 1;
  }

  /* Set the delivery report callback.
   * This callback will be called once per message to inform
   * the application if delivery succeeded or failed.
   * See dr_msg_cb() above.
   * The callback is only triggered from rd_kafka_poll() and
   * rd_kafka_flush(). */
  rd_kafka_conf_set_dr_msg_cb(conf, dr_msg_cb);

  /*
   * Create producer instance.
   *
   * NOTE: rd_kafka_new() takes ownership of the conf object
   *       and the application must not reference it again after
   *       this call.
   */
  rk = rd_kafka_new(RD_KAFKA_PRODUCER, conf, errstr, sizeof(errstr));
  if (!rk) {
      fprintf(stderr,
              "%% Failed to create new producer: %s\\n", errstr);
      return 1;
  }

  /* Signal handler for clean shutdown */
  signal(SIGINT, stop);

  fprintf(stderr,
          "%% Type some text and hit enter to produce message\\n"
```

```
            "%% Or just hit enter to only serve delivery reports\\n"
            "%% Press Ctrl-C or Ctrl-D to exit\\n");

    while (run && fgets(buf, sizeof(buf), stdin)) {
        size_t len = strlen(buf);
        rd_kafka_resp_err_t err;

        if (buf[len-1] == '\\n') /* Remove newline */
            buf[--len] = '\\0';

        if (len == 0) {
            /* Empty line: only serve delivery reports */
            rd_kafka_poll(rk, 0/*non-blocking */);
            continue;
        }

        /*
         * Send/Produce message.
         * This is an asynchronous call, on success it will only
         * enqueue the message on the internal producer queue.
         * The actual delivery attempts to the broker are handled
         * by background threads.
         * The previously registered delivery report callback
         * (dr_msg_cb) is used to signal back to the application
         * when the message has been delivered (or failed).
         */
        retry:
        err = rd_kafka_producev(
                /* Producer handle */
                rk,
                /* Topic name */
                RD_KAFKA_V_TOPIC(topic),
                /* Make a copy of the payload. */
                RD_KAFKA_V_MSGFLAGS(RD_KAFKA_MSG_F_COPY),
                /* Message value and length */
                RD_KAFKA_V_VALUE(buf, len),
                /* Per-Message opaque, provided in
                 * delivery report callback as
                 * msg_opaque. */
                RD_KAFKA_V_OPAQUE(NULL),
                /* End sentinel */
                RD_KAFKA_V_END);

        if (err) {
            /*
             * Failed to *enqueue* message for producing.
             */
```

```
            fprintf(stderr,
                    "%% Failed to produce to topic %s: %s\\n",
                    topic, rd_kafka_err2str(err));

            if (err == RD_KAFKA_RESP_ERR__QUEUE_FULL) {
                /* If the internal queue is full, wait for
                 * messages to be delivered and then retry.
                 * The internal queue represents both
                 * messages to be sent and messages that have
                 * been sent or failed, awaiting their
                 * delivery report callback to be called.
                 *
                 * The internal queue is limited by the
                 * configuration property
                 * queue.buffering.max.messages */
                rd_kafka_poll(rk, 1000/*block for max 1000ms*/);
                goto retry;
            }
        } else {
            fprintf(stderr, "%% Enqueued message (%zd bytes) "
                            "for topic %s\\n",
                    len, topic);
        }


        /* A producer application should continually serve
         * the delivery report queue by calling rd_kafka_poll()
         * at frequent intervals.
         * Either put the poll call in your main loop, or in a
         * dedicated thread, or call it after every
         * rd_kafka_produce() call.
         * Just make sure that rd_kafka_poll() is still called
         * during periods where you are not producing any messages
         * to make sure previously produced messages have their
         * delivery report callback served (and any other callbacks
         * you register). */
        rd_kafka_poll(rk, 0/*non-blocking*/);
    }


    /* Wait for final messages to be delivered or fail.
     * rd_kafka_flush() is an abstraction over rd_kafka_poll() which
     * waits for all messages to be delivered. */
    fprintf(stderr, "%% Flushing final messages..\\n");
    rd_kafka_flush(rk, 10*1000 /* wait for max 10 seconds */);

    /* If the output queue is still not empty there is an issue
```

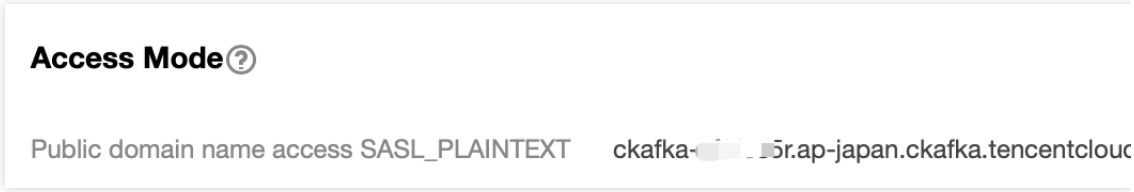
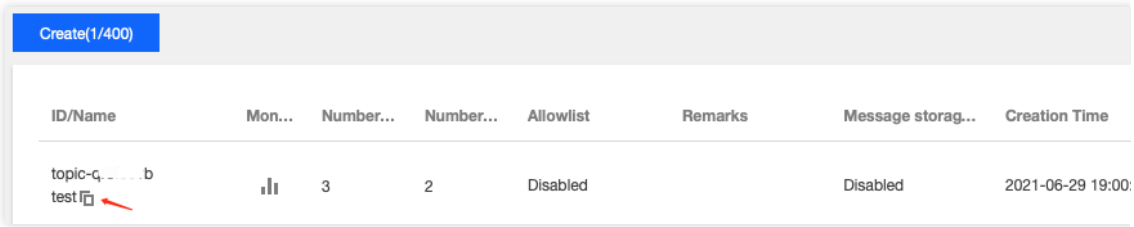
```
     * with producing messages to the clusters. */
    if (rd_kafka_outq_len(rk) > 0)
        fprintf(stderr, "%% %d message(s) were not delivered\\n",
                rd_kafka_outq_len(rk));

    /* Destroy the producer instance */
    rd_kafka_destroy(rk);

    return 0;
}
```

2. Run the command below to compile `producer.c` .

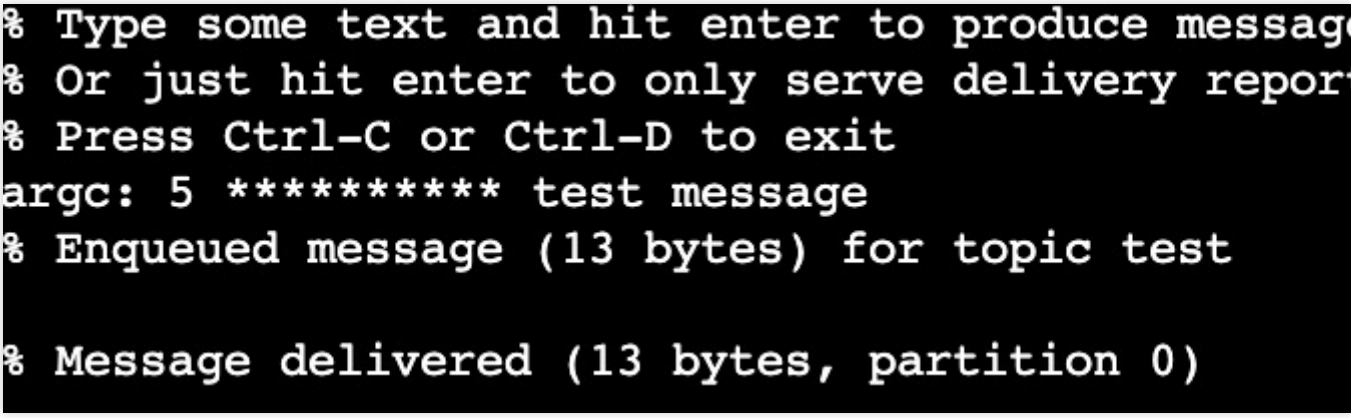
```
gcc -lrdkafka ./producer.c -o producer
```

3. Run the command below to send messages.

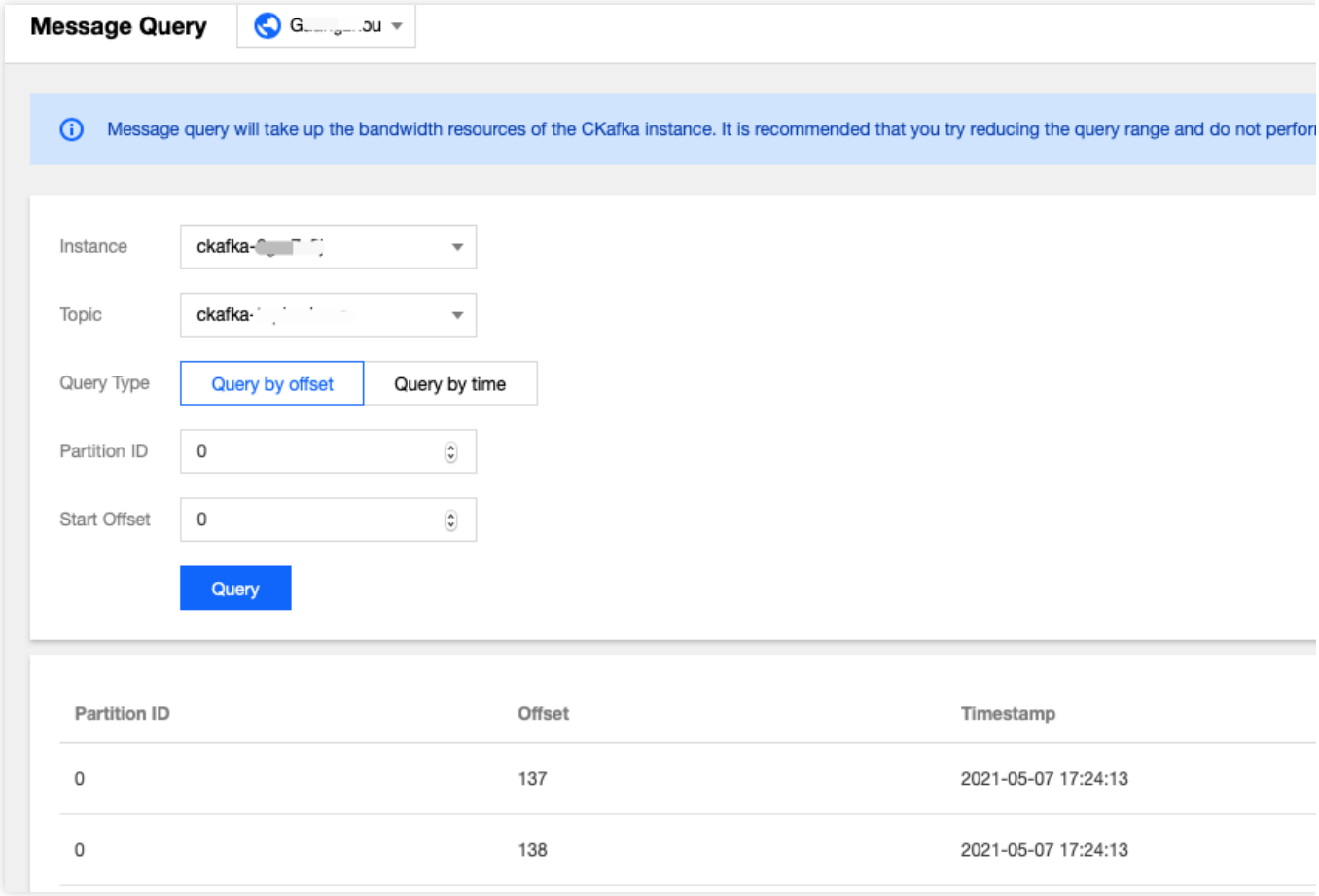
```
./produce <broker> <topic> <username> <password>
```

Parameter	Description
broker	Accessed network, which can be copied from the **Network** column in the **Access Mode** section in **I** details page in the console. 
topic	Topic name, which can be copied in **Topic Management** on the instance details page in the consol 
username	Username in the format of `instance ID` + `#` + `configured username` . The instance ID **Info** on the instance details page in the CKafka console, and the username is set when the user is cr **Management** tab page in the console.
password	Configured password, which is set when the user is created on the **User Management** tab page in t

The execution result is as follows:



4. On the **Topic Management** tab page on the instance details page in the CKafka console, select the target topic, and click **More** > **Message Query** to view the message just sent.



Step 4. Consume messages

1. Create the `consumer.c` file.

```
/*
 * librdkafka - Apache Kafka C library
 *
 * Copyright (c) 2019, Magnus Edenhill
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
*/

/**
 * Simple high-level balanced Apache Kafka consumer
 * using the Kafka driver from librdkafka
 * (https://github.com/edenhill/librdkafka)
 */

#include <stdio.h>
#include <signal.h>
#include <string.h>
#include <ctype.h>
#include <librdkafka/rdkafka.h>


static volatile sig_atomic_t run = 1;
```

```
/**
* @brief Signal termination of program
*/
static void stop (int sig) {
    run = 0;
}

/**
 * @returns 1 if all bytes are printable, else 0.
*/
static int is_printable (const char *buf, size_t size) {
    size_t i;

    for (i = 0 ; i < size ; i++)
            if (!isprint((int)buf[i]))
                    return 0;

    return 1;
}


int main (int argc, char **argv) {
    rd_kafka_t *rk;          /* Consumer instance handle */
    rd_kafka_conf_t *conf;   /* Temporary configuration object */
    rd_kafka_resp_err_t err; /* librdkafka API error code */
    char errstr[512];        /* librdkafka API error reporting buffer */
    const char * user;  /*Argument: sasl username*/
    const char * password;   /*Argument: sasl password*/
    const char *brokers;     /* Argument: broker list */
    const char *groupid;     /* Argument: Consumer group id */
    char **topics;           /* Argument: list of topics to subscribe to */
    int topic_cnt;           /* Number of topics to subscribe to */
    rd_kafka_topic_partition_list_t *subscription; /* Subscribed topics */
    int i;

    /*
    * Argument validation
    */
    if (argc < 6) {
        fprintf(stderr,
                "%% Usage: "
                "%s <broker> <group.id> <username> <password> <topic1> <topic2>..\\
                argv[0]);
        return 1;
    }

    brokers   = argv[1];
```

```
    groupid   = argv[2];
    user      = argv[3];
    password  = argv[4];
    topics    = &argv[5];
    topic_cnt = argc - 5;


    /*
    * Create Kafka client configuration place-holder
    */
    conf = rd_kafka_conf_new();

    /* Set bootstrap broker(s) as a comma-separated list of
    * host or host:port (default port 9092).
    * librdkafka will use the bootstrap brokers to acquire the full
    * set of brokers from the cluster. */
    if (rd_kafka_conf_set(conf, "bootstrap.servers", brokers,
            errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK) {
        fprintf(stderr, "%s\\n", errstr);
        rd_kafka_conf_destroy(conf);
        return 1;
    }

    /* Set the consumer group id.
    * All consumers sharing the same group id will join the same
    * group, and the subscribed topic' partitions will be assigned
    * according to the partition.assignment.strategy
    * (consumer config property) to the consumers in the group. */
    if (rd_kafka_conf_set(conf, "group.id", groupid,
                              errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK) {
        fprintf(stderr, "%s\\n", errstr);
        rd_kafka_conf_destroy(conf);
        return 1;
    }

    /* If there is no previously committed offset for a partition
    * the auto.offset.reset strategy will be used to decide where
    * in the partition to start fetching messages.
    * By setting this to earliest the consumer will read all messages
    * in the partition if there was no previously committed offset. */
    if (rd_kafka_conf_set(conf, "auto.offset.reset", "earliest",
                              errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK) {
        fprintf(stderr, "%s\\n", errstr);
        rd_kafka_conf_destroy(conf);
        return 1;
    }
    /* Tencent Cloud recommended configuration parameters
```

```
    * https://cloud.tencent.com/document/product/597/30203
    */
    if(rd_kafka_conf_set(conf,"debug","all",errstr,sizeof(errstr))!= RD_KAFKA_CONF_
        fprintf(stderr,"%s\\n",errstr);
        return 1;
    }

    if(rd_kafka_conf_set(conf,"session.timeout.ms","10000",errstr,sizeof(errstr)) !
        fprintf(stderr,"%s\\n",errstr);
        return 1;
    }
    if(rd_kafka_conf_set(conf,"heartbeat.interval.ms","3000",errstr,sizeof(errstr))
        fprintf(stderr,"%s\\n",errstr);
        return 1;
    }

    /*Set sasl config*/
    if(rd_kafka_conf_set(conf,"security.protocol","sasl_plaintext",errstr,sizeof(er
        fprintf(stderr,"%s\\n",errstr);
        return 1;
    }
    if(rd_kafka_conf_set(conf, "sasl.mechanisms", "PLAIN", errstr, sizeof(errstr))
        fprintf(stderr,"%s\\n",errstr);
        return 1;
    }
    if(rd_kafka_conf_set(conf,"sasl.username",user,errstr,sizeof(errstr)) !=RD_KAFK
        fprintf(stderr,"%s\\n",errstr);
        return 1;
    }
    if(rd_kafka_conf_set(conf,"sasl.password",password,errstr,sizeof(errstr)) !=RD_
        fprintf(stderr,"%s\\n",errstr);
        return 1;
    }
    /*
    * Create consumer instance.
    *
    * NOTE: rd_kafka_new() takes ownership of the conf object
    *       and the application must not reference it again after
    *       this call.
    */
    rk = rd_kafka_new(RD_KAFKA_CONSUMER, conf, errstr, sizeof(errstr));
    if (!rk) {
        fprintf(stderr,
                "%% Failed to create new consumer: %s\\n", errstr);
        return 1;
    }
```

```
    conf = NULL; /* Configuration object is now owned, and freed,
                  * by the rd_kafka_t instance. */


    /* Redirect all messages from per-partition queues to
    * the main queue so that messages can be consumed with one
    * call from all assigned partitions.
    *
    * The alternative is to poll the main queue (for events)
    * and each partition queue separately, which requires setting
    * up a rebalance callback and keeping track of the assignment:
    * but that is more complex and typically not recommended. */
    rd_kafka_poll_set_consumer(rk);


    /* Convert the list of topics to a format suitable for librdkafka */
    subscription = rd_kafka_topic_partition_list_new(topic_cnt);
    for (i = 0 ; i < topic_cnt ; i++)
        rd_kafka_topic_partition_list_add(subscription,
                                          topics[i],
                                          /* the partition is ignored
                                           * by subscribe() */
                                          RD_KAFKA_PARTITION_UA);

    /* Subscribe to the list of topics */
    err = rd_kafka_subscribe(rk, subscription);
    if (err) {
        fprintf(stderr,
                "%% Failed to subscribe to %d topics: %s\\n",
                subscription->cnt, rd_kafka_err2str(err));
        rd_kafka_topic_partition_list_destroy(subscription);
        rd_kafka_destroy(rk);
        return 1;
    }

    fprintf(stderr,
                "%% Subscribed to %d topic(s), "
                "waiting for rebalance and messages...\\n",
                subscription->cnt);

    rd_kafka_topic_partition_list_destroy(subscription);


    /* Signal handler for clean shutdown */
    signal(SIGINT, stop);

    /* Subscribing to topics will trigger a group rebalance
```

```
     * which may take some time to finish, but there is no need
     * for the application to handle this idle period in a special way
     * since a rebalance may happen at any time.
     * Start polling for messages. */

     while (run) {
         rd_kafka_message_t *rkm;

         rkm = rd_kafka_consumer_poll(rk, 100);
         if (!rkm)
             continue; /* Timeout: no message within 100ms,
             *  try again. This short timeout allows
             *  checking for `run` at frequent intervals.
             */

             /* consumer_poll() will return either a proper message
             * or a consumer error (rkm->err is set). */
             if (rkm->err) {
                 /* Consumer errors are generally to be considered
                 * informational as the consumer will automatically
                 * try to recover from all types of errors. */
                 fprintf(stderr,
                         "%% Consumer error: %s\\n",
                         rd_kafka_message_errstr(rkm));
                 rd_kafka_message_destroy(rkm);
                 continue;
             }

             /* Proper message. */
             printf("Message on %s [%"PRId32"] at offset %"PRId64":\\n",
                     rd_kafka_topic_name(rkm->rkt), rkm->partition,
                     rkm->offset);

             /* Print the message key. */
             if (rkm->key && is_printable(rkm->key, rkm->key_len))
                 printf(" Key: %.*s\\n",
                         (int)rkm->key_len, (const char *)rkm->key);
             else if (rkm->key)
                 printf(" Key: (%d bytes)\\n", (int)rkm->key_len);

            /* Print the message value/payload. */
            if (rkm->payload && is_printable(rkm->payload, rkm->len))
                     printf(" Value: %.*s\\n",
                             (int)rkm->len, (const char *)rkm->payload);
            else if (rkm->payload)
                printf(" Value: (%d bytes)\\n", (int)rkm->len);
```

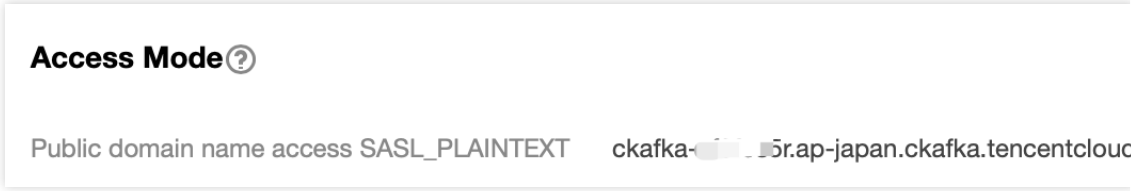
```
            rd_kafka_message_destroy(rkm);
    }


    /* Close the consumer: commit final offsets and leave the group. */
    fprintf(stderr, "%% Closing consumer\\n");
    rd_kafka_consumer_close(rk);


