TDMQ for CKafka

Getting Started

Product Documentation

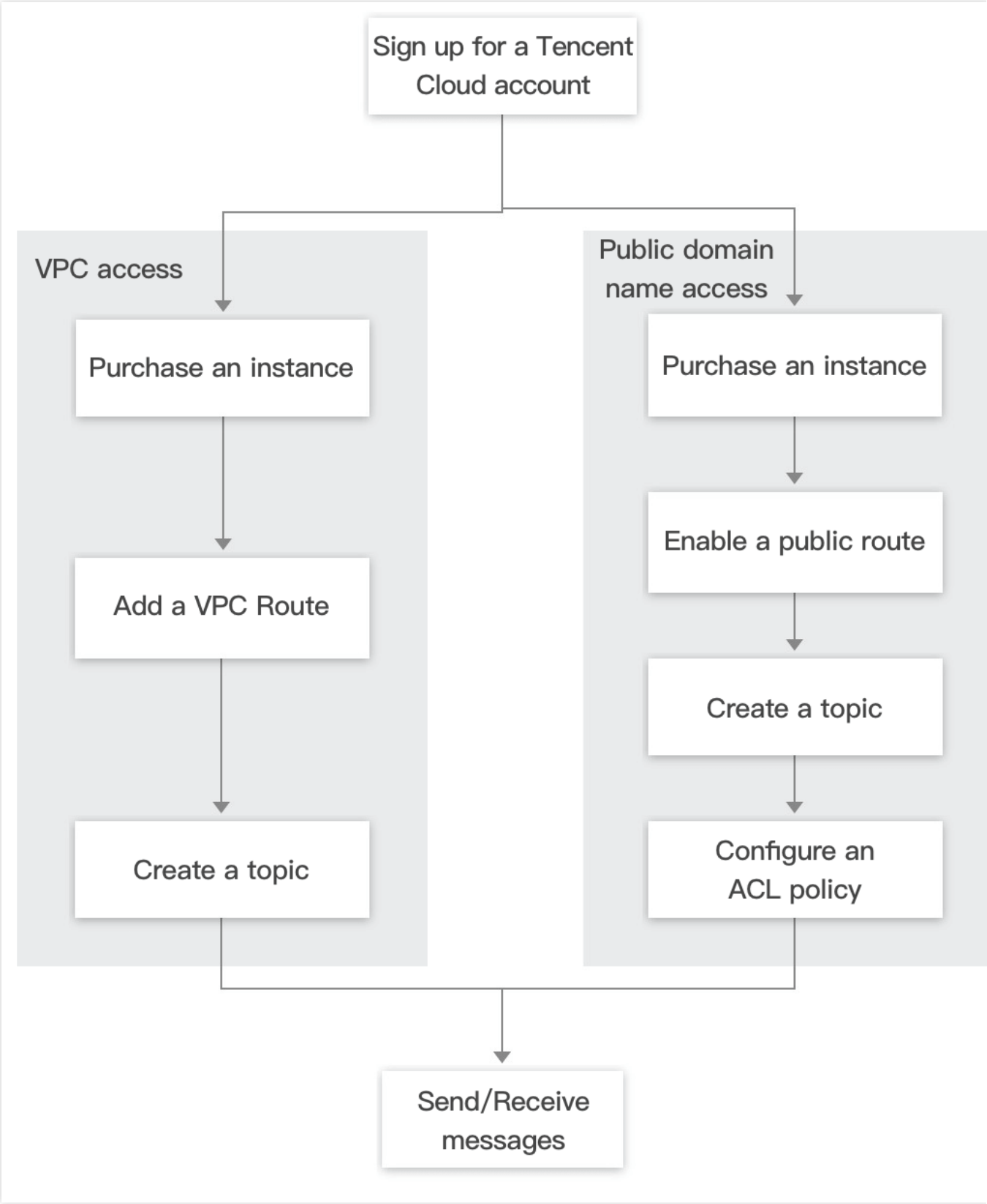
Contents

Getting Started
Process Overview

Last updated：2024-01-09 14:45:02

The process of accessing CKafka varies by network type:
For access via VPC, you can select an appropriate VPC according to your business needs.
For access via public network route, you need to enable a separate public route and configure an ACL policy for the topic.

Flowchart



Obtaining Access Permission
Getting Access Authorization

Last updated：2024-01-09 14:45:02

Basic CAM Concepts

The root account authorizes sub-accounts by associating policies. The policy setting can be specific to the level of **[API, Resource, User/User Group, Allow/Deny, and Condition]**.

Accounts

Root account: It owns all Tencent Cloud resources and can access any of its resources.
Sub-account: This includes sub-users and collaborators.
Sub-user: It is created and fully owned by a root account.
Collaborator: It has the identity of a root account. After it is added as a collaborator of the current root account, it becomes one of the sub-accounts of the current root account and can switch back to its root account identity.
Identity credential: includes login credentials and access certificates. **Login credential** refers to a user's login name and password. **Access certificate** refers to Tencent Cloud API keys (SecretId and SecretKey).

Resource and permission

Resource: An object that is operated in Tencent Cloud services, such as a CVM instance, a COS bucket, or a VPC instance.
Permission: It is an authorization that allows or forbids users to perform certain operations. By default, the **root account** has full access to all resources under the account, while a **sub-account** does not have access to any resources under its root account.
Policy: It is a syntax rule that defines and describes one or more permissions. The **root account** performs authorization by **associating policies** with users/user groups.

Using CKafka with Sub-Account