    /* Destroy the consumer */
    rd_kafka_destroy(rk);

    return 0;
}
```

2. Run the command below to compile `consumer.c` .

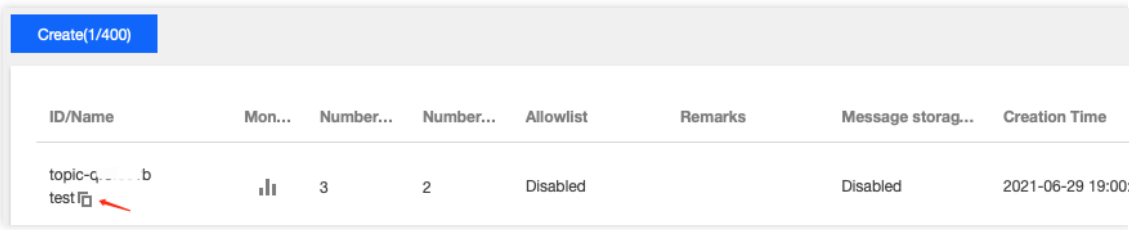
```
gcc -lrdkafka ./consumer.c -o consumer
```

3. Run the command below to send messages.

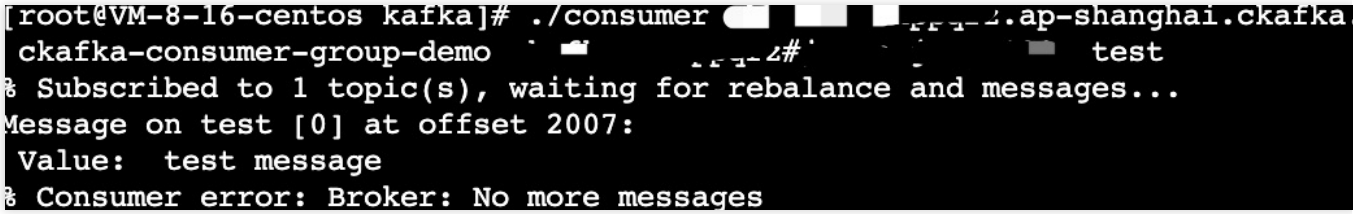
```
./consumer <broker> <group.id> <username> <password> <topic1> <topic2>..
```

Parameter	Description
broker	Accessed network, which can be copied from the **Network** column in the **Access Mode** section in **I** details page in the console. 
group.id	Consumer group name. You can customize it. After the demo runs successfully, you can see the con **Group** page.
username	Username in the format of `instance ID` + `#` + `configured username` . The instance ID **Info** on the instance details page in the CKafka console, and the username is set when the user is cr **Management** tab page in the console.
password	Configured password, which is set when the user is created on the **User Management** page in the
topic1	Topic name, which can be copied in **Topic Management** on the instance details page in the consol

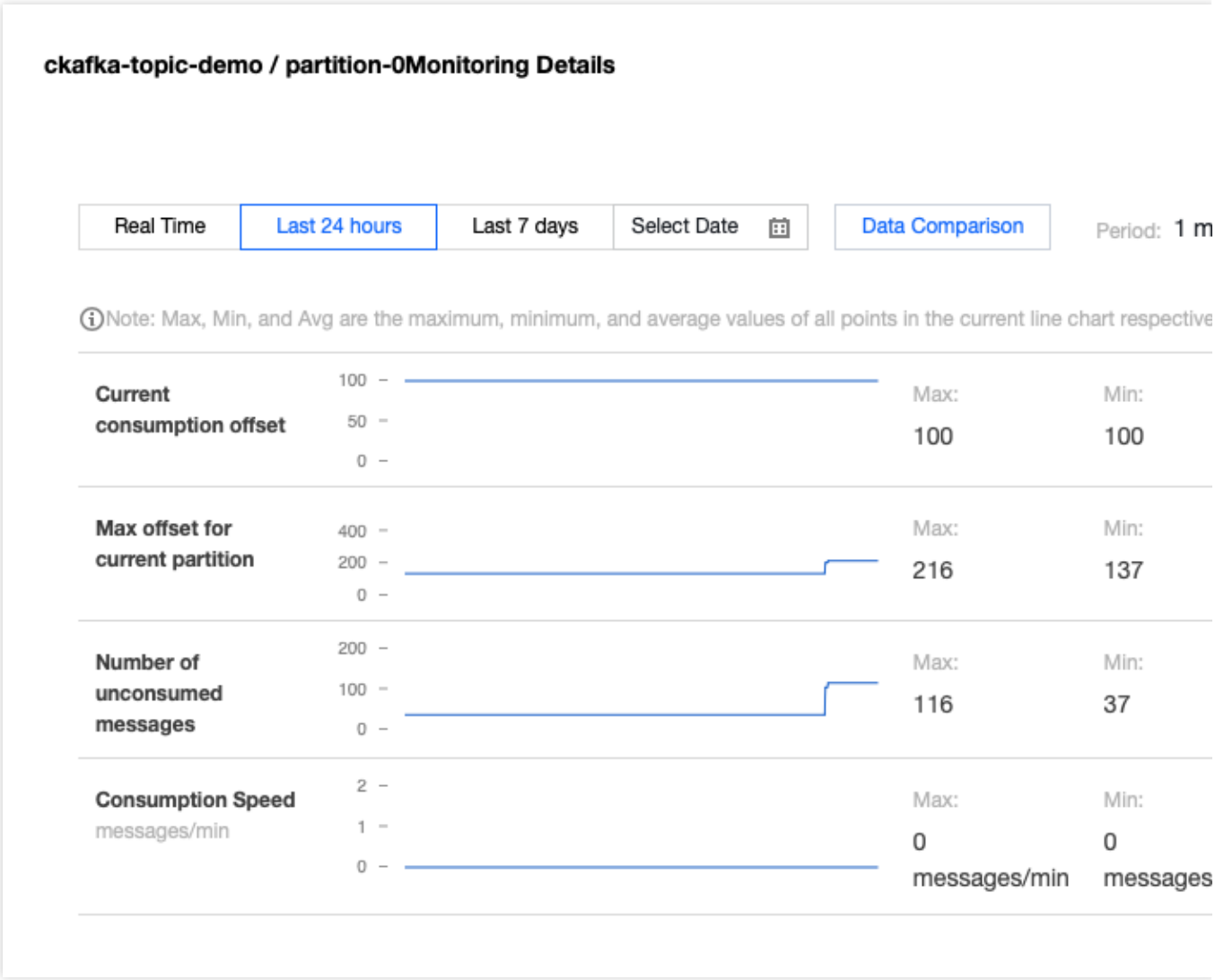
topic2..



The execution result is as follows:



4. On the **Consumer Group** tab page on the instance details page in the CKafka console, select the corresponding consumer group name, enter the topic name, and click **Query Details** to view the consumption details.



SDK for Node.js VPC Access

Last updated：2024-01-09 15:00:32

Overview

This document describes how to access CKafka to send/receive messages with the SDK for Node.js in a VPC.

Prerequisites

You have installed GCC.
You have installed Node.js.
You have downloaded the demo.

Directions

Preparations

1. Upload the `nodejskafkademo` in the downloaded demo to the Linux server.
2. Log in to the Linux server and enter the `nodejskafkademo` directory.

Step 1. Install the C++ dependent library

1. Run the following command to switch to the `yum` source configuration directory `/etc/yum.repos.d/` .

```
cd /etc/yum.repos.d/
```

2. Create the `yum` source configuration file `confluent.repo` .

```
[Confluent.dist]
name=Confluent repository (dist)
baseurl=https://packages.confluent.io/rpm/5.1/7
gpgcheck=1
gpgkey=https://packages.confluent.io/rpm/5.1/archive.key
enabled=1
[Confluent]
name=Confluent repository
baseurl=https://packages.confluent.io/rpm/5.1
gpgcheck=1
```

```
gpgkey=https://packages.confluent.io/rpm/5.1/archive.key
enabled=1
```

3. Run the following command to install the C++ dependent library.

```
yum install librdkafka-devel
```

Step 2. Install the Node.js dependent library

1. Run the following command to specify the OpenSSL header file path for the preprocessor.

```
export CPPFLAGS=-I/usr/local/opt/openssl/include
```

2. Run the following command to specify the OpenSSL library path for the connector.

```
export LDFLAGS=-L/usr/local/opt/openssl/lib
```

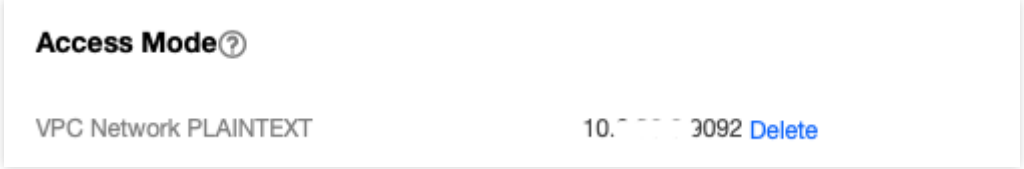
3. Run the following command to install the Node.js dependent library.

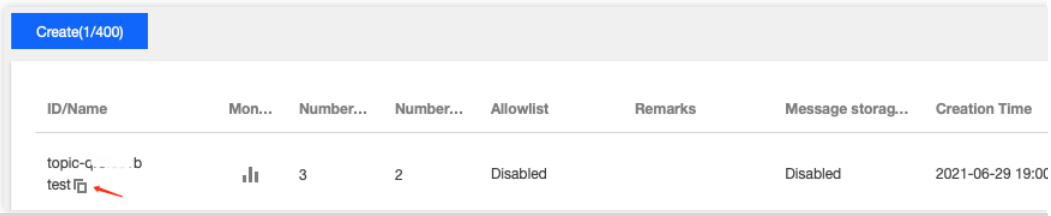
```
npm install i --unsafe-perm node-rdkafka
```

Step 3. Prepare configurations

Create the CKafka configuration file `setting.js` .

```
module.exports = {
    'bootstrap_servers': ["xxx.xx.xxx:xxxx"],
    'topic_name': 'xxx',
    'group_id': 'xxx'
}
```

Parameter	Description
bootstrap_servers	Accessed network, which can be copied from the **Network** column in the **Access Mode** sect instance details page in the console. 
topic_name	Topic name, which can be copied on the **Topic Management** page in the console.

	
group_id	You can customize it. After the demo runs successfully, you can see the consumer group on t page.

Step 4. Send messages

1. Write a message production program named `producer.js` .

```
const Kafka = require('node-rdkafka');
    const config = require('./setting');
    console.log("features:" + Kafka.features);
    console.log(Kafka.librdkafkaVersion);

    var producer = new Kafka.Producer({
        'api.version.request': 'true',
        // To set the entry service, obtain the corresponding service address in the
        'bootstrap.servers': config['bootstrap_servers'],
        'dr_cb': true,
        'dr_msg_cb': true,

        // Number of retries upon request error. We recommend that you set the param
        'retries': '0',
        // Interval between a request failure and the next retry
        "retry.backoff.ms": 100,
        // Network request timeout of the producer
        'socket.timeout.ms': 6000,
    });

    var connected = false

    producer.setPollInterval(100);

    producer.connect();

    producer.on('ready', function() {
    connected = true
    console.log("connect ok")
    });

    producer.on("disconnected", function() {
    connected = false;
```

```
producer.connect();
})

producer.on('event.log', function(event) {
    console.log("event.log", event);
});

producer.on("error", function(error) {
    console.log("error:" + error);
});

function produce() {
    try {
        producer.produce(
        config['topic_name'],
        null,
        new Buffer('Hello CKafka Default'),
        null,
        Date.now()
        );
    } catch (err) {
        console.error('Error occurred when sending message(s)');
        console.error(err);
    }

}

producer.on('delivery-report', function(err, report) {
    console.log("delivery-report: producer ok");
});

producer.on('event.error', function(err) {
    console.error('event.error:' + err);
})

setInterval(produce, 1000, "Interval");
```

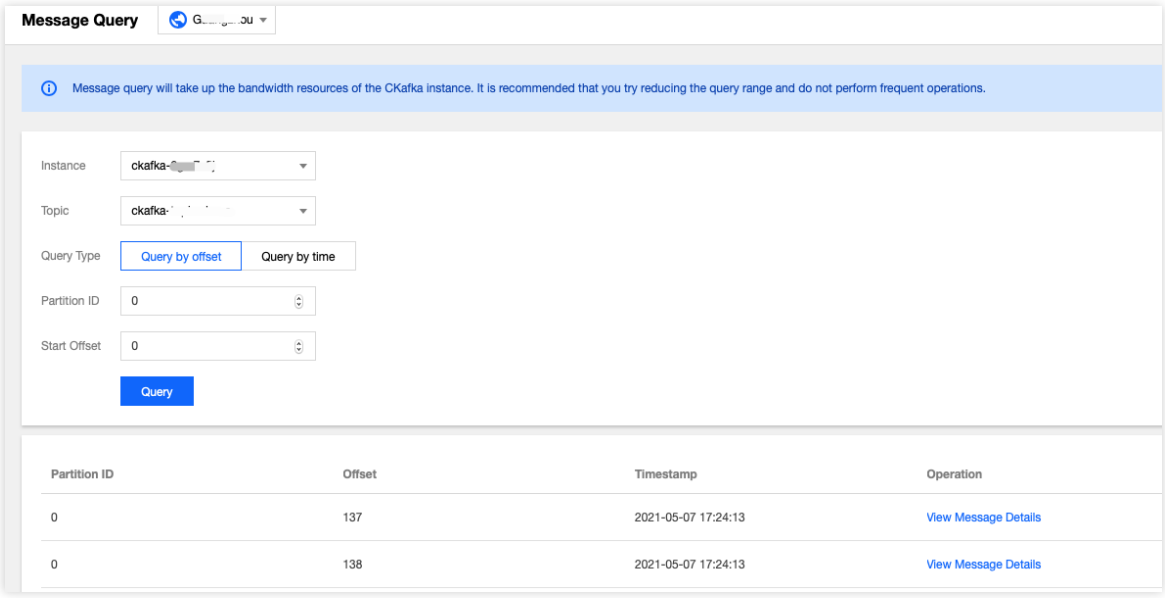
2. Run the following command to send messages.

```
node producer.js
```

3. View the execution result.

```
~/Demos/ckafka-demo/nodejskafkademo/sasl ckafka_demo ± node producer.js
features:gzip,snappy,sasl,regex,lz4
0.9.5
connect ok
(node:59829) [DEP0005] DeprecationWarning: Buffer() is deprecated due to security and usability issues. Please use the Buffer.a
lloc(), Buffer.allocUnsafe(), or Buffer.from() methods instead.
delivery-report: producer ok
delivery-report: producer ok
delivery-report: producer ok
delivery-report: producer ok
delivery-report: producer ok
delivery-report: producer ok
delivery-report: producer ok
delivery-report: producer ok
delivery-report: producer ok
delivery-report: producer ok
delivery-report: producer ok
delivery-report: producer ok
delivery-report: producer ok
delivery-report: producer ok
```

4. On the **Topic Management** tab on the instance details page in the CKafka console, select the target topic and click **More** > **Message Query** to view the message just sent.



Step 5. Subscribe to messages

1. Create the message consumption program `consumer.js` .

```
const Kafka = require('node-rdkafka');
    const config = require('./setting');
    console.log(Kafka.features);
    console.log(Kafka.librdkafkaVersion);
    console.log(config)

    var consumer = new Kafka.KafkaConsumer({
        'api.version.request': 'true',
        // To set the entry service, obtain the corresponding service address in the
```

```
        'bootstrap.servers': config['bootstrap_servers'],
        'group.id' : config['group_id'],

        // Consumer timeout period when the Kafka consumer grouping mechanism is use
        // the consumer will be considered to have failed and the broker will initia
        'session.timeout.ms': 10000,
        // Client request timeout period. If no response is received after this time
        'metadata.request.timeout.ms': 305000,
        // Set the internal retry interval of the client
        'reconnect.backoff.max.ms': 3000

    });

    consumer.connect();

    consumer.on('ready', function() {
    console.log("connect ok");
    consumer.subscribe([config['topic_name']]);
    consumer.consume();
    })

    consumer.on('data', function(data) {
    console.log(data);
    });

    consumer.on('event.log', function(event) {
        console.log("event.log", event);
    });

    consumer.on('error', function(error) {
        console.log("error:" + error);
    });

    consumer.on('event', function(event) {
            console.log("event:" + event);
    });
```

2. Execute the following command to send messages.

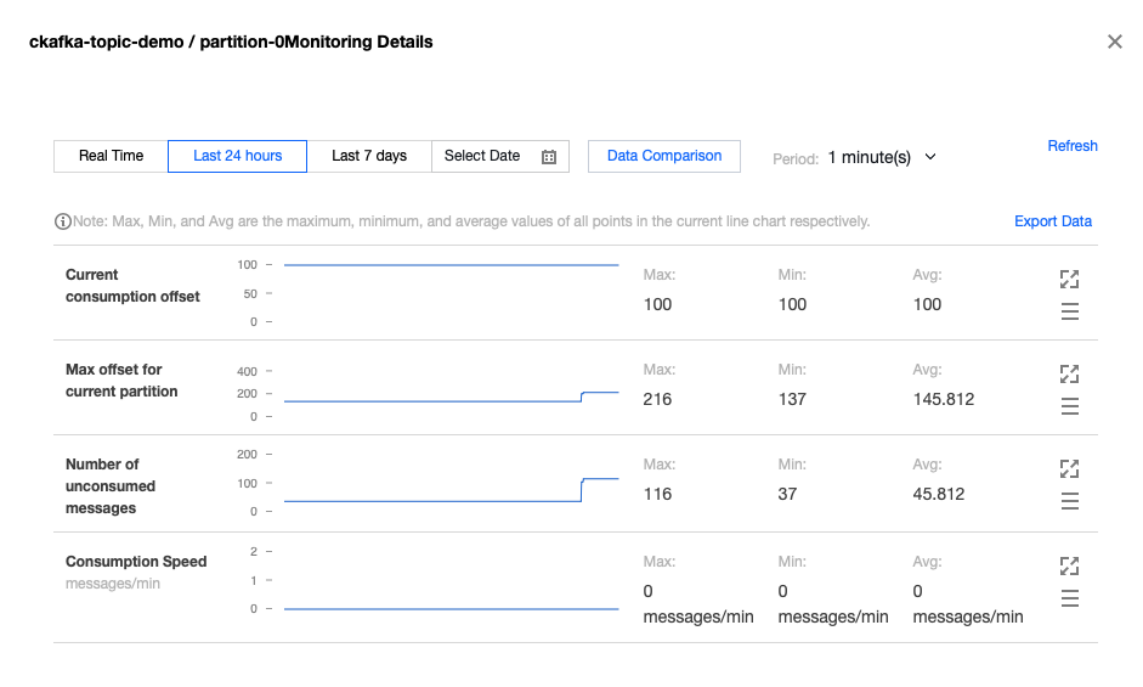
```
node consumer.js
```

3. View the execution result.

```
[redacted] MB2 ~/Demos/ckafka-demo/nodejskafkademo/sasl ckafka_demo ± node consumer.js
[ 'gzip', 'snappy', 'sasl', 'regex', 'lz4' ]
0.9.5
{ sasl_plain_username: 'ckafka-[redacted]#ckafkademo',
  sasl_plain_password: 'ckafkademo123',
  bootstrap_servers:
   [ 'ckafka-[redacted]:ckafka.tencentcloudmq.com:6018' ],
  topic_name: 'ckafka-topic-demo',
  consumer_id: 'nodejs-demo' }
connect ok
{ value: null,
  size: 0,
  key: '1620379451745',
  topic: 'ckafka-topic-demo',
  offset: 137,
  partition: 0 }
{ value: null,
  size: 0,
  key: '1620379452745',
  topic: 'ckafka-topic-demo',
  offset: 138,
  partition: 0 }
```

4. On the **Consumer Group** page in the Ckafka console, click the triangle icon on the left of the target consumer group name, enter the topic name in the search box, and click **View Details** to view the consumption details.

5.



Public Network Access Through SASL_PLAINTEXT

Last updated：2024-01-09 15:00:32

Overview

This document describes how to access CKafka to send/receive messages with the SDK for Node.js through SASL_PLAINTEXT over public network.

Prerequisites

You have installed GCC.
You have installed Node.js.
You have configured an ACL policy.
You have downloaded the demo.

Directions

Step 1. Install the C++ dependent library

1. Run the following command to switch to the `yum` source configuration directory `/etc/yum.repos.d/`.

```
cd /etc/yum.repos.d/
```

2. Create the `yum` source configuration file `confluent.repo`.

```
[Confluent.dist]
name=Confluent repository (dist)
baseurl=https://packages.confluent.io/rpm/5.1/7
gpgcheck=1
gpgkey=https://packages.confluent.io/rpm/5.1/archive.key
enabled=1
[Confluent]
name=Confluent repository
baseurl=https://packages.confluent.io/rpm/5.1
gpgcheck=1
gpgkey=https://packages.confluent.io/rpm/5.1/archive.key
enabled=1
```

3. Run the following command to install the C++ dependent library.

```
yum install librdkafka-devel
```

Step 2. Install the Node.js dependent library

1. Run the following command to specify the OpenSSL header file path for the preprocessor.

```
export CPPFLAGS=-I/usr/local/opt/openssl/include
```

2. Run the following command to specify the OpenSSL library path for the connector.

```
export LDFLAGS=-L/usr/local/opt/openssl/lib
```

3. Run the following command to install the Node.js dependent library.

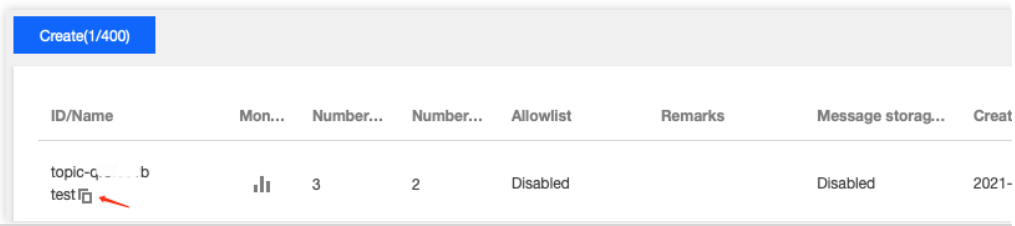
```
npm install i --unsafe-perm node-rdkafka
```

Step 3. Prepare configurations

Create the CKafka configuration file `setting.js` .

```
module.exports = {
    'sasl_plain_username': 'ckafka-xxxxxxx#ckafkademo',
    'sasl_plain_password': 'ckafkademo123',
    'bootstrap_servers': ["xxx.ckafka.tencentcloudmq.com:6018"],
    'topic_name': 'xxx',
    'group_id': 'xxx'
}
```

Parameter	Description
sasl_plain_username	Username in the format of `instance ID` + `#` + `username` . The instance ID can instance details page in the CKafka console, and the username is set when the user is cre > **User Management**.
sasl_plain_password	User password, which is set when the user is created in **ACL Policy Management** > **Use** details page in the CKafka console.
bootstrap_servers	SASL access point. You can obtain it in **Basic Info** >**Access Mode** on the instance detail 
topic_name	Topic name, which can be created and obtained in **Topic Management** on the instance d

	
group_id	Consumer group ID, which can be customized as needed.

Step 4. Send messages

1. Write a message production program named `producer.js` .

```
const Kafka = require('node-rdkafka');
const config = require('./setting');
console.log("features:" + Kafka.features);
console.log(Kafka.librdkafkaVersion);

var producer = new Kafka.Producer({
   'api.version.request': 'true',
   'bootstrap.servers': config['bootstrap_servers'],
   'dr_cb': true,
   'dr_msg_cb': true,
   'security.protocol' : 'SASL_PLAINTEXT',
   'sasl.mechanisms' : 'PLAIN',
   'sasl.username' : config['sasl_plain_username'],
   'sasl.password' : config['sasl_plain_password']
});

var connected = false

producer.setPollInterval(100);

producer.connect();

producer.on('ready', function() {
connected = true
console.log("connect ok")

});

function produce() {
try {
   producer.produce(
   config['topic_name'],
   new Buffer('Hello CKafka SASL'),
   null,
```

```
    Date.now()
    );
} catch (err) {
    console.error('Error occurred when sending message(s)');
    console.error(err);
}
}

producer.on("disconnected", function() {
connected = false;
producer.connect();
})

producer.on('event.log', function(event) {
    console.log("event.log", event);
});

producer.on("error", function(error) {
    console.log("error:" + error);
});

producer.on('delivery-report', function(err, report) {
    console.log("delivery-report: producer ok");
});
// Any errors we encounter, including connection errors
producer.on('event.error', function(err) {
    console.error('event.error:' + err);
})

setInterval(produce,1000,"Interval");
```

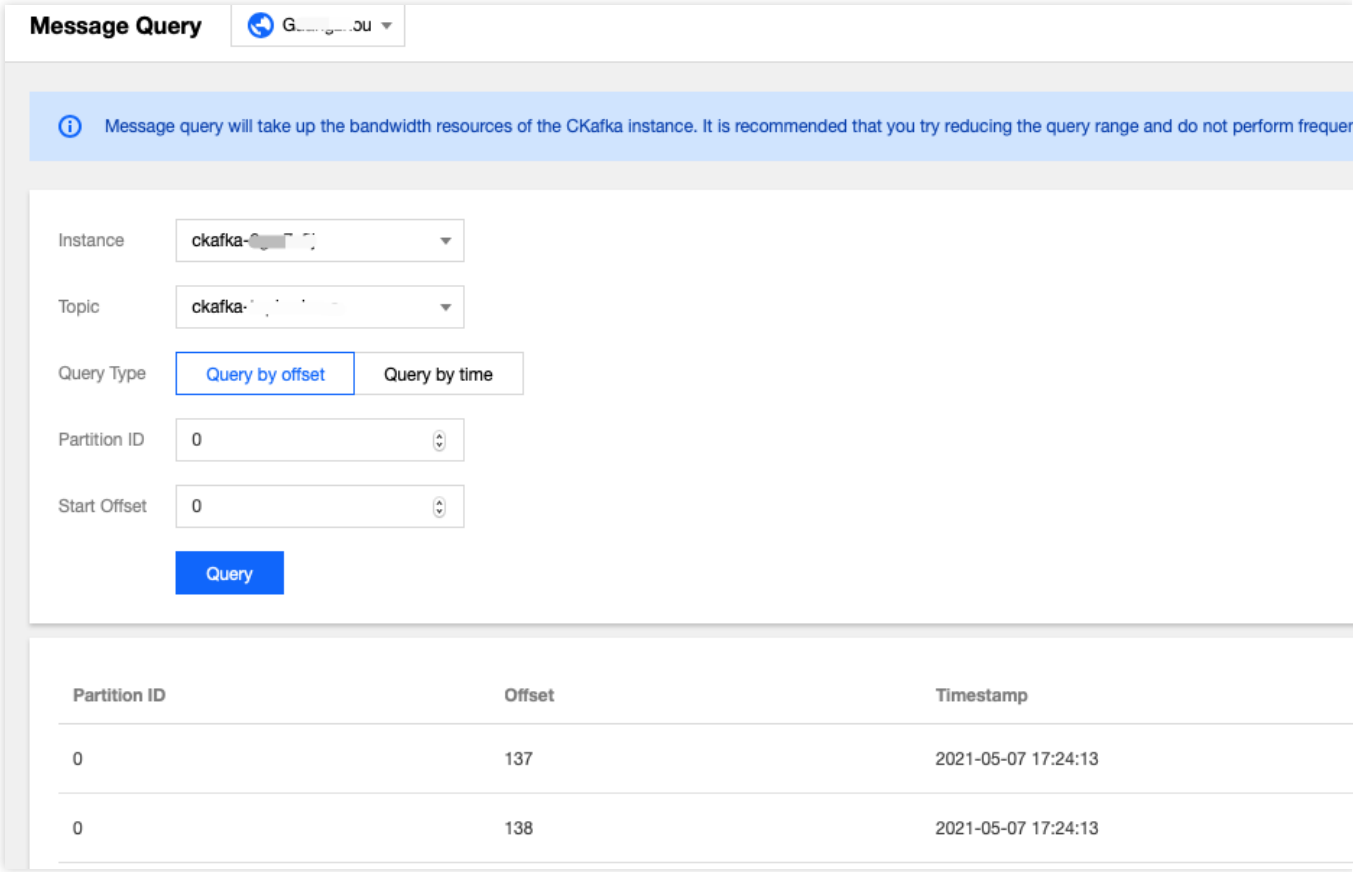
2. Run the following command to send messages.

```
node producer.js
```

3. View the execution result.

```
~/Demos/ckafka-demo/nodejskafkademo/sasl  ckafka_demo ±  node produc
features:gzip,snappy,sasl,regex,lz4
0.9.5
connect ok
(node:59829) [DEP0005] DeprecationWarning: Buffer() is deprecated due to security and usability issue
lloc(), Buffer.allocUnsafe(), or Buffer.from() methods instead.
delivery-report: producer ok
delivery-report: producer ok
delivery-report: producer ok
delivery-report: producer ok
delivery-report: producer ok
delivery-report: producer ok
delivery-report: producer ok
delivery-report: producer ok
delivery-report: producer ok
delivery-report: producer ok
delivery-report: producer ok
delivery-report: producer ok
delivery-report: producer ok
delivery-report: producer ok
```

4. On the **Topic Management** tab on the instance details page in the CKafka console, select the target topic and click **More** > **Message Query** to view the message just sent.



Step 5. Subscribe to messages

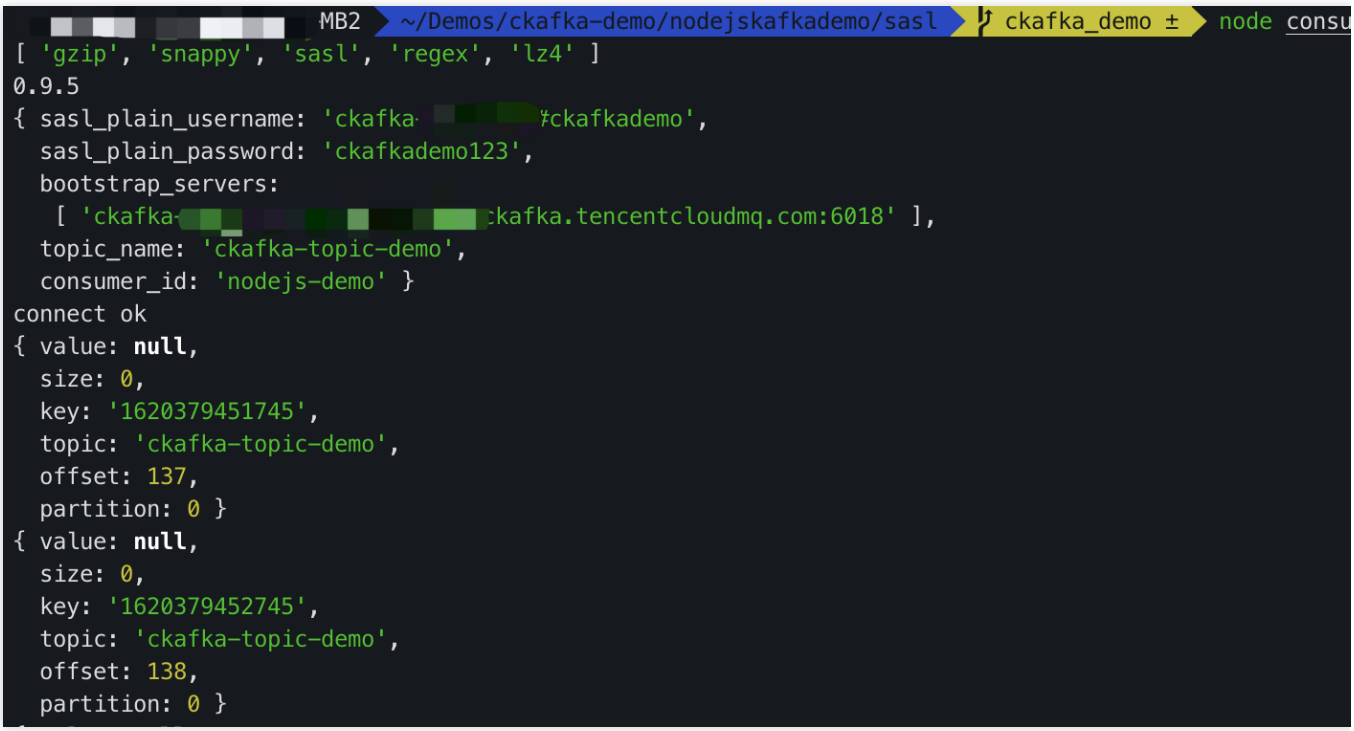
1. Create the message consumption program `consumer.js` .

```
 consumer.on('event.log', function(event) {
    console.log("event.log", event);
});

consumer.on('error', function(error) {
    console.log("error:" + error);
});

consumer.on('event', function(event) {
        console.log("event:" + event);
});const Kafka = require('node-rdkafka');
const config = require('./setting');
console.log(Kafka.features);
console.log(Kafka.librdkafkaVersion);
console.log(config)

var consumer = new Kafka.KafkaConsumer({
    'api.version.request': 'true',
    'bootstrap.servers': config['bootstrap_servers'],
    'security.protocol' : 'SASL_PLAINTEXT',
    'sasl.mechanisms' : 'PLAIN',
    'message.max.bytes': 32000,
    'fetch.message.max.bytes': 32000,
    'max.partition.fetch.bytes': 32000,
    'sasl.username' : config['sasl_plain_username'],
    'sasl.password' : config['sasl_plain_password'],
    'group.id' : config['group_id']
});

consumer.connect();

consumer.on('ready', function() {
console.log("connect ok");
consumer.subscribe([config['topic_name']]);
consumer.consume();
})

consumer.on('data', function(data) {
console.log(data);
});
```

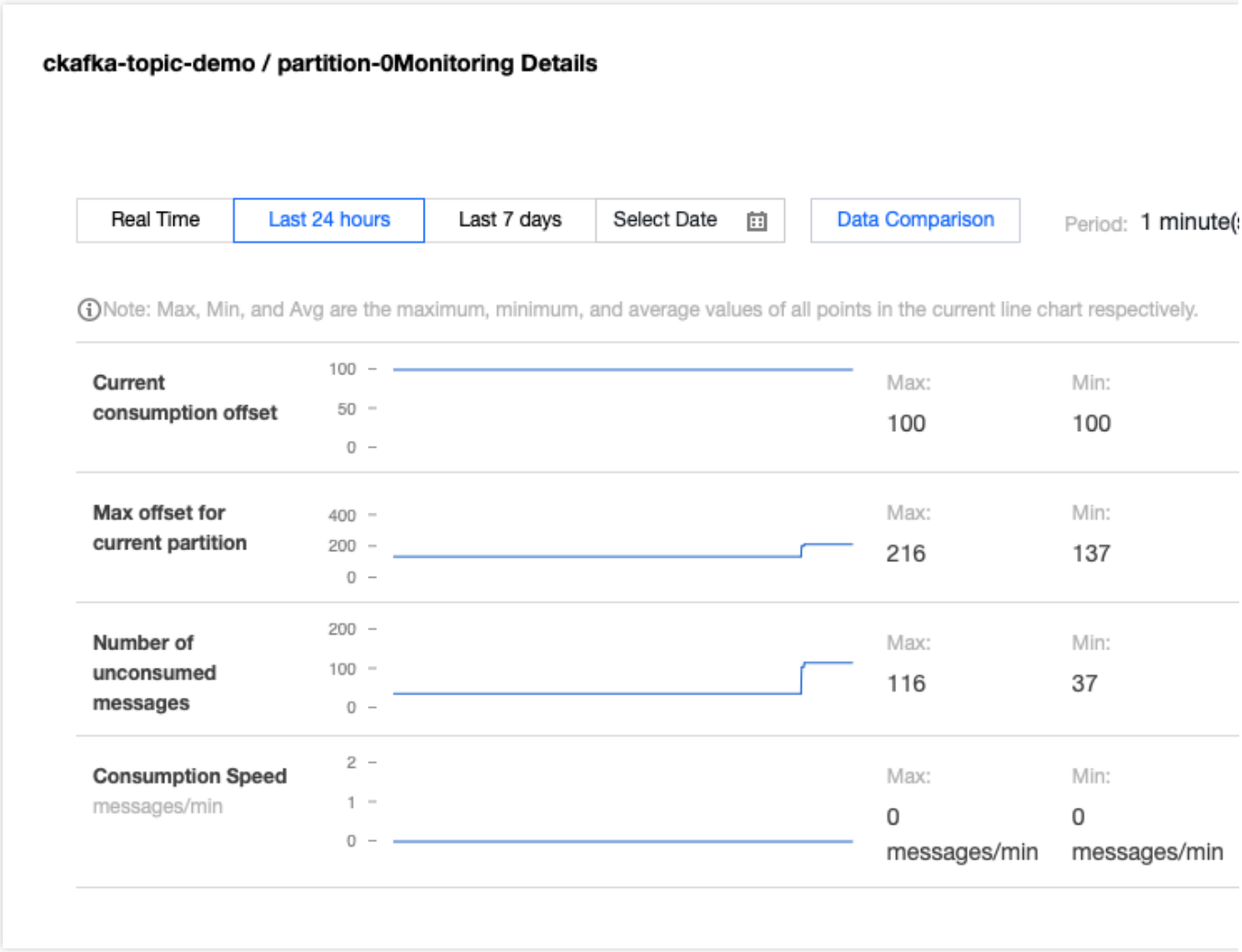
2. Execute the following command to send messages.

```
node consumer.js
```

3. View the execution result.



4. On the **Consumer Group** page in the Ckafka console, click the triangle icon on the left of the target consumer group name, enter the topic name in the search box, and click **View Details** to view the consumption details.



SDK for Connector
Data Reporting SDK

Last updated：2024-01-09 15:00:32

Overview

This document uses Java as an example to describe how to integrate the data reporting SDK for Java with the client to quickly report data to DataHub.

Directions

Step 1. Create an HTTP access point

Create an HTTP access point in the DataHub console as instructed in Reporting over HTTP and get the `DatahubId` identifying the reporting endpoint.

Step 2. Import the SDK for Java

Import the data reporting SDK through Maven or Gradle into the Java project.

Step 3. Report the data

After importing the SDK, you can call the `SendMessage` API of the SDK to report a single data entry or batch report data entries as follows:

1. Instantiate the authentication object.
2. Instantiate the client object.
3. Call `SendMessage` to request to report data.
4. Process the returned result.

```
import com.tencentcloudapi.common.Credential;
import com.tencentcloudapi.common.profile.ClientProfile;
import com.tencentcloudapi.common.profile.HttpProfile;
import com.tencentcloudapi.common.exception.TencentCloudSDKException;
import com.tencentcloudapi.ckafka.v20190819.CkafkaClient;
import com.tencentcloudapi.ckafka.v20190819.models.*;

public class SendMessage
{
    public static void main(String [] args) {
        try{
```

```
            // Instantiate an authentication object. Pass in `secretID` and `secret
            // You can get them at https://console.tencentcloud.com/cam/capi
            Credential cred = new Credential("SecretId", "SecretKey");
            // (Optional) Instantiate an HTTP option
            HttpProfile httpProfile = new HttpProfile();
            httpProfile.setEndpoint("ckafka.tencentcloudapi.com");
            // (Optional) Instantiate a client option
            ClientProfile clientProfile = new ClientProfile();
            clientProfile.setHttpProfile(httpProfile);
            // Instantiate the client object of the requested product. `clientProfi
            CkafkaClient client = new CkafkaClient(cred, "ap-beijing", clientProfil
            // Instantiate a request object. Each API corresponds to a request obje
            SendMessageRequest req = new SendMessageRequest();
            req.setDataHubId("datahub-r6gkngy3");

            BatchContent[] batchContents1 = new BatchContent[2];
            BatchContent batchContent1 = new BatchContent();
            batchContent1.setBody("test1");
            batchContents1[0] = batchContent1;

            BatchContent batchContent2 = new BatchContent();
            batchContent2.setBody("test2");
            batchContents1[1] = batchContent2;

            req.setMessage(batchContents1);

            // The returned `resp` is an instance of `SendMessageResponse` which co
            SendMessageResponse resp = client.SendMessage(req);
            // A string response packet in JSON format is output
            System.out.println(SendMessageResponse.toJsonString(resp));
        } catch (TencentCloudSDKException e) {
            System.out.println(e.toString());
        }
    }
}
```

Step 4. Query the message

After the data is sent, you can check whether it is sent successfully on the message query page. For more information, see Querying Message.

Source Code Demo

Java
Python
Node.JS
PHP
GoLang
.Net
C++

```
import com.tencentcloudapi.common.Credential;
import com.tencentcloudapi.common.profile.ClientProfile;
import com.tencentcloudapi.common.profile.HttpProfile;
import com.tencentcloudapi.common.exception.TencentCloudSDKException;
import com.tencentcloudapi.ckafka.v20190819.CkafkaClient;
import com.tencentcloudapi.ckafka.v20190819.models.*;

public class SendMessage
{
    public static void main(String [] args) {
        try{
            // Instantiate an authentication object. Pass in `secretID` and `secret
            // You can get them at https://console.tencentcloud.com/cam/capi
            Credential cred = new Credential("SecretId", "SecretKey");
            // (Optional) Instantiate an HTTP option
            HttpProfile httpProfile = new HttpProfile();
            httpProfile.setEndpoint("ckafka.tencentcloudapi.com");
            // (Optional) Instantiate a client option
            ClientProfile clientProfile = new ClientProfile();
            clientProfile.setHttpProfile(httpProfile);
            // Instantiate the client object of the requested product. `clientProfi
            CkafkaClient client = new CkafkaClient(cred, "ap-beijing", clientProfil
            // Instantiate a request object. Each API corresponds to a request obje
            SendMessageRequest req = new SendMessageRequest();
            req.setDataHubId("datahub-r6gkngy3");

            BatchContent[] batchContents1 = new BatchContent[2];
            BatchContent batchContent1 = new BatchContent();
            batchContent1.setBody("test1");
            batchContents1[0] = batchContent1;

            BatchContent batchContent2 = new BatchContent();
            batchContent2.setBody("test2");
            batchContents1[1] = batchContent2;

            req.setMessage(batchContents1);
```

```
            // The returned `resp` is an instance of `SendMessageResponse` which co
            SendMessageResponse resp = client.SendMessage(req);
            // A string response packet in JSON format is output
            System.out.println(SendMessageResponse.toJsonString(resp));
        } catch (TencentCloudSDKException e) {
            System.out.println(e.toString());
        }
    }
}
```

```
import json
from tencentcloud.common import credential
from tencentcloud.common.profile.client_profile import ClientProfile
from tencentcloud.common.profile.http_profile import HttpProfile
from tencentcloud.common.exception.tencent_cloud_sdk_exception import TencentCloudS
from tencentcloud.ckafka.v20190819 import ckafka_client, models
try:
    cred = credential.Credential("SecretId", "SecretKey")
    httpProfile = HttpProfile()
    httpProfile.endpoint = "ckafka.tencentcloudapi.com"

    clientProfile = ClientProfile()
    clientProfile.httpProfile = httpProfile
    client = ckafka_client.CkafkaClient(cred, "ap-beijing", clientProfile)

    req = models.SendMessageRequest()
    params = {
        "DataHubId": "datahub-r6gkngy3",
        "Message": [
            {
                "Body": "test1"
            },
            {
                "Body": "test2"
            }
        ]
    }
    req.from_json_string(json.dumps(params))

    resp = client.SendMessage(req)
    print(resp.to_json_string())

except TencentCloudSDKException as err:
    print(err)
```

```
// Depends on tencentcloud-sdk-nodejs version 4.0.3 or higher
const tencentcloud = require("tencentcloud-sdk-nodejs");

const CkafkaClient = tencentcloud.ckafka.v20190819.Client;

const clientConfig = {
  credential: {
    secretId: "SecretId",
    secretKey: "SecretKey",
  },
  region: "ap-beijing",
  profile: {
    httpProfile: {
      endpoint: "ckafka.tencentcloudapi.com",
    },
  },
};

const client = new CkafkaClient(clientConfig);
const params = {
    "DataHubId": "datahub-r6gkngy3",
    "Message": [
        {
            "Body": "test1"
        },
        {
            "Body": "test2"
        }
    ]
};
client.SendMessage(params).then(
  (data) => {
    console.log(data);
  },
  (err) => {
    console.error("error", err);
  }
);
```

```
<?php
require_once 'vendor/autoload.php';
use TencentCloud\\Common\\Credential;
use TencentCloud\\Common\\Profile\\ClientProfile;
use TencentCloud\\Common\\Profile\\HttpProfile;
use TencentCloud\\Common\\Exception\\TencentCloudSDKException;
```

```
use TencentCloud\\Ckafka\\V20190819\\CkafkaClient;
use TencentCloud\\Ckafka\\V20190819\\Models\\SendMessageRequest;
try {

    $cred = new Credential("SecretId", "SecretKey");
    $httpProfile = new HttpProfile();
    $httpProfile->setEndpoint("ckafka.tencentcloudapi.com");

    $clientProfile = new ClientProfile();
    $clientProfile->setHttpProfile($httpProfile);
    $client = new CkafkaClient($cred, "ap-beijing", $clientProfile);

    $req = new SendMessageRequest();

    $params = array(
        "DataHubId" => "datahub-r6gkngy3",
        "Message" => array(
            array(
                "Body" => "test1"
            ),
            array(
                "Body" => "test2"
            )
        )
    );
    $req->fromJsonString(json_encode($params));

    $resp = $client->SendMessage($req);

    print_r($resp->toJsonString());
}
catch(TencentCloudSDKException $e) {
    echo $e;
}
```

```
package main

import (
        "fmt"

        "github.com/tencentcloud/tencentcloud-sdk-go/tencentcloud/common"
        "github.com/tencentcloud/tencentcloud-sdk-go/tencentcloud/common/errors"
        "github.com/tencentcloud/tencentcloud-sdk-go/tencentcloud/common/profile"
        ckafka "github.com/tencentcloud/tencentcloud-sdk-go/tencentcloud/ckafka/v20
)
```

```
func main() {

        credential := common.NewCredential(
                "SecretId",
                "SecretKey",
        )
        cpf := profile.NewClientProfile()
        cpf.HttpProfile.Endpoint = "ckafka.tencentcloudapi.com"
        client, _ := ckafka.NewClient(credential, "ap-beijing", cpf)

        request := ckafka.NewSendMessageRequest()

        request.DataHubId = common.StringPtr("datahub-r6gkngy3")
        request.Message = []*ckafka.BatchContent {
                &ckafka.BatchContent {
                        Body: common.StringPtr("test1"),
                },
                &ckafka.BatchContent {
                        Body: common.StringPtr("test2"),
                },
        }

        response, err := client.SendMessage(request)
        if _, ok := err.(*errors.TencentCloudSDKError); ok {
                fmt.Printf("An API error has returned: %s", err)
                return
        }
        if err != nil {
                panic(err)
        }
        fmt.Printf("%s", response.ToJsonString())
}
```

```
using System;
using System.Threading.Tasks;
using TencentCloud.Common;
using TencentCloud.Common.Profile;
using TencentCloud.Ckafka.V20190819;
using TencentCloud.Ckafka.V20190819.Models;

namespace TencentCloudExamples
{
    class SendMessage
    {
```

```
        static void Main(string[] args)
        {
            try
            {
                Credential cred = new Credential {
                    SecretId = "SecretId",
                    SecretKey = "SecretKey"
                };

                ClientProfile clientProfile = new ClientProfile();
                HttpProfile httpProfile = new HttpProfile();
                httpProfile.Endpoint = ("ckafka.tencentcloudapi.com");
                clientProfile.HttpProfile = httpProfile;

                CkafkaClient client = new CkafkaClient(cred, "ap-beijing", clientPr
                SendMessageRequest req = new SendMessageRequest();
                req.DataHubId = "datahub-r6gkngy3";
                BatchContent batchContent1 = new BatchContent();
                batchContent1.Body = "test1";

                BatchContent batchContent2 = new BatchContent();
                batchContent2.Body = "test2";
                req.Message = new BatchContent[] { batchContent1, batchContent2 };

                SendMessageResponse resp = client.SendMessageSync(req);
                Console.WriteLine(AbstractModel.ToJsonString(resp));
            }
            catch (Exception e)
            {
                Console.WriteLine(e.ToString());
            }
            Console.Read();
        }
    }
}
```

```
#include <tencentcloud/core/Credential.h>
#include <tencentcloud/core/profile/ClientProfile.h>
#include <tencentcloud/core/profile/HttpProfile.h>
#include <tencentcloud/ckafka/v20190819/CkafkaClient.h>
#include <tencentcloud/ckafka/v20190819/model/SendMessageRequest.h>
#include <tencentcloud/ckafka/v20190819/model/SendMessageResponse.h>
#include <iostream>
#include <string>
#include <vector>
```

```
using namespace TencentCloud;
using namespace TencentCloud::Ckafka::V20190819;
using namespace TencentCloud::Ckafka::V20190819::Model;
using namespace std;

int main() {

        Credential cred = Credential("SecretId", "SecretKey");

        HttpProfile httpProfile = HttpProfile();
        httpProfile.SetEndpoint("ckafka.tencentcloudapi.com");

        ClientProfile clientProfile = ClientProfile();
        clientProfile.SetHttpProfile(httpProfile);
        CkafkaClient client = CkafkaClient(cred, "ap-beijing", clientProfile);

        SendMessageRequest req = SendMessageRequest();

        req.SetDataHubId("datahub-r6gkngy3");
        BatchContent batchContent1;
        batchContent1.SetBody("test1");
        BatchContent batchContent2;
        batchContent2.SetBody("test2");

        vector<BatchContent> batchContents1 = {batchContent1, batchContent2};
        req.SetMessage(batchContents1);

        auto outcome = client.SendMessage(req);
        if (!outcome.IsSuccess())
        {
            cout << outcome.GetError().PrintAll() << endl;
            return -1;
        }
        SendMessageResponse resp = outcome.GetResult();
        cout << resp.ToJsonString() << endl;

    return 0;
}
```

Elastic Topic Message Sending and Receiving
Java SDK

Last updated：2024-09-09 21:24:45

Overview

This document introduces the directions for using the Java client to connect to an elastic Topic and send and receive messages.

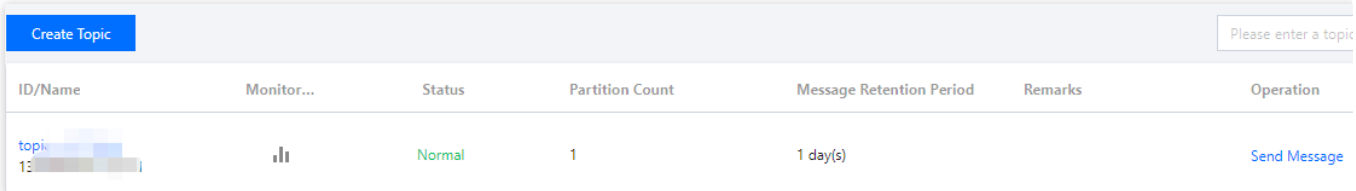
Prerequisites

Install JDK 1.8 or later
Install Maven 2.5 or later

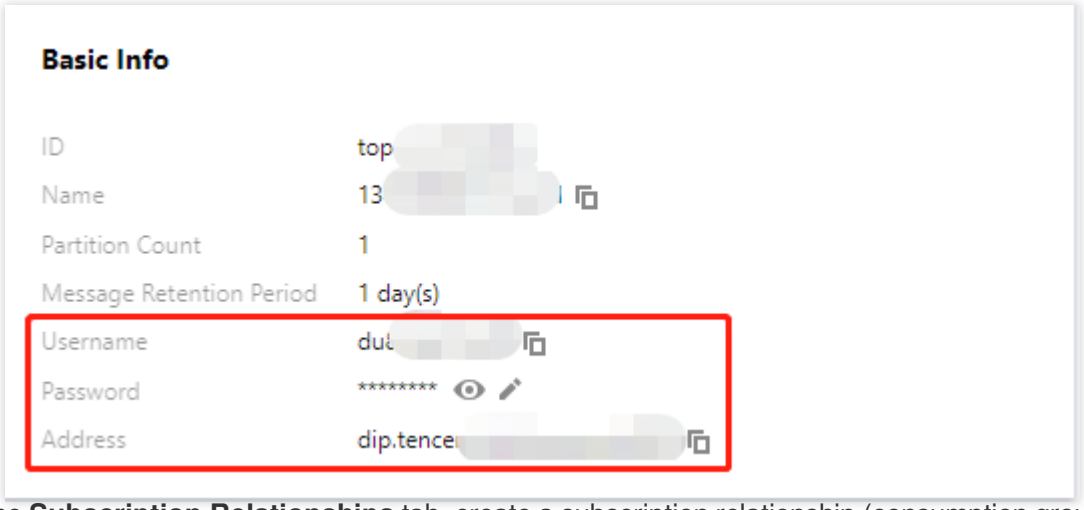
Directions

Step 1: Creating a Topic and Subscription Relationship

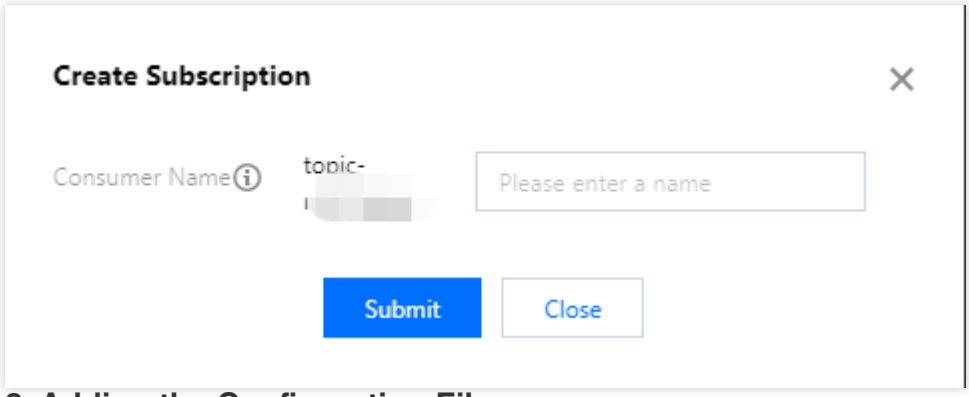
1. On the Elastic Topic list page of the console, create a Topic.



2. Click the **ID** of the Topic to enter the basic information page and obtain the username, password, and address information.



3. In the **Subscription Relationships** tab, create a subscription relationship (consumption group).



Step 2: Adding the Configuration File

1. Add the following dependencies to pom.xml.

```
<dependencies>
   <dependency>
      <groupId>org.apache.kafka</groupId>
      <artifactId>kafka-clients</artifactId>
      <version>2.1.0</version>
   </dependency>
   <dependency>
      <groupId>org.slf4j</groupId>
      <artifactId>slf4j-api</artifactId>
      <version>1.7.5</version>
   </dependency>
   <dependency>
      <groupId>org.slf4j</groupId>
      <artifactId>slf4j-simple</artifactId>
      <version>1.6.4</version>
   </dependency>
</dependencies>
```

2. Create a JAAS configuration file `ckafka_client_jaas.conf` and modify it with the user created on the **User Management** interface.

```
KafkaClient {
    org.apache.kafka.common.security.plain.PlainLoginModule required
    username="username"
    password="password";
};
```

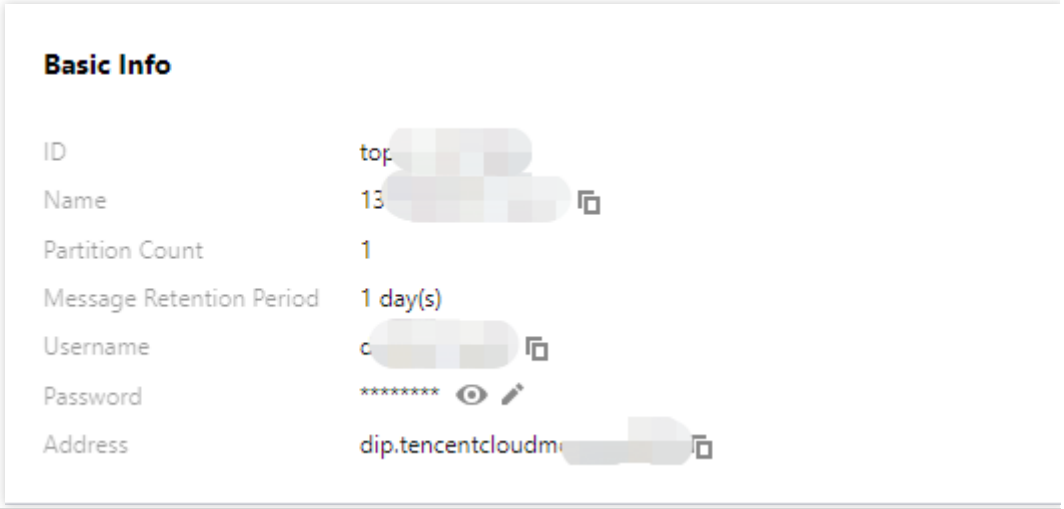
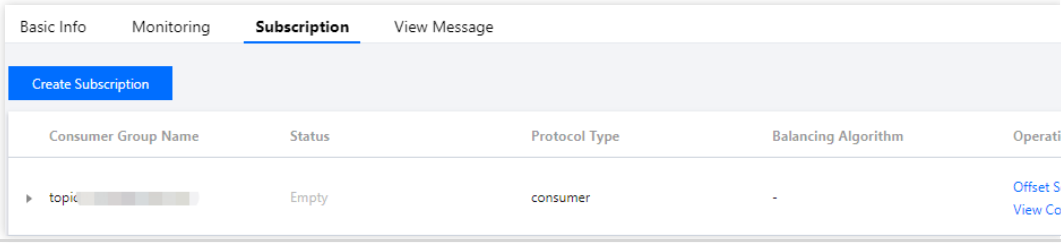
3. Create the CKafka configuration file kafka.properties.

```
## Configure the connection address. It can be obtained from the basic information
bootstrap.servers=xx.xx.xx.xx:port
## The topic name. It can be obtained from the basic information page of an elastic
topic=XXX
## The consumption group name. It can be obtained from the **Subscription Relations
group.id=XXX
## SASL configuration
java.security.auth.login.config.plain=/xxxx/ckafka_client_jaas.conf
```

4. Create the configuration file load program named CKafkaConfigurer.java.

```
public class CKafkaConfigurer {

    private static Properties properties;

    public static void configureSaslPlain() {
        //If it is already set by -D or other means, you can skip setting here.
        if (null == System.getProperty("java.security.auth.login.config")) {
            //Make sure to change XXX to your own path.
            System.setProperty("java.security.auth.login.config",
                    getCKafkaProperties().getProperty("java.security.auth.login.con
        }
    }

    public synchronized static Properties getCKafkaProperties() {
        if (null != properties) {
            return properties;
        }
        //Get the content of the configuration file `kafka.properties`.
        Properties kafkaProperties = new Properties();
        try {
            kafkaProperties.load(CKafkaProducerDemo.class.getClassLoader().getResou
        } catch (Exception e) {
            System.out.println("getCKafkaProperties error");
        }
        properties = kafkaProperties;
```

```
        return kafkaProperties;
    }
}
```

Parameter	Description
bootstrapServers	The connection address. It can be obtained from the basic information page of an elastic Topic 
username	The username. It can be obtained from the basic information page of an elastic Topic in the co
password	The user password. It can be obtained from the basic information page of an elastic Topic in th
topic	The topic name. It can be obtained from the basic information page of an elastic Topic in the c
group.id	The consumption group name. It can be obtained from the subscription relationship list in the c 

Step 3: Producing Messages

1. Create a program named KafkaSaslProducerDemo.java to send messages.

```
public class KafkaSaslProducerDemo {

public static void main(String[] args) {
   //Set the path of the JAAS configuration file.
   CKafkaConfigurer.configureSaslPlain();
```

```
        //Load kafka.properties.
        Properties kafkaProperties = CKafkaConfigurer.getCKafkaProperties();

        Properties props = new Properties();
        //To set the connection point, obtain the connection point of the correspondi
        props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
                kafkaProperties.getProperty("bootstrap.servers"));

        //
        //  SASL_PLAINTEXT public network connection
        //
        props.put(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG, "SASL_PLAINTEXT");
        //Use Plain mode for SASL.
        props.put(SaslConfigs.SASL_MECHANISM, "PLAIN");

        //The method for serializing TDMQ for Kafka messages.
        props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
                "org.apache.kafka.common.serialization.StringSerializer");
        props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
                "org.apache.kafka.common.serialization.StringSerializer");
        //The maximum request wait time.
        props.put(ProducerConfig.MAX_BLOCK_MS_CONFIG, 30 * 1000);
        //Set the number of internal retries for the client.
        props.put(ProducerConfig.RETRIES_CONFIG, 5);
        //Set the internal retry interval for the client.
        props.put(ProducerConfig.RECONNECT_BACKOFF_MS_CONFIG, 3000);
        //If `ack` is 0, the producer will not wait for confirmation from the broker,
        //If `ack` is 1, the broker leader will directly return `ack` without waiting
        //If `ack` is `all`, the broker leader will return `ack` only after receiving
        props.put(ProducerConfig.ACKS_CONFIG, "all");
        //Build a producer object. Note: A producer object is thread-safe, and genera
        KafkaProducer<String, String> producer = new KafkaProducer<>(props);

        //Build a TDMQ for CKafka message.
        String topic = kafkaProperties.getProperty("topic"); //The Topic to which the
        String value = "this is ckafka msg value"; //Content of the message.

        try {
            //Batch getting Future objects can speed up the process. Note that the bat
            List<Future<RecordMetadata>> futures = new ArrayList<>(128);
            for (int i = 0; i < 100; i++) {
                //Send the message and get a Future object.
                ProducerRecord<String, String> kafkaMessage = new ProducerRecord<>(topi
                        value + ": " + i);
                Future<RecordMetadata> metadataFuture = producer.send(kafkaMessage);
                futures.add(metadataFuture);
```

```
            }
            producer.flush();
            for (Future<RecordMetadata> future : futures) {
                //Synchronize the Future object obtained.
                RecordMetadata recordMetadata = future.get();
                System.out.println("Produce ok:" + recordMetadata.toString());
            }
        } catch (Exception e) {
            //If the sending still fails after client internal retries, the system nee
            System.out.println("error occurred");
        }
    }
}
```

2. Compile and run KafkaSaslProducerDemo.java to send messages.

3. View the operation result (output).

```
Produce ok:ckafka-topic-demo-0@198
Produce ok:ckafka-topic-demo-0@199
```

Step 4: Consuming Messages

1. Create the program for a consumer to subscribe to messages named `KafkaSaslConsumerDemo.java` .

```
public class KafkaSaslConsumerDemo {

    public static void main(String[] args) {
        //Set the path of the JAAS configuration file.
        CKafkaConfigurer.configureSaslPlain();

        //Load `kafka.properties`.
        Properties kafkaProperties = CKafkaConfigurer.getCKafkaProperties();

        Properties props = new Properties();
        //Set the connection point. Get the connection point of the corresponding top
        props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
                kafkaProperties.getProperty("bootstrap.servers"));

        //
        //  SASL_PLAINTEXT public network connection
        //
        props.put(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG, "SASL_PLAINTEXT");
        //Use Plain mode for SASL.
        props.put(SaslConfigs.SASL_MECHANISM, "PLAIN");

        //Consumer timeout period
```

```
        //If a consumer does not send a heartbeat within this duration, the server de
        props.put(ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG, 30000);
        //The maximum time interval between two polls.
        //Before the version 0.10.1.0 is released, these two concepts were mixed, bot
        props.put(ConsumerConfig.MAX_POLL_INTERVAL_MS_CONFIG, 30000);
        //The maximum number of each poll.
        //Do not set this parameter to an excessively large value. If polled messages
        props.put(ConsumerConfig.MAX_POLL_RECORDS_CONFIG, 30);
        //Set the method for deserializing messages.
        props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
                "org.apache.kafka.common.serialization.StringDeserializer");
        props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
                "org.apache.kafka.common.serialization.StringDeserializer");
        //The consumption group of the current consumption instance after you apply f
        //The instances in the same consumption group consume messages in load-balanc
        props.put(ConsumerConfig.GROUP_ID_CONFIG, kafkaProperties.getProperty("group.
        //Build a consumption object. This generates a consumption instance.
        KafkaConsumer<String, String> consumer = new KafkaConsumer<String, String>(pr
        //Set one or more topics to which the consumption group subscribes.
        //It is recommended to configure consumption instances with the same `GROUP_I
        List<String> subscribedTopics = new ArrayList<String>();
        //If you want to subscribe to multiple topics, add the topics here.
        //You need to create the topics in the console in advance.
        String topicStr = kafkaProperties.getProperty("topic");
        String[] topics = topicStr.split(",");
        for (String topic : topics) {
           subscribedTopics.add(topic.trim());
        }
        consumer.subscribe(subscribedTopics);

        //Consume messages in loop.
        while (true) {
           try {
              ConsumerRecords<String, String> records = consumer.poll(1000);
              //All messages should be consumed before the next poll, and the total d
              for (ConsumerRecord<String, String> record : records) {
                 System.out.println(
                         String.format("Consume partition:%d offset:%d", record.parti
                                 record.offset()));
              }
           } catch (Exception e) {
              System.out.println("consumer error!");
           }
        }
    }
}
```

2. Compile and run KafkaSaslConsumerDemo.java to consume messages.

3. View the execution result.

```
Consume partition:0 offset:298
Consume partition:0 offset:299
```

Python SDK

Last updated：2024-09-09 21:25:33

Overview

This task uses the Python client as an example to guide you on how to use the elastic Topic of TDMQ for CKafka and send and receive messages.

Prerequisites

Install Python
Install pip

Directions

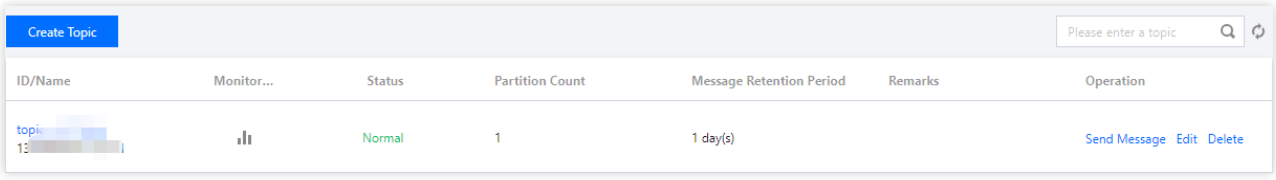
Step 1: Preparing the Environment

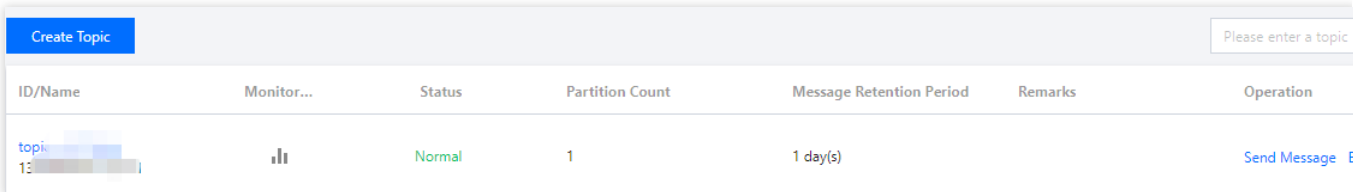
Run the following command to install the Python dependency database.

```
pip install kafka-python
```

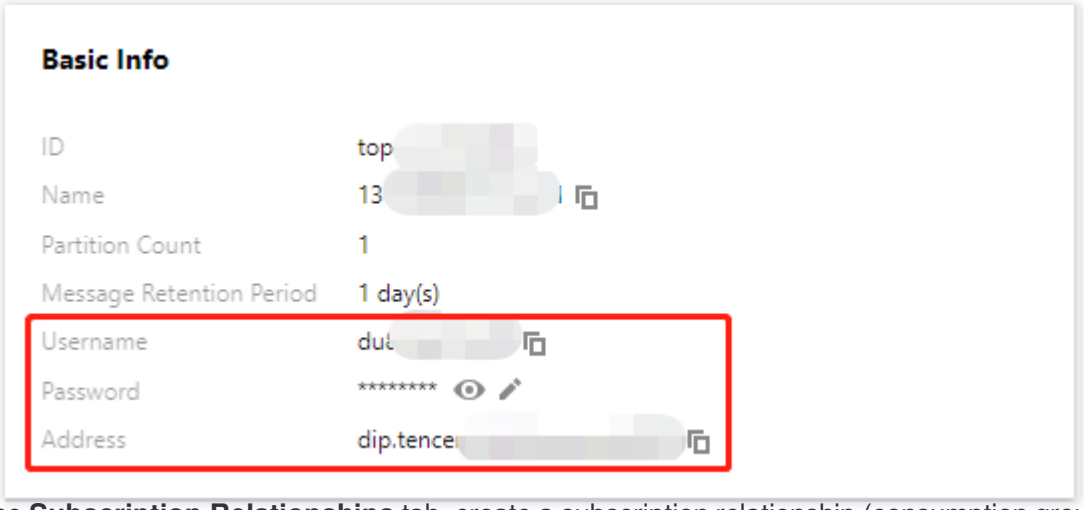
Step 2: Creating a Topic and Subscription Relationship

1. On the Elastic Topic list page of the console, create a Topic.

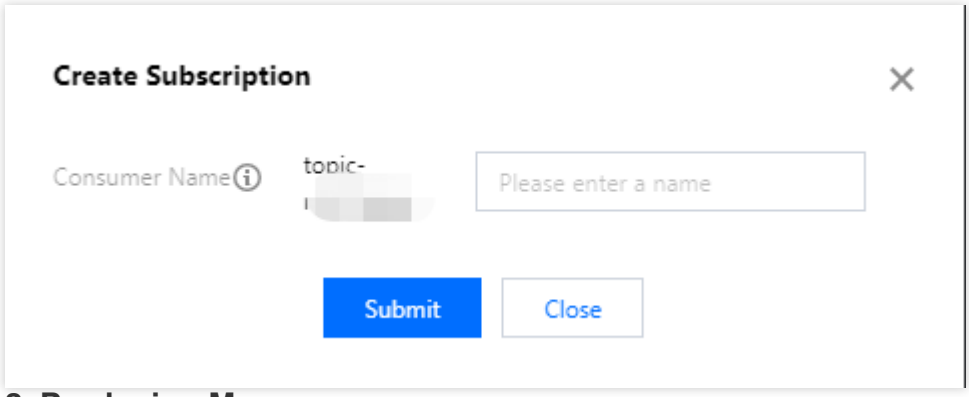




2. Click the **ID** of the Topic to enter the **Basic Information** page and obtain the username, password, and address information.



3. In the **Subscription Relationships** tab, create a subscription relationship (consumption group).

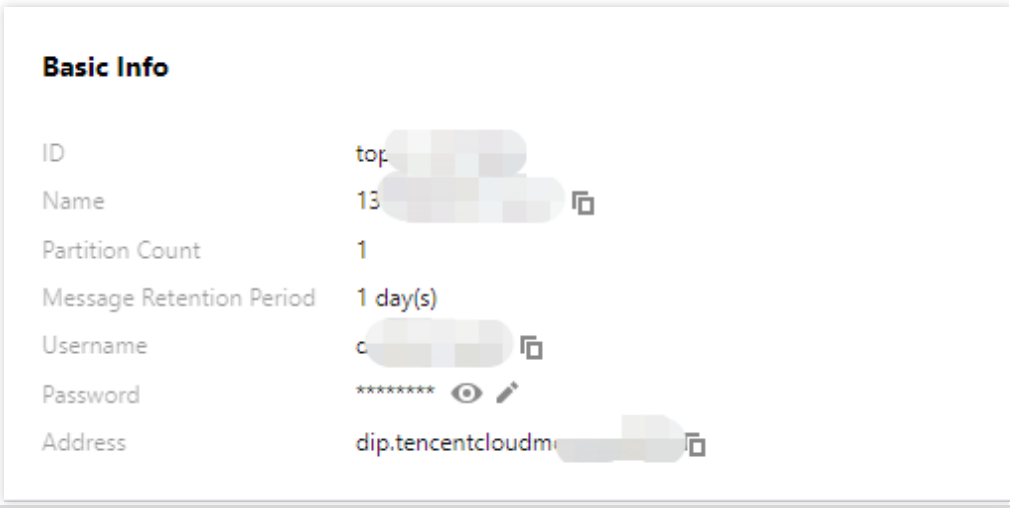


Step 3: Producing Messages

1. Modify the configuration parameters in the message production program `producer.py` .

```
producer = KafkaProducer(
        bootstrap_servers = ['xx.xx.xx.xx:port'],$address
        api_version = (1, 1),
        security_protocol = "SASL_PLAINTEXT",
        sasl_mechanism = "PLAIN",
        sasl_plain_username = "username",# username
        sasl_plain_password = "password",# password
)

message = "Hello World! Hello Ckafka!"
msg = json.dumps(message).encode()
producer.send('topic_name', value = msg)#Topic name
print("produce message " + message + " success.")
producer.close()
```

Parameter	Description
bootstrapServers	The connection address. It can be obtained from the basic information page of an elastic T

	
sasl_plain_username	The username. It can be obtained from the basic information page of an elastic Topic in th
sasl_plain_password	The user password. It can be obtained from the basic information page of an elastic Topic
topic_name	The topic name. It can be obtained from the basic information page of an elastic Topic in t

2. Compile and run producer.py.

3. View the execution result.

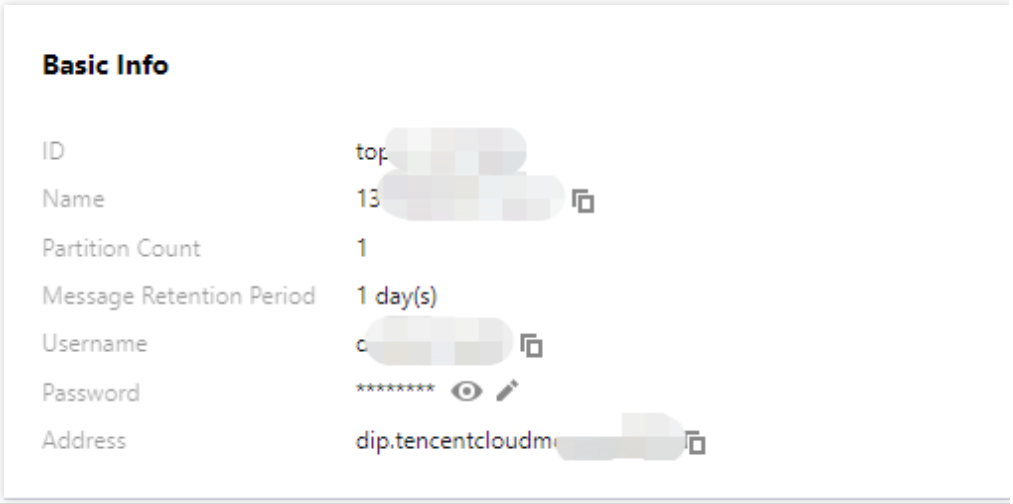
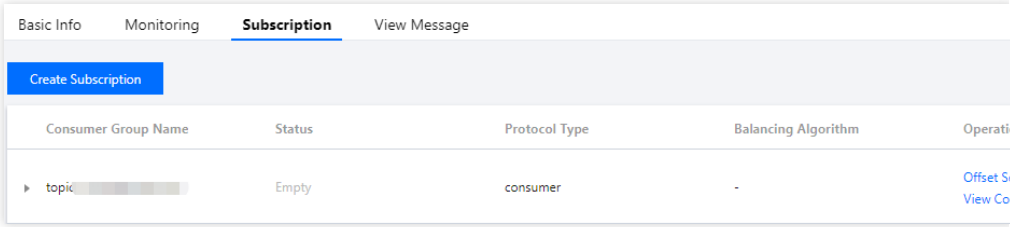
```
[root@VM-8-16-centos sasl]# python3 producer.py
produce message Hello World! Hello Ckafka! success.
```

Step 4: Consuming Messages

1. Modify the configuration parameters in the message consumption program consumer.py.

```
consumer = KafkaConsumer(
    'topic_name',#topic name
    group_id = "group_id",#Consumption group
    bootstrap_servers = ['xx.xx.xx.xx:port'],# address
    api_version = (1,1),

    security_protocol = "SASL_PLAINTEXT",
    sasl_mechanism = 'PLAIN',
    sasl_plain_username = "username",# username
    sasl_plain_password = "password",# password
)

for message in consumer:
    print ("Topic:[%s] Partition:[%d] Offset:[%d] Value:[%s]" %
    (message.topic, message.partition, message.offset, message.value))
```

Parameter	Description

bootstrapServers	The connection address. It can be obtained from the basic information page of an elastic T 
sasl_plain_username	The username. It can be obtained from the basic information page of an elastic Topic in th
sasl_plain_password	The user password. It can be obtained from the basic information page of an elastic Topic
topic_name	The topic name. It can be obtained from the basic information page of an elastic Topic in t
group.id	The consumption group name. It can be obtained from the subscription relationship list of a the console. 

2. Compile and run consumer.py.

3. View the execution result.

```
[root@VM-8-16-centos sasl]# python3 consumer.py
Topic:[test] Partition:[0] Offset:[2011] Value:[b'"Hello World! Hello Ckafka!"']
```

Go SDK

Last updated：2024-09-09 21:26:21

Overview

This document introduces the directions for using the Go client to connect to an elastic Topic of CKafka and send and receive messages.

Prerequisites

Install Go

Directions

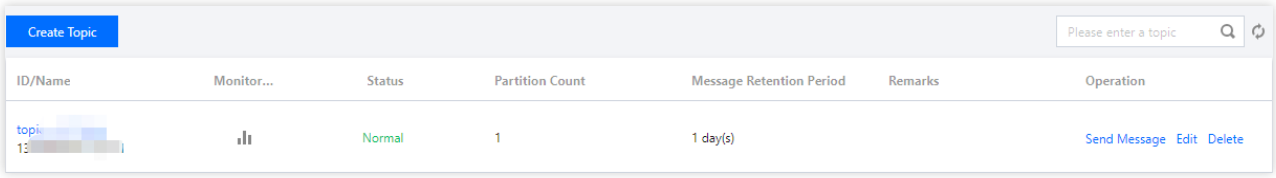
Step 1: Preparing the Environment

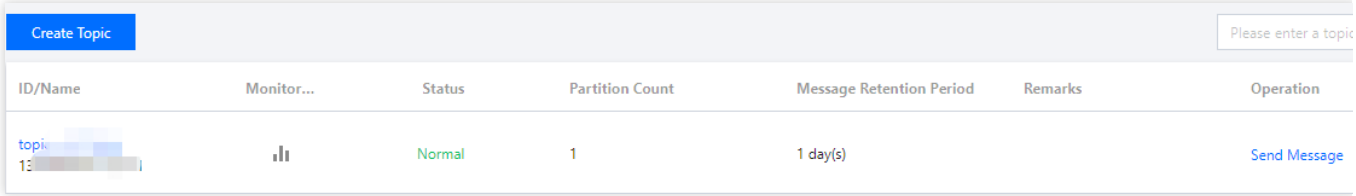
Install Kafka dependencies.

```
go get -v gopkg.in/confluentinc/confluent-kafka-go.v1/kafka
```

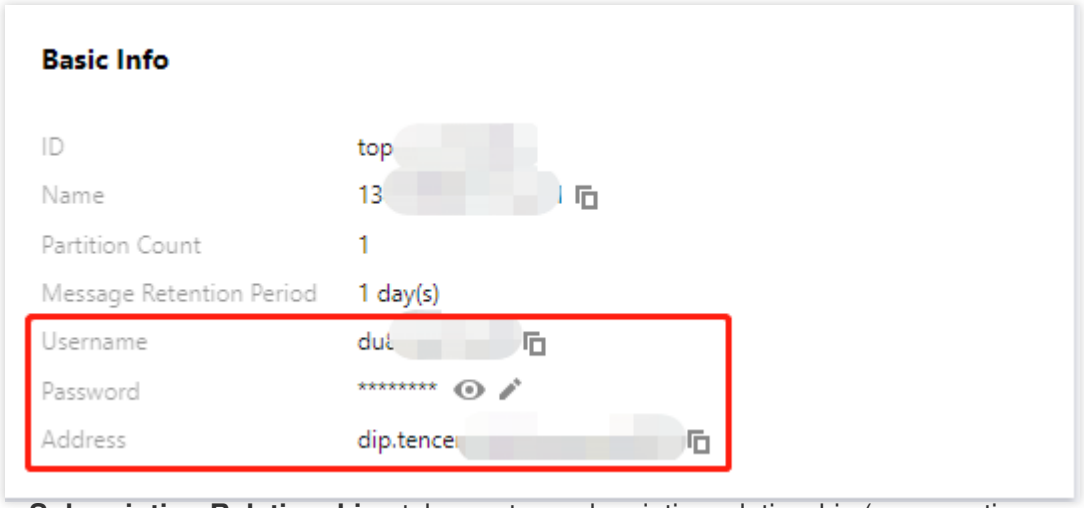
Step 2: Creating a Topic and Subscription Relationship

1. On the Elastic Topic list page of the console, create a Topic.

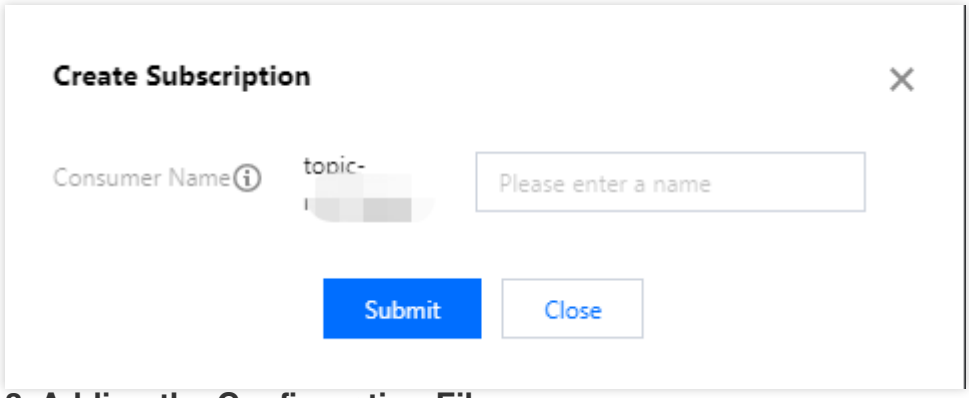




2. Click the **ID** of the Topic to enter the **Basic Information** page and obtain the username, password, and address information.



3. In the **Subscription Relationships** tab, create a subscription relationship (consumption group).

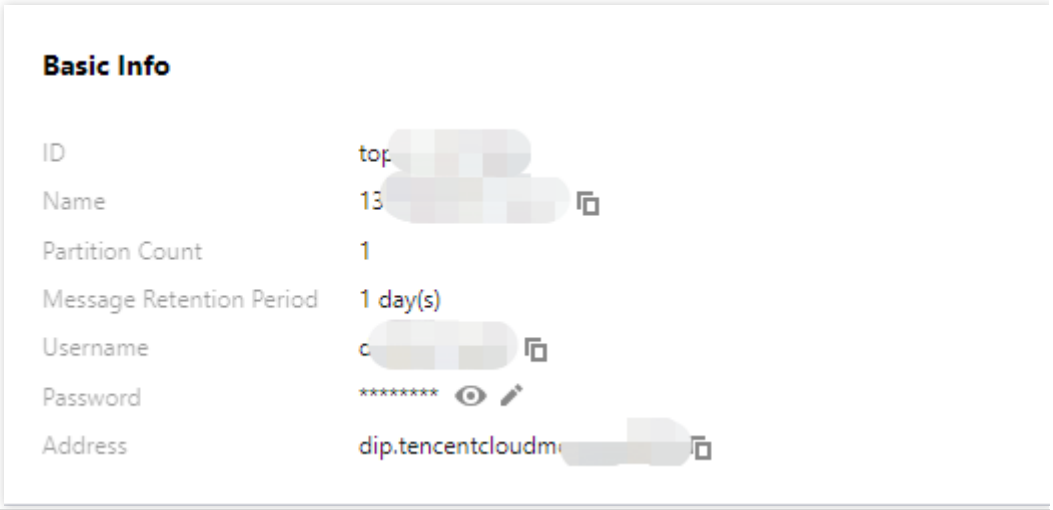
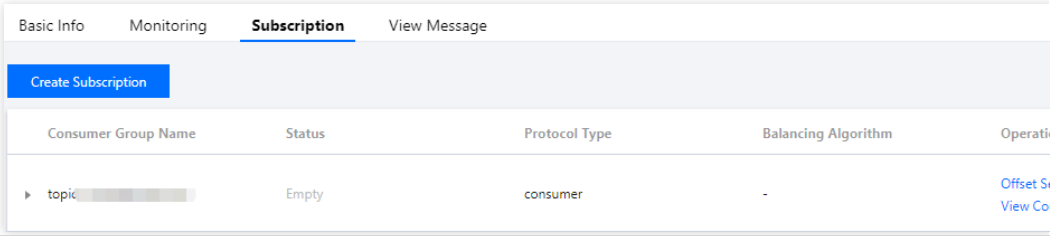


Step 3: Adding the Configuration File

Create the configuration file kafka.json.

Create the configuration file kafka.json.

```
{
  "topic": [
      "xxxx"
  ],
  "sasl": {
      "username": "yourUserName",
      "password": "yourPassword",
  },
  "bootstrapServers": [
      "xx.xx.xx.xx:port"
  ],
  "consumerGroupId": "yourConsumerId"
 }
```

Parameter	Description
bootstrapServers	The connection address. It can be obtained from the basic information page of an elastic Topi 
username	The username. It can be obtained from the basic information page of an elastic Topic in the co
password	The user password. It can be obtained from the basic information page of an elastic Topic in t
topic	The topic name. It can be obtained from the basic information page of an elastic Topic in the c
consumerGroupId	The consumption group name. It can be obtained from the subscription relationship list of an e console. 

Step 4: Producing Messages

1. Write a message production program.

```
package main

import (
    "fmt"
    "gokafkademo/config"
    "log"
    "strings"

    "github.com/confluentinc/confluent-kafka-go/kafka"
)
```

```
func main() {

    cfg, err := config.ParseConfig("../config/kafka.json")
    if err != nil {
        log.Fatal(err)
    }

    p, err := kafka.NewProducer(&kafka.ConfigMap{
        // To set connection points, obtain the corresponding Topic connection po
        "bootstrap.servers": strings.Join(cfg.Servers, ","),
        // The SASL authentication mechanism is PLAIN by default.
        "sasl.mechanism": "PLAIN",
        // Configure the ACL policy locally.
        "security.protocol": "SASL_PLAINTEXT",
        // The username is the configured username, and password is the configure
        "sasl.username": cfg.SASL.Username,
        "sasl.password": cfg.SASL.Password,

        // There are three ack mechanisms for a Kafka producer as described below
        // -1 or all: The broker responds to the producer and continues to send t
        // This configuration offers the highest data reliability. There is no me
        // It can be used together with the `min.insync.replicas` parameter at th
        // 0: The producer does not wait for the confirmation of synchronization
        // 1: The producer sends the next message (or the next batch of messages)
        // When the user does not specify a configuration, the default value is 1
        "acks": 1,
        // The number of retries upon request error. It is recommended to set the
        "retries": 0,
        // The time interval between a failed request and the next retry attempt.
        "retry.backoff.ms": 100,
        // Timeout period for network requests made by the producer.
        "socket.timeout.ms": 6000,
        // Set the internal retry interval of the client.
        "reconnect.backoff.max.ms": 3000,
    })
    if err != nil {
        log.Fatal(err)
    }

    defer p.Close()

    // Deliver the produced messages to the report processing program.
    go func() {
        for e := range p.Events() {
            switch ev := e.(type) {
            case *kafka.Message:
```

```
                if ev.TopicPartition.Error != nil {
                    fmt.Printf("Delivery failed: %v\\n", ev.TopicPartition)
                } else {
                    fmt.Printf("Delivered message to %v\\n", ev.TopicPartition)
                }
            }
        }
    }()

    // Asynchronously send messages.
    topic := cfg.Topic
    for _, word := range []string{"Confluent-Kafka", "Golang Client Message"} {
        _ = p.Produce(&kafka.Message{
            TopicPartition: kafka.TopicPartition{Topic: &topic, Partition: kafka.
            Value:          []byte(word),
        }, nil)
    }

    // Wait for message delivery.
    p.Flush(10 * 1000)
```

2. Compile and run the program to send messages.

```
go run main.go
```

3. View the execution result. An example is shown below.

```
Delivered message to test[0]@628
Delivered message to test[0]@629
```

Step 5: Consuming Messages

1. Write the message consumption program.

```
package main

import (
    "fmt"
    "gokafkademo/config"
    "log"
    "strings"

    "github.com/confluentinc/confluent-kafka-go/kafka"
)

func main() {
```

```
    cfg, err := config.ParseConfig("../config/kafka.json")
    if err != nil {
        log.Fatal(err)
    }

    c, err := kafka.NewConsumer(&kafka.ConfigMap{
        // To set connection points, obtain the corresponding Topic connection po
        "bootstrap.servers": strings.Join(cfg.Servers, ","),
        // The SASL authentication mechanism is PLAIN by default.
        "sasl.mechanism": "PLAIN",
        // Configure the ACL policy locally.
        "security.protocol": "SASL_PLAINTEXT",
        // The username is the configured username, and password is the configure
        "sasl.username": cfg.SASL.Username,
        "sasl.password": cfg.SASL.Password,
        // The configured message consumption group.
        "group.id":          cfg.ConsumerGroupId,
        "auto.offset.reset": "earliest",

        // When you use the Kafka consumption group mechanism, set the consumer t
        // Initiate the Rebalance process. Currently, this value needs to be conf
        "session.timeout.ms": 10000,
    })

    if err != nil {
        log.Fatal(err)
    }
    // The list of subscribed message topics.
    err = c.SubscribeTopics([]string{"test", "test-topic"}, nil)
    if err != nil {
        log.Fatal(err)
    }

    for {
        msg, err := c.ReadMessage(-1)
        if err == nil {
            fmt.Printf("Message on %s: %s\\n", msg.TopicPartition, string(msg.Val
        } else {
            // The client will automatically try to recover all errors.
            fmt.Printf("Consumer error: %v (%v)\\n", err, msg)
        }
    }

    c.Close()
}
```

2. Compile and run the program to consume messages.

```
go run main.go
```

3. View the execution result. An example is shown below.

```
Message on test[0]@628: Confluent-Kafka
Message on test[0]@629: Golang Client Message
```

PHP SDK

Last updated：2024-09-09 21:27:07

Overview

This document introduces the directions for using the PHP client to connect to an elastic Topic of CKafka and send and receive messages.

Prerequisites

Install librdkafka
Install PHP 5.6 or later
Install PEAR

Directions

Step 1: Preparing the Environment

1. Find the latest version of the rdkafka PHP extension package on the rdkafka Official Page.

Note

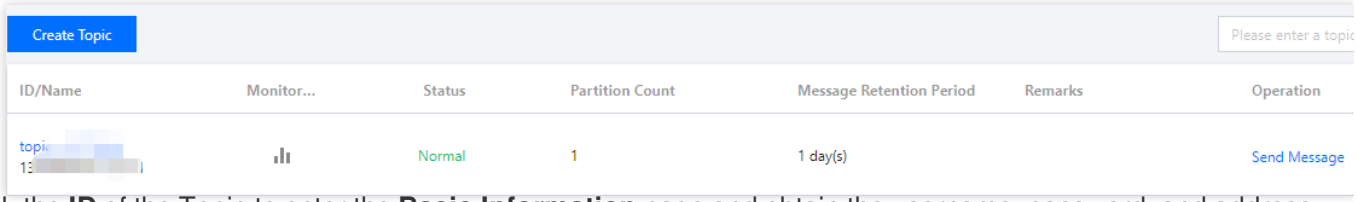
Different package versions require different PHP version requirements. Here, version 4.1.2 is used as an example.

2. Install the rdkafka extension.

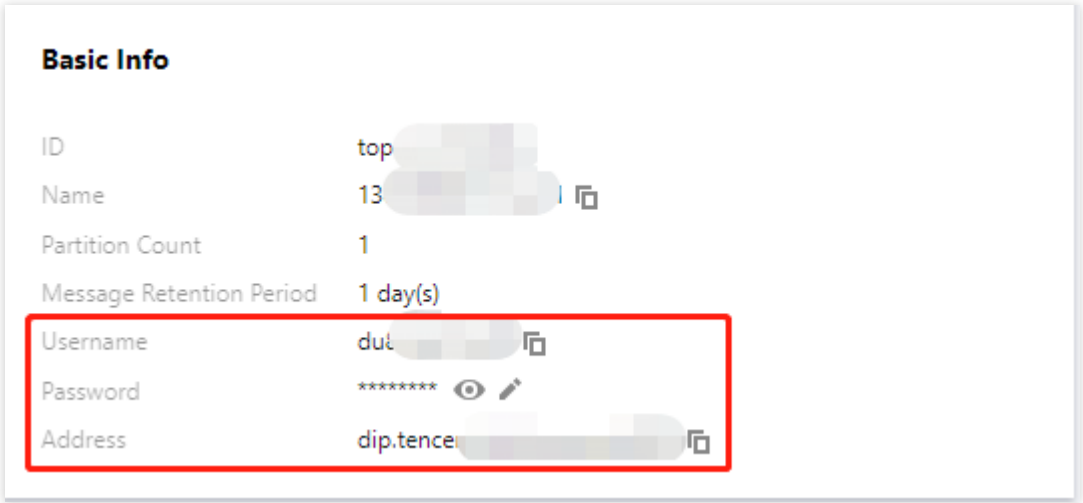
```
wget --no-check-certificate https://pecl.php.net/get/rdkafka-4.1.2.tgz
pear install rdkafka-4.1.2.tgz
# If the installation succeeds, the system will prompt install ok and You should
# If the installation fails and the system prompts could not extract the package
# Resolve other errors according to the provided instructions.
# After successful installation, add `extension=rdkafka.so` to `php.ini`.
# After `php --ini` is executed, `Loaded Configuration File:` shows the location
echo 'extension=rdkafka.so' >> /etc/php.ini
```

Step 2: Creating a Topic and Subscription Relationship

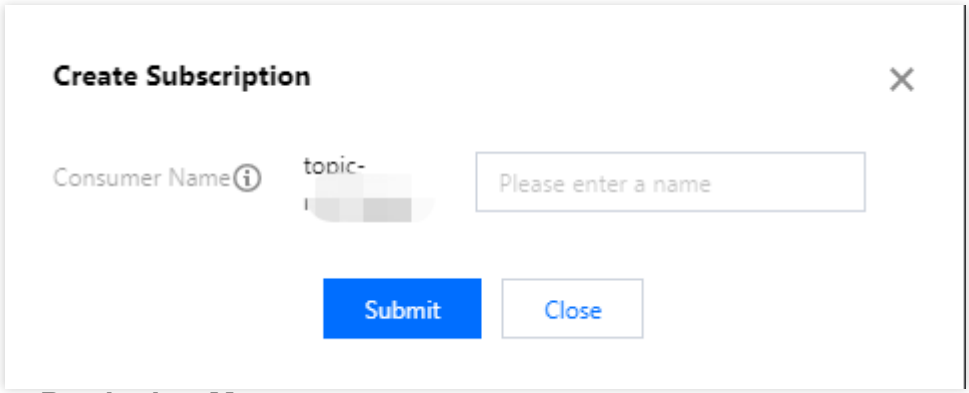
1. On the Elastic Topic list page of the console, create a Topic.



2. Click the **ID** of the Topic to enter the **Basic Information** page and obtain the username, password, and address information.



3. In the **Subscription Relationships** tab, create a subscription relationship (consumption group).



Step 3: Producing Messages

1. Write the message production program Producer.php.

```
<?php

$setting = require __DIR__ . '/CKafkaSetting.php';

$conf = new RdKafka\\Conf();
// Set the entry service, and get the corresponding service address from the consol
$conf->set('bootstrap.servers', $setting['bootstrap_servers']);
// ---------- Set this part if SASL authentication is enabled. ----------
// The SASL authentication mechanism is PLAIN by default.
$conf->set('sasl.mechanism', 'PLAIN');
```

```
// Set `username`: The username configured in **User Management**.
$conf->set('sasl.username', $setting['sasl_username']);
// Set `password`: the password configured in **User Management**
$conf->set('sasl.password', $setting['sasl_password']);
// Configure the ACL policy locally.
$conf->set('security.protocol', 'SASL_PLAINTEXT');
// ---------- Set this part if SASL authentication is enabled. ----------
// There are three ack mechanisms for a Kafka producer as described below:
// -1 or all: The broker responds to the producer and continues to send the next me
// This configuration offers the highest data reliability. There is no message loss
// It can be used together with the `min.insync.replicas` parameter at the topic le
// 0: The producer does not wait for the confirmation of synchronization completion
// (Data loss may occur when the server fails. If the leader is down but the produc
// 1: The producer sends the next message (or batch) only after the leader has succ
// (If the leader is down but not yet replicated, the message may be lost.)
// When the user does not specify a configuration, the default value is 1. Users ca
$conf->set('acks', '1');
// The number of retries upon request error. It is recommended to set the value to
$conf->set('retries', '0');
// The time interval between a failed request and the next retry attempt.
$conf->set('retry.backoff.ms', 100);
// Timeout period for network requests made by the producer.
$conf->set('socket.timeout.ms', 6000);
$conf->set('reconnect.backoff.max.ms', 3000);

// Register the callback for message sending.
$conf->setDrMsgCb(function ($kafka, $message) {
    echo '**Producer** sent a message: message=' . var_export($message, true) . "\\
});
// Register the callback for message sending error.
$conf->setErrorCb(function ($kafka, $err, $reason) {
    echo "**Producer** runs into an error while sending the message: err=$err reaso
});

$producer = new RdKafka\\Producer($conf);
// Please set `LOG_DEBUG` when you perform Debug.
//$producer->setLogLevel(LOG_DEBUG);
$topicConf = new RdKafka\\TopicConf();
$topic = $producer->newTopic($setting['topic_name'], $topicConf);
// Produce a message and send it.
for ($i = 0; $i < 5; $i++) {
    // `RD_KAFKA_PARTITION_UA` lets Kafka select a partition freely.
    $topic->produce(RD_KAFKA_PARTITION_UA, 0, "Message $i");
    $producer->poll(0);
}

while ($producer->getOutQLen() > 0) {
```

```
    $producer->poll(50);
}

echo "**Producer** sent the message successfully\\n";
```

2. Run Producer.php to send a message.

```
php Producer.php
```

3. View the execution result.

```
>**Producer** sends a message: message=RdKafka\\Message::__set_state(array(
>   'err' => 0,
>   'topic_name' => 'topic_name',
>   'timestamp' => 1618800895159,
>   'partition' => 0,
>   'payload' => 'Message 0',
>   'len' => 9,
>   'key' => NULL,
>   'offset' => 0,
>   'headers' => NULL,
>))
>**Producer** sends a message: message=RdKafka\\Message::__set_state(array(
>   'err' => 0,
>   'topic_name' => 'topic_name',
>   'timestamp' => 1618800895159,
>   'partition' => 0,
>   'payload' => 'Message 1',
>   'len' => 9,
>   'key' => NULL,
>   'offset' => 1,
>   'headers' => NULL,
>))

...

>**Producer** sent the message successfully.
```

Step 4: Consuming Messages

1. Write the message subscription and consumption program Consumer.php.

```
<?php

$setting = require __DIR__ . '/CKafkaSetting.php';
```

```
$conf = new RdKafka\\Conf();
$conf->set('group.id', $setting['group_id']);
// Set the entry service, and get the corresponding service address from the consol
$conf->set('bootstrap.servers', $setting['bootstrap_servers']);
// ---------- Set this part if SASL authentication is enabled. ----------
// The SASL authentication mechanism is PLAIN by default.
$conf->set('sasl.mechanism', 'PLAIN');
// Set username : The username configured in **User Management**.
$conf->set('sasl.username', $setting['sasl_username']);
// Set `password`: The password configured in **User Management**.
$conf->set('sasl.password', $setting['sasl_password']);
// Configure the ACL policy locally.
$conf->set('security.protocol', 'SASL_PLAINTEXT');
// ---------- Set this part if SASL authentication is enabled. ----------
// When you use the Kafka consumption group mechanism, set the consumer timeout per
// the consumer is considered as failed and the broker will initiate rebalance.
$conf->set('session.timeout.ms', 10000);
// Client request timeout period. If no response is received after this time period
$conf->set('request.timeout.ms', 305000);
// Set the internal retry interval of the client.
$conf->set('reconnect.backoff.max.ms', 3000);

$topicConf = new RdKafka\\TopicConf();
#$topicConf->set('auto.commit.interval.ms', 100);
// Offset reset policy. Set it as appropriate according to the business scenario. I
$topicConf->set('auto.offset.reset', 'earliest');
$conf->setDefaultTopicConf($topicConf);

$consumer = new RdKafka\\KafkaConsumer($conf);
// Set it to `LOG_DEBUG` when you perform Debug.
//$consumer->setLogLevel(LOG_DEBUG);
$consumer->subscribe([$setting['topic_name']]);

$isConsuming = true;
while ($isConsuming) {
    $message = $consumer->consume(10 * 1000);
    switch ($message->err) {
        case RD_KAFKA_RESP_ERR_NO_ERROR:
            echo "**Consumer** receives a message:" . var_export($message, true) .
            break;
        case RD_KAFKA_RESP_ERR__PARTITION_EOF:
            echo "**Consumer** is waiting for messages\\n";
            break;
        case RD_KAFKA_RESP_ERR__TIMED_OUT:
            echo "**Consumer** waiting times out\\n";
            $isConsuming = false;
            break;
```

```
        default:
            throw new \\Exception($message->errstr(), $message->err);
            break;
    }
}
```

2. Run `Consumer.php` to consume the message.

```
php Consumer.php
```

3. View the execution result.

```
>**Consumer** receives a message: RdKafka\\Message::__set_state(array(
>   'err' => 0,
>   'topic_name' => 'topic_name',
>   'timestamp' => 1618800895159,
>   'partition' => 0,
>   'payload' => 'Message 0',
>   'len' => 9,
>   'key' => NULL,
>   'offset' => 0,
>   'headers' => NULL,
>))
>**Consumer** receives a message: RdKafka\\Message::__set_state(array(
>   'err' => 0,
>   'topic_name' => 'topic_name',
>   'timestamp' => 1618800895159,
>   'partition' => 0,
>   'payload' => 'Message 1',
>   'len' => 9,
>   'key' => NULL,
>   'offset' => 1,
>   'headers' => NULL,
>))
```

C++ SDK

Last updated：2024-09-09 21:27:55

Overview

This document introduces the directions for using the C++ client to connect to an elastic Topic in CKafka and send and receive messages.

Prerequisites

Install GCC

Directions

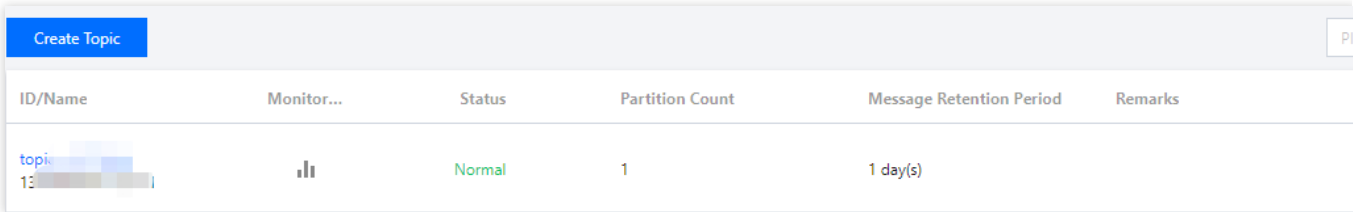
Step 1: Preparing the Environment

1. Install the C/C++ dependency library Install librdkafka.
2. Install the SSL/SASL dependencies.

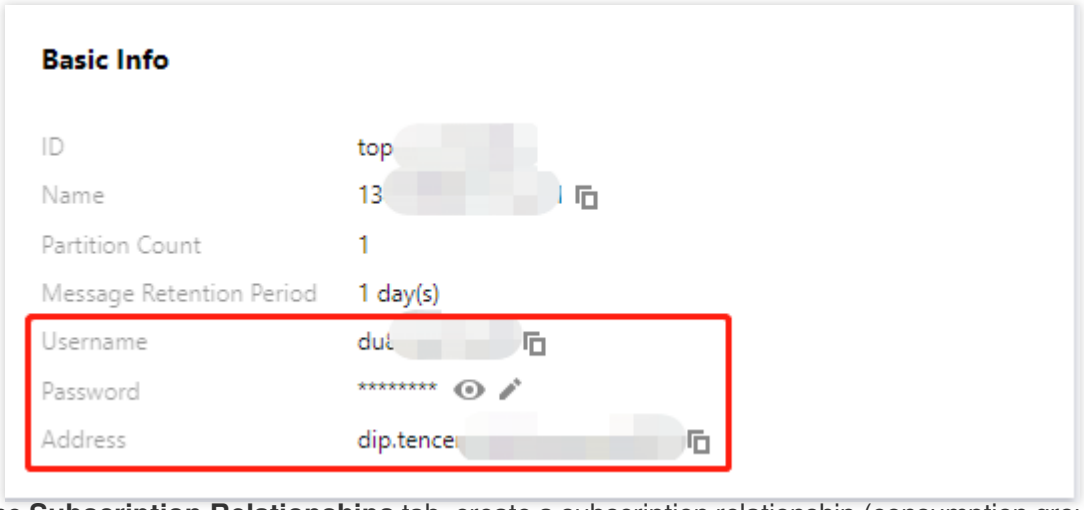
```
yum install openssl openssl-devel
yum install cyrus-sasl{,-plain}
```

Step 2: Creating a Topic and Subscription Relationship

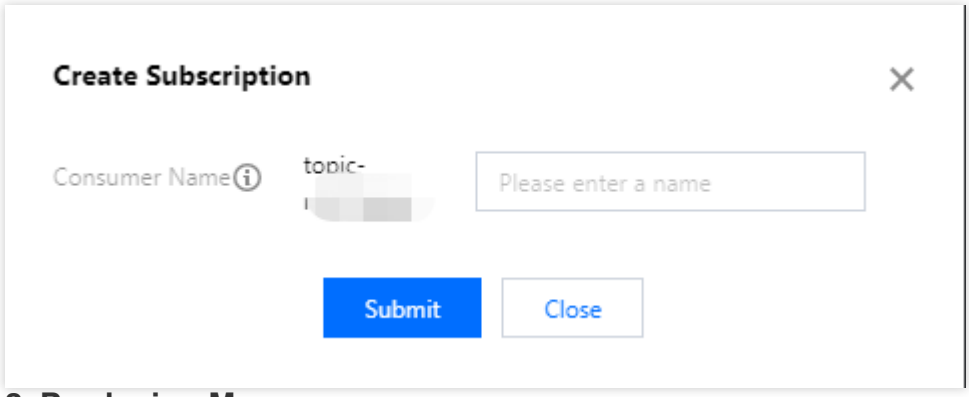
1. On the Elastic Topic list page of the console, create a Topic.



2. Click the **ID** of the Topic to enter the **Basic Information** page and obtain the username, password, and address information.



3. In the **Subscription Relationships** tab, create a subscription relationship (consumption group).



Step 3: Producing Messages

1. Create the producer.c file.

```
/*
 * librdkafka - Apache Kafka C library
 *
 * Copyright (c) 2017, Magnus Edenhill
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
```

```
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

/**
 * Simple Apache Kafka producer
 * using the Kafka driver from librdkafka
 * (https://github.com/edenhill/librdkafka)
 */

#include <stdio.h>
#include <signal.h>
#include <string.h>


#include <librdkafka/rdkafka.h>


static volatile sig_atomic_t run = 1;

/**
 * @brief Signal termination of program
 */
static void stop (int sig) {
    run = 0;
    fclose(stdin); /* abort fgets() */
}


/**
 * @brief Message delivery report callback.
 *
 * This callback is called exactly once per message, indicating if
 * the message was successfully delivered
 * (rkmessage->err == RD_KAFKA_RESP_ERR_NO_ERROR) or permanently
 * failed delivery (rkmessage->err != RD_KAFKA_RESP_ERR_NO_ERROR).
 *
 * The callback is triggered from rd_kafka_poll() and executes on
 * the application's thread.
 */
static void dr_msg_cb (rd_kafka_t *rk,
```

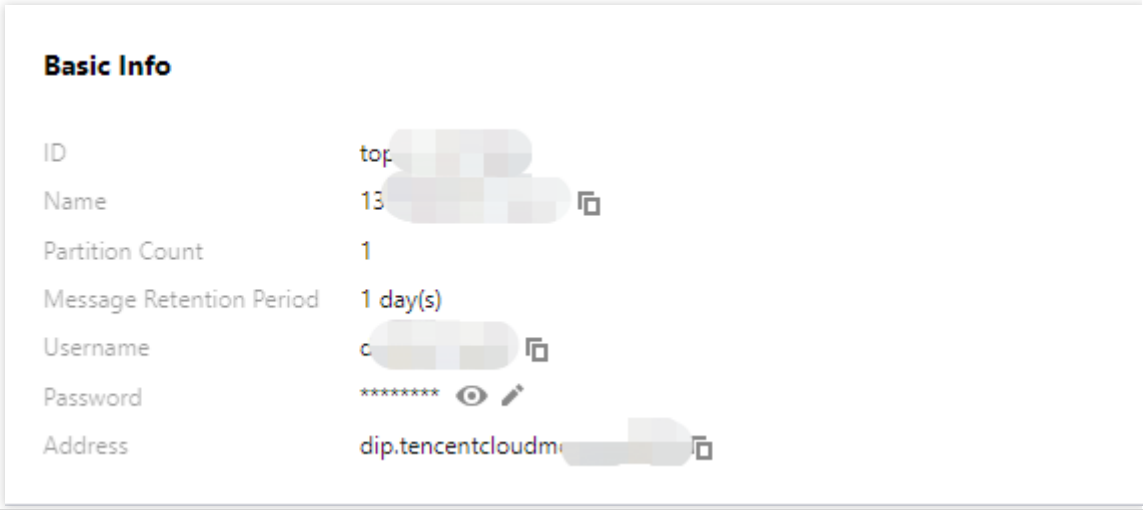
```
                      const rd_kafka_message_t *rkmessage, void *opaque) {
    if (rkmessage->err)
        fprintf(stderr, "%% Message delivery failed: %s\\n",
                rd_kafka_err2str(rkmessage->err));
    else
        fprintf(stderr,
                "%% Message delivered (%zd bytes, "
                "partition %"PRId32")\\n",
            rkmessage->len, rkmessage->partition);

    /* The rkmessage is destroyed automatically by librdkafka */
}



int main (int argc, char **argv) {
    rd_kafka_t *rk;         /* Producer instance handle */
    rd_kafka_conf_t *conf;  /* Temporary configuration object */
    char errstr[512];       /* librdkafka API error reporting buffer */
    char buf[512];          /* Message value temporary buffer */
    const char *brokers;    /* Argument: broker list */
    const char *topic;      /* Argument: topic to produce to */
    const char *user;       /* Argument: sasl username */
    const char *passwd;     /* Argument: sasl password */

    /*
     * Argument validation
     */
    if (argc < 3) {
        fprintf(stderr, "%% Usage: %s <broker> <topic> <username> <password>  \\n O
        return 1;
    }

    brokers = argv[1];
    topic   = argv[2];

    if(argc == 5) {
        user = argv[3];
        passwd = argv[4];
    }


    /*
     * Create Kafka client configuration place-holder
     */
    conf = rd_kafka_conf_new();
```

```
    /* Set bootstrap broker(s) as a comma-separated list of
     * host or host:port (default port 9092).
     * librdkafka will use the bootstrap brokers to acquire the full
     * set of brokers from the cluster. */
    if (rd_kafka_conf_set(conf, "bootstrap.servers", brokers,
                          errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK) {
        fprintf(stderr, "%s\\n", errstr);
        return 1;
    }

    /* Set sasl config*/
    if( user && passwd ) {
        if(rd_kafka_conf_set(conf,"security.protocol","sasl_plaintext",errstr,sizeo
            fprintf(stderr,"%s\\n",errstr);
            return 1;
        }
        if(rd_kafka_conf_set(conf, "sasl.mechanisms", "PLAIN", errstr, sizeof(errst
            fprintf(stderr,"%s\\n",errstr);
            return 1;
        }
        if(rd_kafka_conf_set(conf,"sasl.username",user,errstr,sizeof(errstr)) !=RD_
            fprintf(stderr,"%s\\n",errstr);
            return 1;

        }
        if(rd_kafka_conf_set(conf,"sasl.password",passwd,errstr,sizeof(errstr)) !=R
            fprintf(stderr,"%s\\n",errstr);
            return 1;
        }

    }
    /* Tencent Cloud recommended configuration parameters
      * https://cloud.tencent.com/document/product/597/30203
      */
    //if(rd_kafka_conf_set(conf,"debug","none",errstr,sizeof(errstr))!= RD_KAFKA_CO
      //  fprintf(stderr,"%s\\n",errstr);
     //   return 1;
//  }

    if(rd_kafka_conf_set(conf,"acks","1",errstr,sizeof(errstr)) != RD_KAFKA_CONF_OK
        fprintf(stderr,"%s\\n",errstr);
        return 1;
    }
    if(rd_kafka_conf_set(conf,"request.timeout.ms","30000",errstr,sizeof(errstr)) !
        fprintf(stderr,"%s\\n",errstr);
        return 1;
    }
```

```
    if(rd_kafka_conf_set(conf,"retries","3",errstr,sizeof(errstr)) != RD_KAFKA_CONF
        fprintf(stderr,"%s\\n",errstr);
        return 1;
    }

    if(rd_kafka_conf_set(conf,"retry.backoff.ms","1000",errstr,sizeof(errstr)) != R
        fprintf(stderr,"%s\\n",errstr);
        return 1;
    }

    /* Set the delivery report callback.
     * This callback will be called once per message to inform
     * the application if delivery succeeded or failed.
     * See dr_msg_cb() above.
     * The callback is only triggered from rd_kafka_poll() and
     * rd_kafka_flush(). */
    rd_kafka_conf_set_dr_msg_cb(conf, dr_msg_cb);

    /*
     * Create producer instance.
     *
     * NOTE: rd_kafka_new() takes ownership of the conf object
     *       and the application must not reference it again after
     *       this call.
     */
    rk = rd_kafka_new(RD_KAFKA_PRODUCER, conf, errstr, sizeof(errstr));
    if (!rk) {
        fprintf(stderr,
                "%% Failed to create new producer: %s\\n", errstr);
        return 1;
    }

    /* Signal handler for clean shutdown */
    signal(SIGINT, stop);

    fprintf(stderr,
            "%% Type some text and hit enter to produce message\\n"
            "%% Or just hit enter to only serve delivery reports\\n"
            "%% Press Ctrl-C or Ctrl-D to exit\\n");

    while (run && fgets(buf, sizeof(buf), stdin)) {
        size_t len = strlen(buf);
        rd_kafka_resp_err_t err;

        if (buf[len-1] == '\\n') /* Remove newline */
            buf[--len] = '\\0';
```

```
        if (len == 0) {
            /* Empty line: only serve delivery reports */
            rd_kafka_poll(rk, 0/*non-blocking */);
            continue;
        }

        /*
         * Send/Produce message.
         * This is an asynchronous call, on success it will only
         * enqueue the message on the internal producer queue.
         * The actual delivery attempts to the broker are handled
         * by background threads.
         * The previously registered delivery report callback
         * (dr_msg_cb) is used to signal back to the application
         * when the message has been delivered (or failed).
         */
        retry:
        err = rd_kafka_producev(
                /* Producer handle */
                rk,
                /* Topic name */
                RD_KAFKA_V_TOPIC(topic),
                /* Make a copy of the payload. */
                RD_KAFKA_V_MSGFLAGS(RD_KAFKA_MSG_F_COPY),
                /* Message value and length */
                RD_KAFKA_V_VALUE(buf, len),
                /* Per-Message opaque, provided in
                 * delivery report callback as
                 * msg_opaque. */
                RD_KAFKA_V_OPAQUE(NULL),
                /* End sentinel */
                RD_KAFKA_V_END);

        if (err) {
            /*
             * Failed to *enqueue* message for producing.
             */
            fprintf(stderr,
                    "%% Failed to produce to topic %s: %s\\n",
                    topic, rd_kafka_err2str(err));

            if (err == RD_KAFKA_RESP_ERR__QUEUE_FULL) {
                /* If the internal queue is full, wait for
                 * messages to be delivered and then retry.
                 * The internal queue represents both
                 * messages to be sent and messages that have
                 * been sent or failed, awaiting their
```

```
                 * delivery report callback to be called.
                 *
                 * The internal queue is limited by the
                 * configuration property
                 * queue.buffering.max.messages */
                rd_kafka_poll(rk, 1000/*block for max 1000ms*/);
                goto retry;
            }
        } else {
            fprintf(stderr, "%% Enqueued message (%zd bytes) "
                            "for topic %s\\n",
                    len, topic);
        }


        /* A producer application should continually serve
         * the delivery report queue by calling rd_kafka_poll()
         * at frequent intervals.
         * Either put the poll call in your main loop, or in a
         * dedicated thread, or call it after every
         * rd_kafka_produce() call.
         * Just make sure that rd_kafka_poll() is still called
         * during periods where you are not producing any messages
         * to make sure previously produced messages have their
         * delivery report callback served (and any other callbacks
         * you register). */
        rd_kafka_poll(rk, 0/*non-blocking*/);
    }


    /* Wait for final messages to be delivered or fail.
     * rd_kafka_flush() is an abstraction over rd_kafka_poll() which
     * waits for all messages to be delivered. */
    fprintf(stderr, "%% Flushing final messages..\\n");
    rd_kafka_flush(rk, 10*1000 /* wait for max 10 seconds */);

    /* If the output queue is still not empty there is an issue
     * with producing messages to the clusters. */
    if (rd_kafka_outq_len(rk) > 0)
        fprintf(stderr, "%% %d message(s) were not delivered\\n",
                rd_kafka_outq_len(rk));

    /* Destroy the producer instance */
    rd_kafka_destroy(rk);

    return 0;
}
```

Parameter	Description
broker	The connection address. It can be obtained from the basic information page of an elastic Topic in the 
username	The username. It can be obtained from the basic information page of an elastic Topic in the console.
password	The user password. It can be obtained from the basic information page of an elastic Topic in the cons
topic	The topic name. It can be obtained from the basic information page of an elastic Topic in the console.

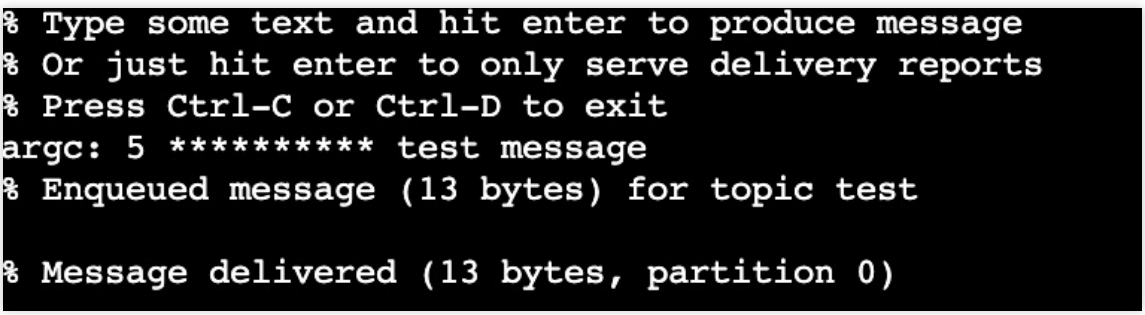
2. Run the following command to compile producer.c.

```
gcc -lrdkafka ./producer.c -o producer
```

3. Run the following command to send the message.

```
./produce <broker> <topic> <username> <password>
```

4. The operation result is as follows:



Step 3: Consuming Messages

1. Create the consumer.c file.

```
/*
```

```
 * librdkafka - Apache Kafka C library
 *
 * Copyright (c) 2019, Magnus Edenhill
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
*/

/**
 * Simple high-level balanced Apache Kafka consumer
 * using the Kafka driver from librdkafka
 * (https://github.com/edenhill/librdkafka)
 */

#include <stdio.h>
#include <signal.h>
#include <string.h>
#include <ctype.h>
#include <librdkafka/rdkafka.h>


static volatile sig_atomic_t run = 1;

/**
* @brief Signal termination of program
*/
static void stop (int sig) {
    run = 0;
```

```
}

/**
 * @returns 1 if all bytes are printable, else 0.
*/
static int is_printable (const char *buf, size_t size) {
    size_t i;

    for (i = 0 ; i < size ; i++)
            if (!isprint((int)buf[i]))
                    return 0;

    return 1;
}


int main (int argc, char **argv) {
    rd_kafka_t *rk;          /* Consumer instance handle */
    rd_kafka_conf_t *conf;   /* Temporary configuration object */
    rd_kafka_resp_err_t err; /* librdkafka API error code */
    char errstr[512];        /* librdkafka API error reporting buffer */
    const char * user;           /*Argument: sasl username*/
    const char * password;   /*Argument: sasl password*/
    const char *brokers;     /* Argument: broker list */
    const char *groupid;     /* Argument: Consumer group id */
    char **topics;           /* Argument: list of topics to subscribe to */
    int topic_cnt;           /* Number of topics to subscribe to */
    rd_kafka_topic_partition_list_t *subscription; /* Subscribed topics */
    int i;

    /*
    * Argument validation
    */
    if (argc < 6) {
        fprintf(stderr,
                "%% Usage: "
                "%s <broker> <group.id> <username> <password> <topic1> <topic2>..\\
                argv[0]);
        return 1;
    }

    brokers   = argv[1];
    groupid   = argv[2];
    user      = argv[3];
    password  = argv[4];
    topics    = &argv[5];
    topic_cnt = argc - 5;
```

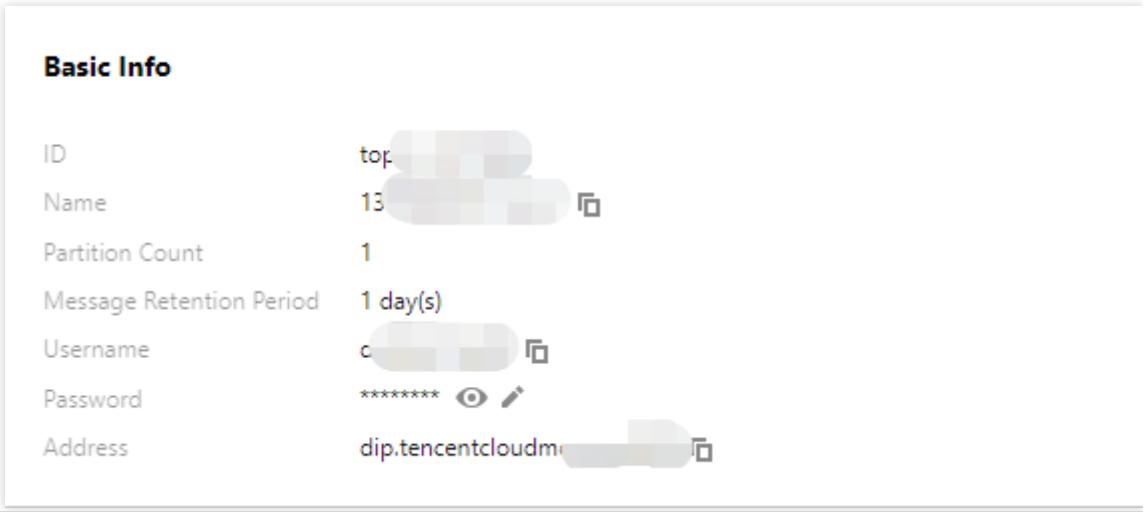
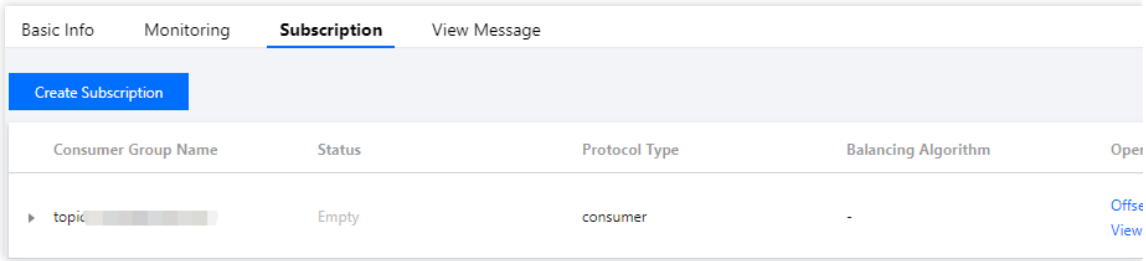
```
/*
* Create Kafka client configuration place-holder
*/
conf = rd_kafka_conf_new();

/* Set bootstrap broker(s) as a comma-separated list of
* host or host:port (default port 9092).
* librdkafka will use the bootstrap brokers to acquire the full
* set of brokers from the cluster. */
if (rd_kafka_conf_set(conf, "bootstrap.servers", brokers,
        errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK) {
    fprintf(stderr, "%s\\n", errstr);
    rd_kafka_conf_destroy(conf);
    return 1;
}

/* Set the consumer group id.
* All consumers sharing the same group id will join the same
* group, and the subscribed topic' partitions will be assigned
* according to the partition.assignment.strategy
* (consumer config property) to the consumers in the group. */
if (rd_kafka_conf_set(conf, "group.id", groupid,
                          errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK) {
    fprintf(stderr, "%s\\n", errstr);
    rd_kafka_conf_destroy(conf);
    return 1;
}

/* If there is no previously committed offset for a partition
* the auto.offset.reset strategy will be used to decide where
* in the partition to start fetching messages.
* By setting this to earliest the consumer will read all messages
* in the partition if there was no previously committed offset. */
if (rd_kafka_conf_set(conf, "auto.offset.reset", "earliest",
                          errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK) {
    fprintf(stderr, "%s\\n", errstr);
    rd_kafka_conf_destroy(conf);
    return 1;
}
/* Tencent Cloud recommended configuration parameters
* https://cloud.tencent.com/document/product/597/30203
*/
if(rd_kafka_conf_set(conf,"debug","all",errstr,sizeof(errstr))!= RD_KAFKA_CONF_
    fprintf(stderr,"%s\\n",errstr);
    return 1;
```

```
    }

    if(rd_kafka_conf_set(conf,"session.timeout.ms","10000",errstr,sizeof(errstr)) !
        fprintf(stderr,"%s\\n",errstr);
        return 1;
    }
    if(rd_kafka_conf_set(conf,"heartbeat.interval.ms","3000",errstr,sizeof(errstr))
        fprintf(stderr,"%s\\n",errstr);
        return 1;
    }

    /*Set sasl config*/
    if(rd_kafka_conf_set(conf,"security.protocol","sasl_plaintext",errstr,sizeof(er
        fprintf(stderr,"%s\\n",errstr);
        return 1;
    }
    if(rd_kafka_conf_set(conf, "sasl.mechanisms", "PLAIN", errstr, sizeof(errstr))
        fprintf(stderr,"%s\\n",errstr);
        return 1;
    }
    if(rd_kafka_conf_set(conf,"sasl.username",user,errstr,sizeof(errstr)) !=RD_KAFK
        fprintf(stderr,"%s\\n",errstr);
        return 1;
    }
    if(rd_kafka_conf_set(conf,"sasl.password",password,errstr,sizeof(errstr)) !=RD_
        fprintf(stderr,"%s\\n",errstr);
        return 1;
    }
    /*
    * Create consumer instance.
    *
    * NOTE: rd_kafka_new() takes ownership of the conf object
    *       and the application must not reference it again after
    *       this call.
    */
    rk = rd_kafka_new(RD_KAFKA_CONSUMER, conf, errstr, sizeof(errstr));
    if (!rk) {
        fprintf(stderr,
                "%% Failed to create new consumer: %s\\n", errstr);
        return 1;
    }

    conf = NULL; /* Configuration object is now owned, and freed,
                      * by the rd_kafka_t instance. */


    /* Redirect all messages from per-partition queues to
```

```
* the main queue so that messages can be consumed with one
* call from all assigned partitions.
*
* The alternative is to poll the main queue (for events)
* and each partition queue separately, which requires setting
* up a rebalance callback and keeping track of the assignment:
* but that is more complex and typically not recommended. */
rd_kafka_poll_set_consumer(rk);


/* Convert the list of topics to a format suitable for librdkafka */
subscription = rd_kafka_topic_partition_list_new(topic_cnt);
for (i = 0 ; i < topic_cnt ; i++)
    rd_kafka_topic_partition_list_add(subscription,
                                      topics[i],
                                      /* the partition is ignored
                                       * by subscribe() */
                                      RD_KAFKA_PARTITION_UA);

/* Subscribe to the list of topics */
err = rd_kafka_subscribe(rk, subscription);
if (err) {
    fprintf(stderr,
            "%% Failed to subscribe to %d topics: %s\\n",
            subscription->cnt, rd_kafka_err2str(err));
    rd_kafka_topic_partition_list_destroy(subscription);
    rd_kafka_destroy(rk);
    return 1;
}

fprintf(stderr,
            "%% Subscribed to %d topic(s), "
            "waiting for rebalance and messages...\\n",
            subscription->cnt);

rd_kafka_topic_partition_list_destroy(subscription);


/* Signal handler for clean shutdown */
signal(SIGINT, stop);

/* Subscribing to topics will trigger a group rebalance
* which may take some time to finish, but there is no need
* for the application to handle this idle period in a special way
* since a rebalance may happen at any time.
* Start polling for messages. */
```

```
    while (run) {
        rd_kafka_message_t *rkm;

        rkm = rd_kafka_consumer_poll(rk, 100);
        if (!rkm)
            continue; /* Timeout: no message within 100ms,
            *  try again. This short timeout allows
            *  checking for `run` at frequent intervals.
            */

            /* consumer_poll() will return either a proper message
            * or a consumer error (rkm->err is set). */
            if (rkm->err) {
                /* Consumer errors are generally to be considered
                * informational as the consumer will automatically
                * try to recover from all types of errors. */
                fprintf(stderr,
                        "%% Consumer error: %s\\n",
                        rd_kafka_message_errstr(rkm));
                rd_kafka_message_destroy(rkm);
                continue;
            }

            /* Proper message. */
            printf("Message on %s [%"PRId32"] at offset %"PRId64":\\n",
                    rd_kafka_topic_name(rkm->rkt), rkm->partition,
                    rkm->offset);

            /* Print the message key. */
            if (rkm->key && is_printable(rkm->key, rkm->key_len))
                printf(" Key: %.*s\\n",
                        (int)rkm->key_len, (const char *)rkm->key);
            else if (rkm->key)
                printf(" Key: (%d bytes)\\n", (int)rkm->key_len);

           /* Print the message value/payload. */
           if (rkm->payload && is_printable(rkm->payload, rkm->len))
                    printf(" Value: %.*s\\n",
                            (int)rkm->len, (const char *)rkm->payload);
           else if (rkm->payload)
               printf(" Value: (%d bytes)\\n", (int)rkm->len);

           rd_kafka_message_destroy(rkm);
    }


    /* Close the consumer: commit final offsets and leave the group. */
```

```
    fprintf(stderr, "%% Closing consumer\\n");
    rd_kafka_consumer_close(rk);


    /* Destroy the consumer */
    rd_kafka_destroy(rk);

    return 0;
}
```

Parameter	Description
broker	The connection address. It can be obtained from the basic information page of an elastic Topic in the 
username	The username. It can be obtained from the basic information page of an elastic Topic in the console.
password	The user password. It can be obtained from the basic information page of an elastic Topic in the cons
topic	The topic name. It can be obtained from the basic information page of an elastic Topic in the console.
group.id	The consumption group name. It can be obtained from the subscription relationship list of an elastic T 

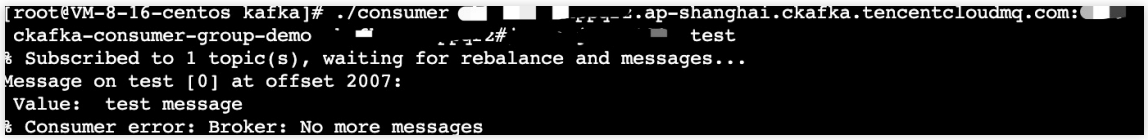
2. Run the following command to compile consumer.c.

```
gcc -lrdkafka ./consumer.c -o consumer
```

3. Run the following command to send the message.

```
./consumer <broker> <group.id> <username> <password> <topic>.
```

4. The execution result is as follows:



Node.js SDK

Last updated：2024-09-09 21:28:39

Overview

This document introduces the directions for using a Node.js client to connect to an elastic Topic of CKafka and send and receive messages.

Prerequisites

Install GCC
Install Node.js

Directions

Step 1: Preparing the Environment

Installing the C++ Dependency Database

1. Run the following command to switch to the Yum repository configuration directory `/etc/yum.repos.d/` .

```
cd /etc/yum.repos.d/
```

2. Create the Yum repository configuration file confluent.repo.

```
[Confluent.dist]
name=Confluent repository (dist)
baseurl=https://packages.confluent.io/rpm/5.1/7
gpgcheck=1
gpgkey=https://packages.confluent.io/rpm/5.1/archive.key
enabled=1
[Confluent]
name=Confluent repository
baseurl=https://packages.confluent.io/rpm/5.1
gpgcheck=1
gpgkey=https://packages.confluent.io/rpm/5.1/archive.key
enabled=1
```

3. Run the following command to install the C++ dependency database.

```
yum install librdkafka-devel
```

Installing the Node.js Dependency Database

1. Run the following command to specify the OpenSSL header file path for the preprocessor.

```
export CPPFLAGS=-I/usr/local/opt/openssl/include
```

2. Run the following command to specify the OpenSSL database path for the connector.

```
export LDFLAGS=-L/usr/local/opt/openssl/lib
```

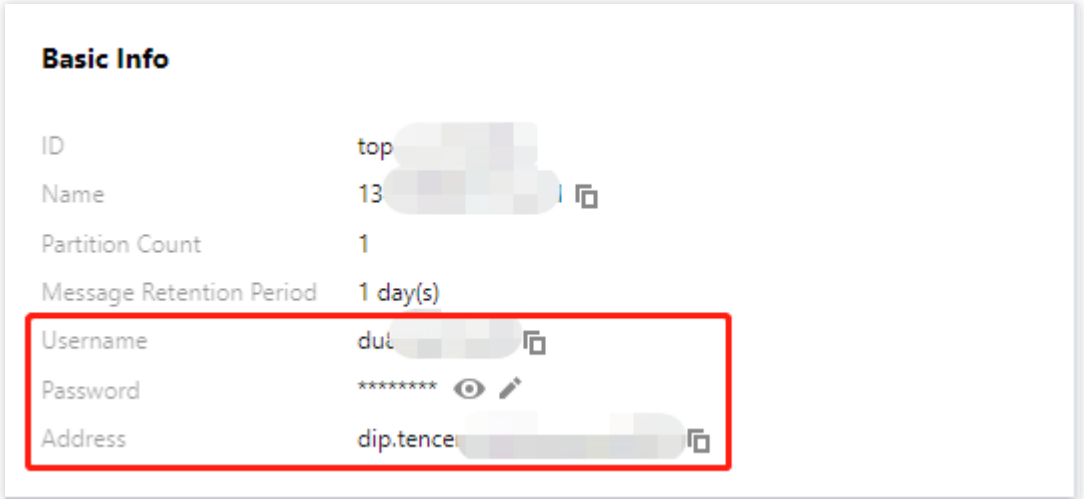
3. Run the following command to install the Node.js dependency database.

```
npm install i --unsafe-perm node-rdkafka
```

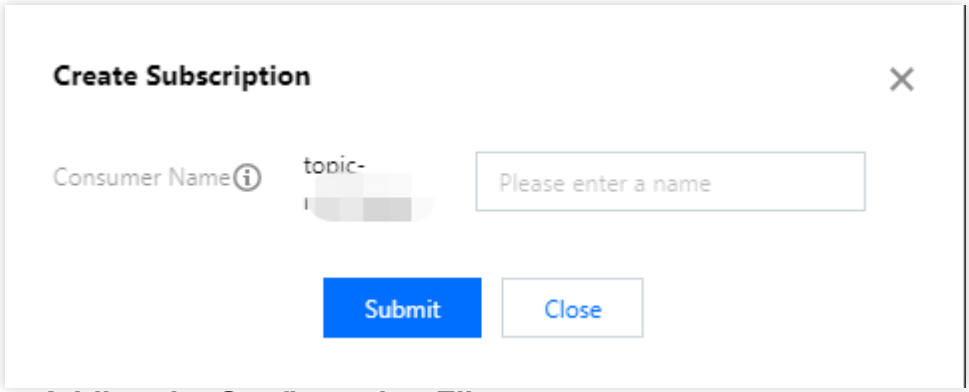
Step 2: Creating a Topic and Subscription Relationship

1. On the Elastic Topic list page of the console, create a Topic.

2. Click the **ID** of the Topic to enter the **Basic Information** page and obtain the username, password, and address information.

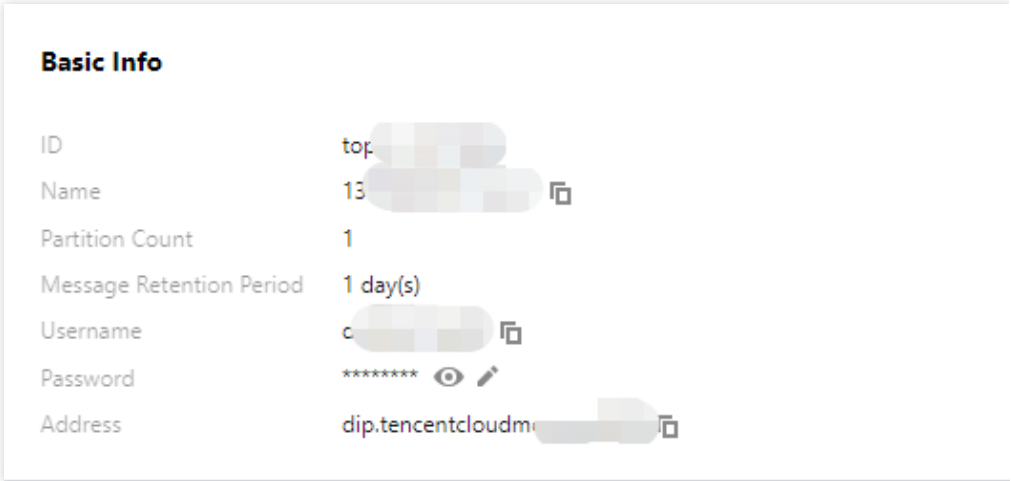


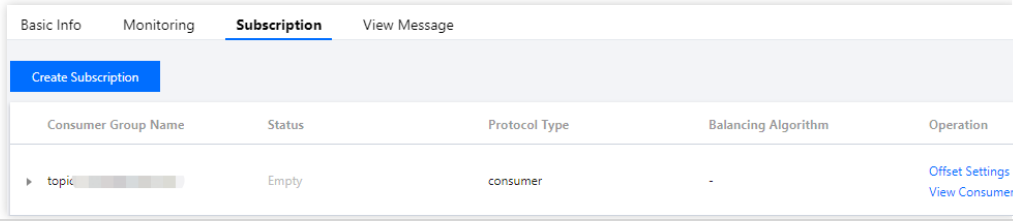
3. In the **Subscription Relationships** tab, create a subscription relationship (consumption group).



Step 2: Adding the Configuration File

```
module.exports = {
    'sasl_plain_username': 'your_user_name',
    'sasl_plain_password': 'your_user_password',
    'bootstrap_servers': ["xxx.xx.xx.xx:port"],
    'topic_name': 'xxx',
    'group_id': 'xxx'
}
```

Parameter	Description
bootstrapServers	The connection address. It can be obtained from the basic information page of an elastic T console. 
sasl_plain_username	The username. It can be obtained from the basic information page of an elastic Topic in th
sasl_plain_password	The user password. It can be obtained from the basic information page of an elastic Topic
topic_name	The topic name. It can be obtained from the basic information page of an elastic Topic in t
group.id	The consumption group name. It can be obtained from the subscription relationship list of a in the console.



Step 3: Producing Messages

1. Write the message production program producer.js.

```
const Kafka = require('node-rdkafka');
const config = require('./setting');
console.log("features:" + Kafka.features);
console.log(Kafka.librdkafkaVersion);

var producer = new Kafka.Producer({
    'api.version.request': 'true',
    'bootstrap.servers': config['bootstrap_servers'],
    'dr_cb': true,
    'dr_msg_cb': true,
    'security.protocol' : 'SASL_PLAINTEXT',
    'sasl.mechanisms' : 'PLAIN',
    'sasl.username' : config['sasl_plain_username'],
    'sasl.password' : config['sasl_plain_password']
});

var connected = false

producer.setPollInterval(100);

producer.connect();

producer.on('ready', function() {
connected = true
console.log("connect ok")

});

function produce() {
try {
    producer.produce(
    config['topic_name'],
    new Buffer('Hello CKafka SASL'),
    null,
    Date.now()
    );
```

```
    } catch (err) {
        console.error('Error occurred when sending message(s)');
        console.error(err);
    }
    }

    producer.on("disconnected", function() {
    connected = false;
    producer.connect();
    })

    producer.on('event.log', function(event) {
        console.log("event.log", event);
    });

    producer.on("error", function(error) {
        console.log("error:" + error);
    });

    producer.on('delivery-report', function(err, report) {
        console.log("delivery-report: producer ok");
    });
    // Any errors we encounter, including connection errors
    producer.on('event.error', function(err) {
        console.error('event.error:' + err);
    })

    setInterval(produce,1000,"Interval");
```

2. Run the following command to send the message.

```
node producer.js
```

3. View the execution result.

Step 4: Consuming Messages

1. Create the message consumption program consumer.js.

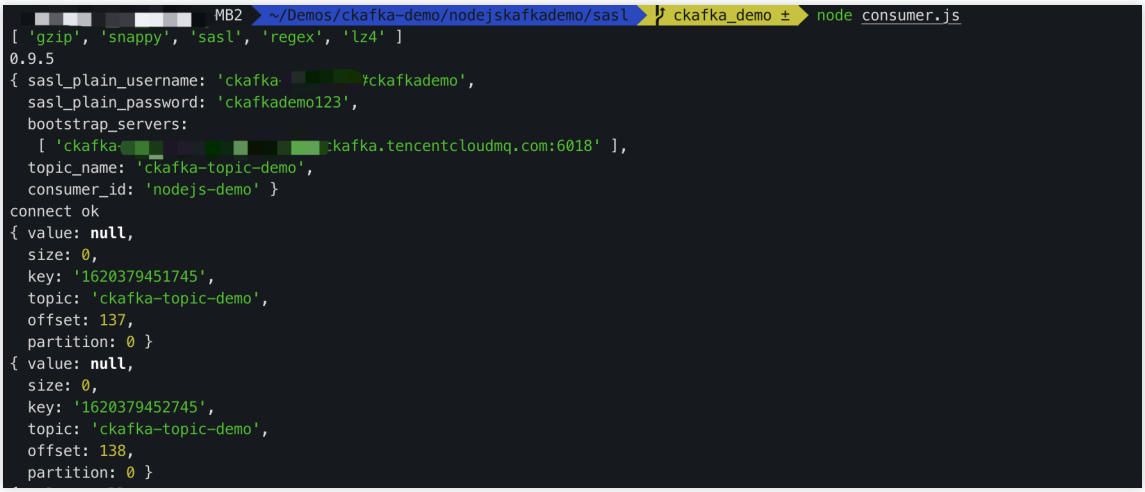
```
 consumer.on('event.log', function(event) {
    console.log("event.log", event);
});

consumer.on('error', function(error) {
    console.log("error:" + error);
});

consumer.on('event', function(event) {
        console.log("event:" + event);
});
const Kafka = require('node-rdkafka');
const config = require('./setting');
console.log(Kafka.features);
console.log(Kafka.librdkafkaVersion);
console.log(config)

var consumer = new Kafka.KafkaConsumer({
    'api.version.request': 'true',
    'bootstrap.servers': config['bootstrap_servers'],
    'security.protocol' : 'SASL_PLAINTEXT',
    'sasl.mechanisms' : 'PLAIN',
    'message.max.bytes': 32000,
    'fetch.message.max.bytes': 32000,
    'max.partition.fetch.bytes': 32000,
    'sasl.username' : config['sasl_plain_username'],
    'sasl.password' : config['sasl_plain_password'],
    'group.id' : config['group_id']
});

consumer.connect();
```

```
consumer.on('ready', function() {
console.log("connect ok");
consumer.subscribe([config['topic_name']]);
consumer.consume();
})

consumer.on('data', function(data) {
console.log(data);
});

consumer.on('event.log', function(event) {
    console.log("event.log", event);
});

consumer.on('error', function(error) {
    console.log("error:" + error);
});

consumer.on('event', function(event) {
        console.log("event:" + event);
});
```

2. Run the following command to consume the message.

```
node consumer.js
```

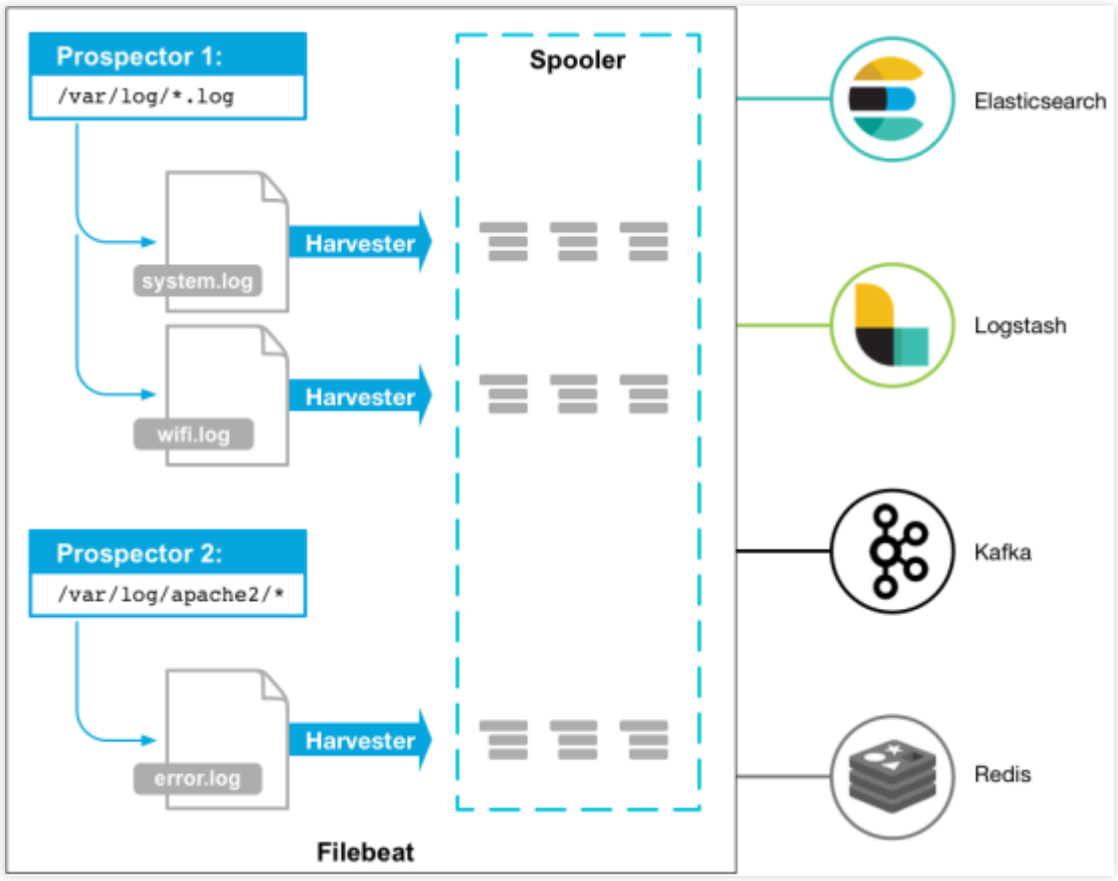
3. View the execution result.



Connecting Filebeats to CKafka

Last updated：2024-09-09 21:29:32

Beats platform integrates a variety of single-purpose data collectors. Once installed, these collectors can serve as lightweight agents, sending collected data from hundreds or thousands of machines to a specified target.



Beats offers multiple collectors, and you can download the appropriate collector according to your needs. This document will introduce how to connect Filebeat (a lightweight log collector) to CKafka and solutions to FAQs encountered after connection.

Prerequisites

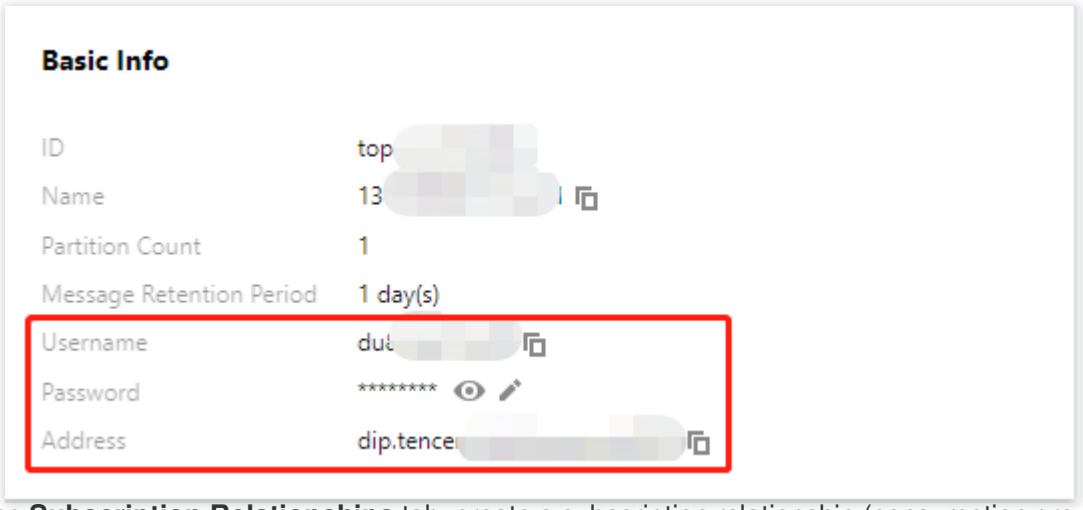
Download and install Filebeat (see Download Filebeat)
Download and install JDK 8 (see Download JDK 8)

Directions

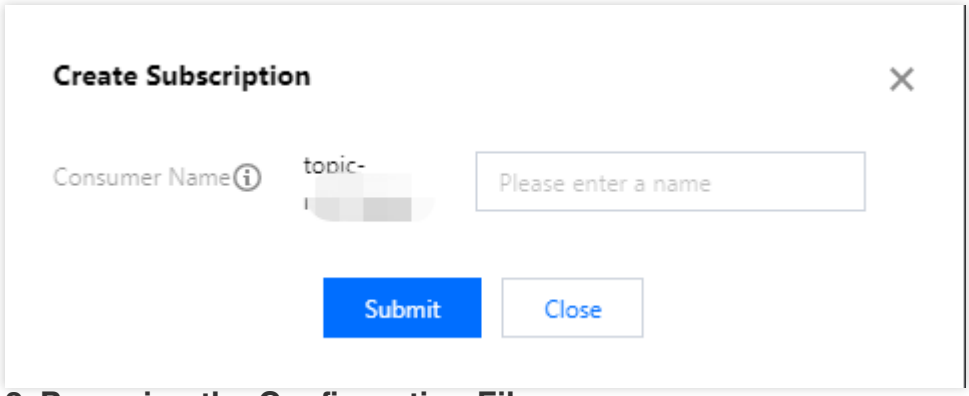
Step 1: Preparations

1. On the Elastic Topic list page of the console, create a Topic.

2. Click the **ID** of the Topic to enter the **Basic Information** page and obtain the username, password, and address information.



3. In the **Subscription Relationships** tab, create a subscription relationship (consumption group).

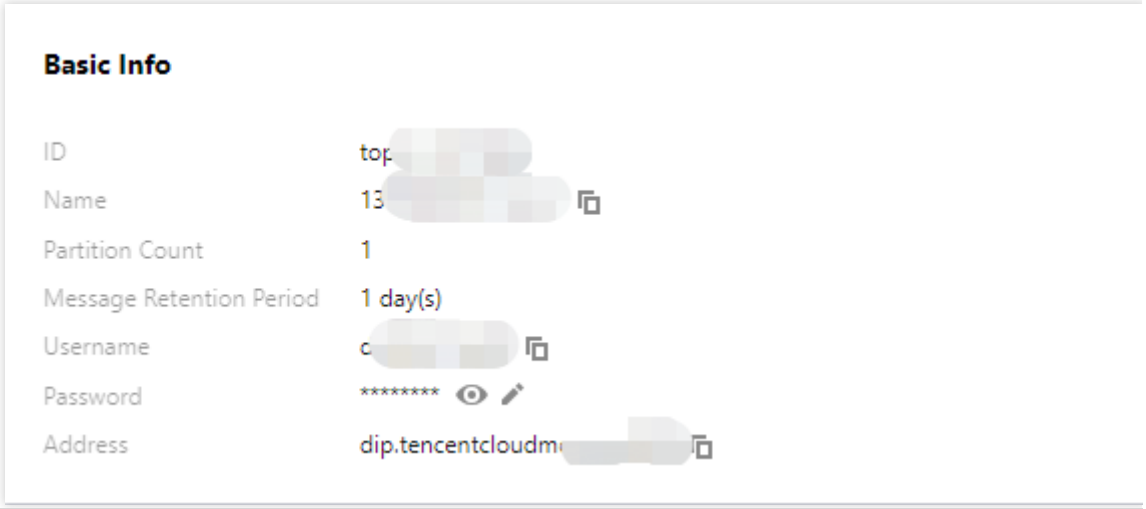


Step 2: Preparing the Configuration File

Enter the Filebeat installation directory and create the configuration monitoring file named filebeat.yml.

```
#======= For versions later than Filebeat 7.x, change `filebeat.prospectors` to `fi
filebeat.prospectors:


- input_type: log
```

```
# This is the monitoring file path.
  paths:
    - /var/log/messages

#=======  Outputs =========

#------------------ kafka -------------------------------------
output.kafka:
  version:0.10.2 // Configure them according to different CKafka instance open-sour
  # Set the CKafka connection address.
  hosts: ["xx.xx.xx.xx:xxxx"]
  # Set the name of the target topic.
  topic: 'test'
  partition.round_robin:
    reachable_only: false

  required_acks: 1
  compression: none
  max_message_bytes: 1000000

  # SASL requires the following information to be configured. If SASL is not needed
  username: "yourusername"
  password: "yourpassword"
```

Parameter	Description
host	The connection address. It can be obtained from the basic information page of an elastic Topic in the 
username	The username. It can be obtained from the basic information page of an elastic Topic in the console.
password	The user password. It can be obtained from the basic information page of an elastic Topic in the cons
topic	The topic name. It can be obtained from the basic information page of an elastic Topic in the console.

Step 3: Sending Messages with Filebeat

1. Run the following command to start the client.

```
sudo ./filebeat -e -c filebeat.yml
```

2. Add data to the monitoring file (example: write to the monitored testlog file).

```
    echo ckafka1 >> testlog
    echo ckafka2 >> testlog
    echo ckafka3 >> testlog
```

3. Start the Consumer to consume the corresponding Topic and get the following data.

```
    {"@timestamp":"2017-09-29T10:01:27.936Z","beat":{"hostname":"10.193.9.26","name
    {"@timestamp":"2017-09-29T10:01:30.936Z","beat":{"hostname":"10.193.9.26","name
    {"@timestamp":"2017-09-29T10:01:33.937Z","beat":{"hostname":"10.193.9.26","name
```

SASL/PLAINTEXT Mode

If you want to configure SASL/PLAINTEXT, you need to set the username and password under the Kafka configuration.

```
# SASL requires the following information to be configured. If SASL is not needed,
 username: "yourusername"
 password: "yourpassword"
```

FAQs

If you find a large number of INFO logs in the Filebeat logs (default path `/var/log/filebeat/filebeat`), such as:

```
2019-03-20T08:55:02.198+0800    INFO    kafka/log.go:53 producer/broker/544 startin
2019-03-20T08:55:02.198+0800    INFO    kafka/log.go:53 producer/broker/544 state c
2019-03-20T08:55:02.198+0800    INFO    kafka/log.go:53 producer/leader/wp-news-fil
2019-03-20T08:55:02.198+0800    INFO    kafka/log.go:53 producer/broker/478 state c
2019-03-20T08:55:02.199+0800    INFO    kafka/log.go:53 Closed connection to broker
2019-03-20T08:55:02.199+0800    INFO    kafka/log.go:53 producer/leader/wp-news-fil
2019-03-20T08:55:02.199+0800    INFO    kafka/log.go:53 producer/leader/wp-news-fil
2019-03-20T08:55:02.199+0800    INFO    kafka/log.go:53 producer/leader/wp-news-fil
2019-03-20T08:55:02.199+0800    INFO    kafka/log.go:53 producer/leader/wp-news-fil
2019-03-20T08:55:02.199+0800    INFO    kafka/log.go:53 producer/leader/wp-news-fil
2019-03-20T08:55:02.199+0800    INFO    kafka/log.go:53 producer/leader/wp-news-fil
2019-03-20T08:55:02.199+0800    INFO    kafka/log.go:53 producer/broker/478 shut do
```

A large number of INFO logs may indicate a potential issue with the Filebeat version. Products in the Elastic family are updated frequently, and different major versions often have compatibility issues.
For example, v6.5.x supports Kafka v0.9, v0.10, v1.1.0, and v2.0.0 by default, while v5.6.x supports Kafka v0.8.2.0 by default.
You should check the version configuration in the configuration file:

```
output.kafka:
  version:0.10.2 // Configure them according to different CKafka instance open-sour
```

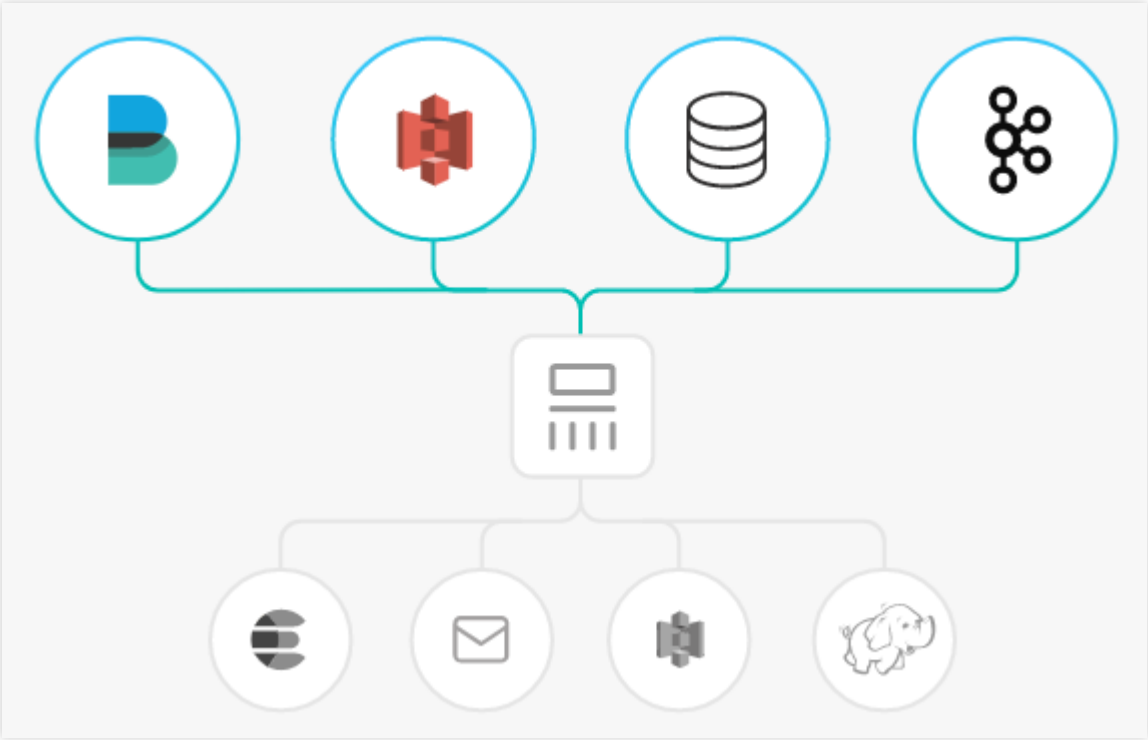
Connecting Logstash to CKafka

Last updated：2024-09-09 21:30:22

Logstash is an open-source log processing tool that can be used to collect data from multiple sources, filter the collected data, and store it for various uses.
Logstash is highly flexible and has powerful syntax analysis capabilities. With a variety of plugins, it supports multiple input and output sources. In addition, as a horizontally scalable data pipeline, Logstash works seamlessly with Elasticsearch and Kibana, offering robust functionality in log collection and search.

How Logstash Works

The Logstash data processing can be divided into three stages: inputs → filters → outputs.
1. Inputs: Collect data from multiple sources like file, syslog, redis, and beats.
2. Filters: Modify and filter the collected data. Filters are intermediate processing components in the Logstash data pipeline. They can modify events based on specific conditions. Some commonly used filters are grok, mutate, drop, and clone.
3. Outputs: Transfer the processed data to other destinations. An event can be transferred to multiple outputs, and the event ends when the transfer is completed. Elasticsearch is the most commonly used outputs.
In addition, Logstash supports encoding and decoding data, so you can specify data formats on the input and output ends.



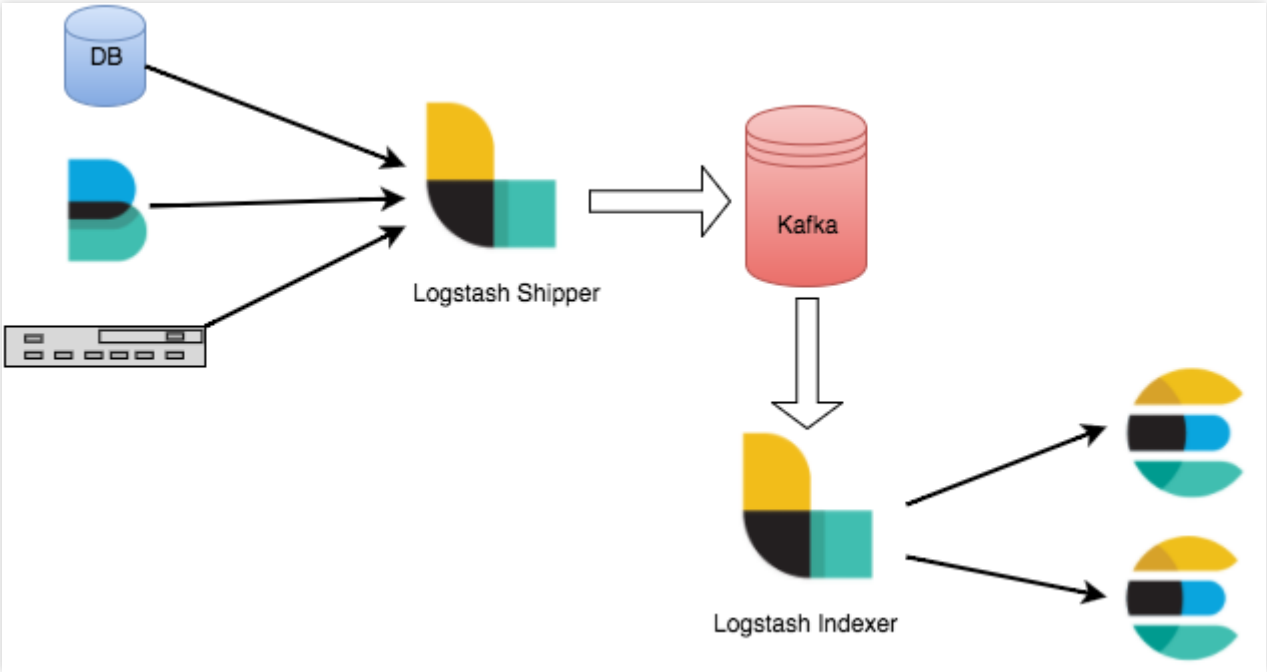
Advantages of Connecting Logstash to Kafka

Asynchronous data processing: It can prevent traffic spikes.

Decoupling: When an exception occurs in Elasticsearch, the upstream work will not be affected.

Note

Logstash consumes resources when filtering data. If you deploy Logstash on a production server, the performance of the server may be affected.



Prerequisites

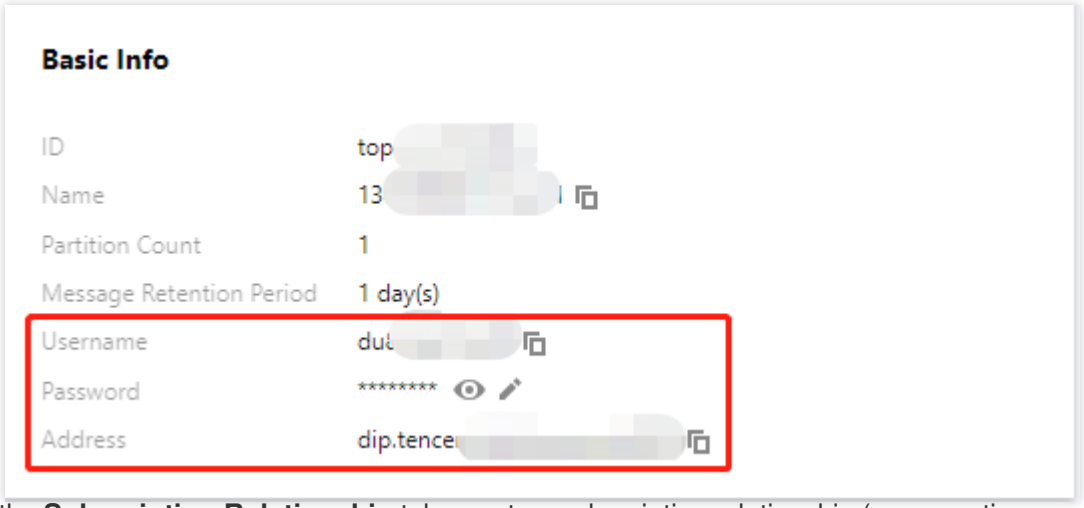
Download and install Logstash, see Download Logstash.
Download and install JDK 8, see Download JDK 8.

Directions

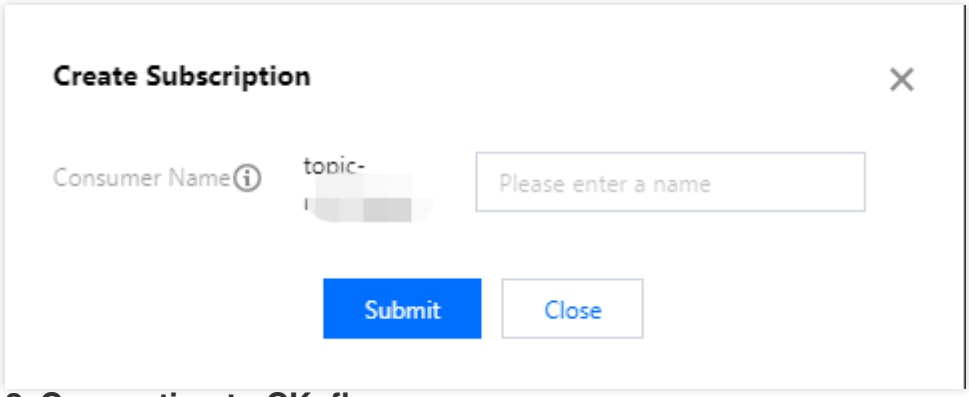
Step 1: Preparations

1. On the Elastic Topic list page of the console, create a Topic.

2. Click the **ID** of the Topic to enter the **Basic Information** page, and obtain username, password, and address information.



3. On the **Subscription Relationship** tab, create a subscription relationship (consumption group).



Step 2: Connecting to CKafka

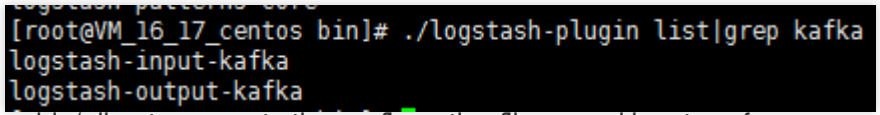
Note

You can click the tabs below to view the specific steps for connecting CKafka as inputs or outputs.

Connecting as Inputs

Connecting as Outputs

1. Run `bin/logstash-plugin list` to check if the supported plugins include logstash-input-kafka.



2. In the .bin/ directory, create the configuration file named input.conf.

Here, standard output serves as the data destination and Kafka serves as the data source. The kafka-client-jaas.conf file is used for configuring SASL-PLAINTEXT username and password.

```
input {
    kafka {
        bootstrap_servers => "xx.xx.xx.xx:xxxx" // Ckafka connection address
        group_id => "logstash_group"  // CKafka group ID
        topics => ["logstash_test"] // CKafka topic name
```

```
            consumer_threads => 3 // Number of consumption threads, which is general
            auto_offset_reset => "earliest"
            security_protocol => "SASL_PLAINTEXT"
            sasl_mechanism => "PLAIN"
            jaas_path => "xx/xx/kafka-client-jaas.conf"
        }
    }
    output {
        stdout{codec=>rubydebug}
    }
```

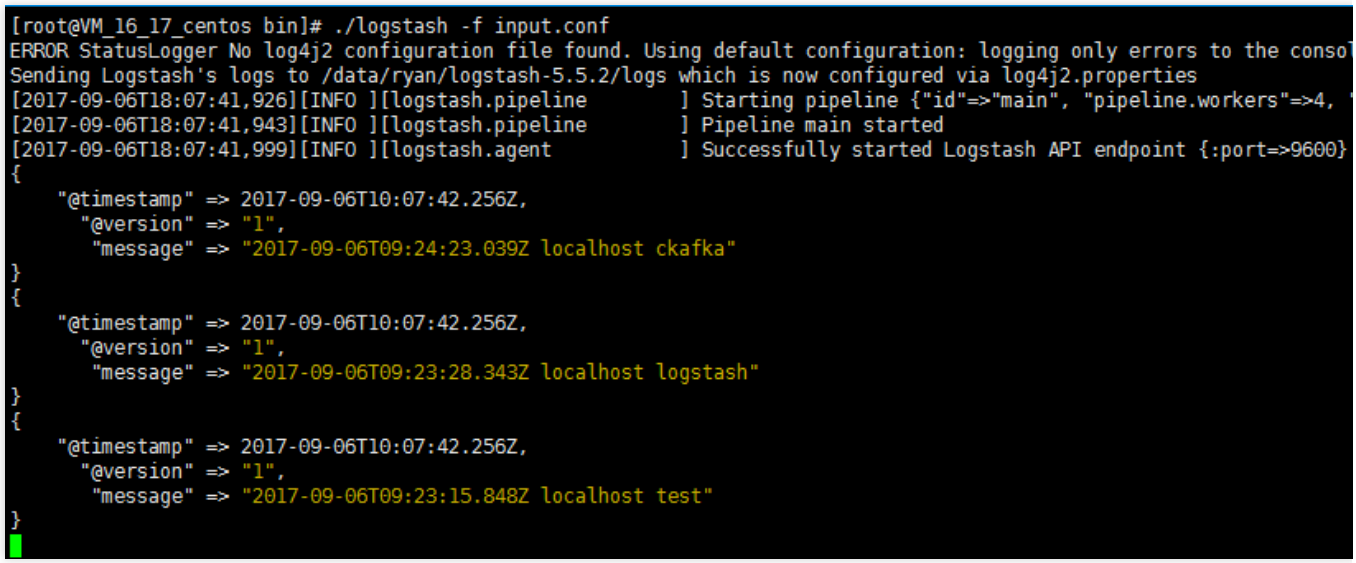
The content of kafka-client-jaas.conf is as follows:

```
    KafkaClient {
        org.apache.kafka.common.security.plain.PlainLoginModule required
        username="username"
        password="password";
    };
```

3. Run the following command to start Logstash and begin message consumption:

```
./logstash -f input.conf
```

The returned result is as follows:



You can see the data from the Topic has been consumed.

1. Run `bin/logstash-plugin list` to check if the supported plugins include logstash-output-kafka.

```
[root@VM_16_17_centos bin]# ./logstash-plugin list|grep kafka
logstash-input-kafka
logstash-output-kafka
```

2. Write a configuration file output.conf in the .bin/ directory.

Here, standard input serves as the data source and Kafka serves as the data destination. The kafka-client-jaas.conf file is used for configuring SASL-PLAINTEXT username and password.

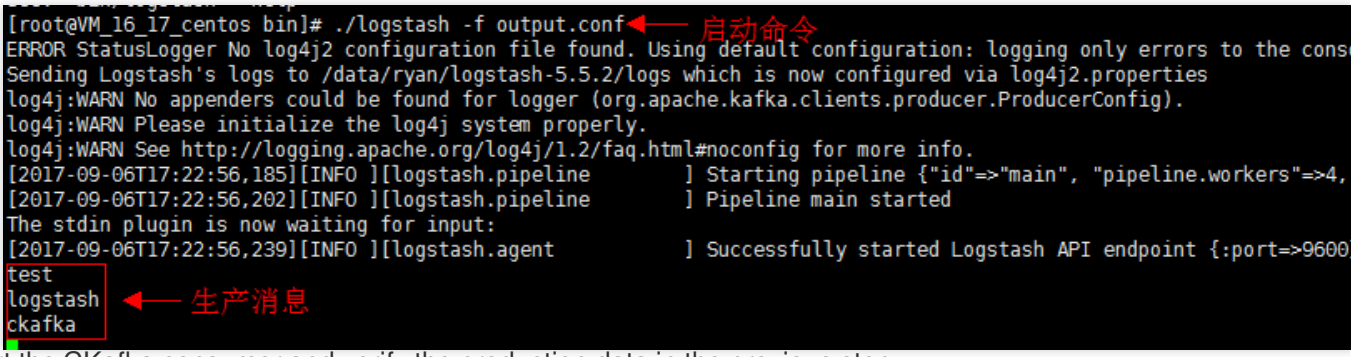
```
input {
      stdin{}
}

output {
   kafka {
        bootstrap_servers => "xx.xx.xx.xx:xxxx"  // Ckafka connection address
        topic_id => "logstash_test" // CKafka topic name
        security_protocol => "SASL_PLAINTEXT"
        sasl_mechanism => "PLAIN"
        jaas_path => "xx/xx/kafka-client-jaas.conf"
       }
}
```

The content of kafka-client-jaas.conf is as follows:

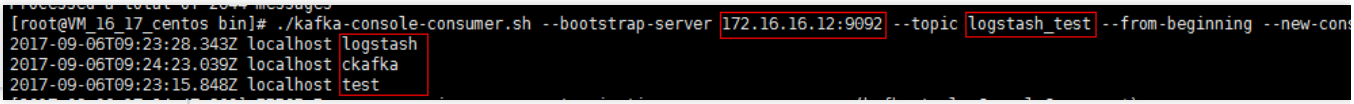
```
KafkaClient {
    org.apache.kafka.common.security.plain.PlainLoginModule required
    username="username"
    password="password";
};
```

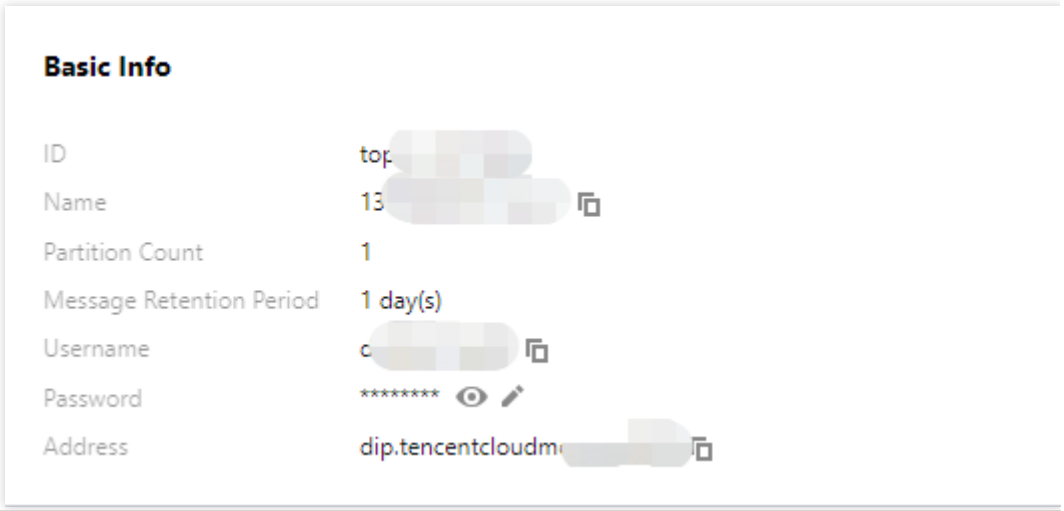
3. Run the following command to start Logstash and send messages to the created Topic.

```
./logstash -f output.conf
```



4. Start the CKafka consumer and verify the production data in the previous step.



Parameter	Description
bootstrapServers	The connection address. It can be obtained from the basic information page of an elastic Topic 
username	The username. It can be obtained from the basic information page of an elastic Topic in the co
password	The user password. It can be obtained from the basic information page of an elastic Topic in th
topic_id	The topic name. It can be obtained from the basic information page of an elastic Topic in the c
group.id	The consumption group name. It can be obtained from the subscription relationship list of an el console.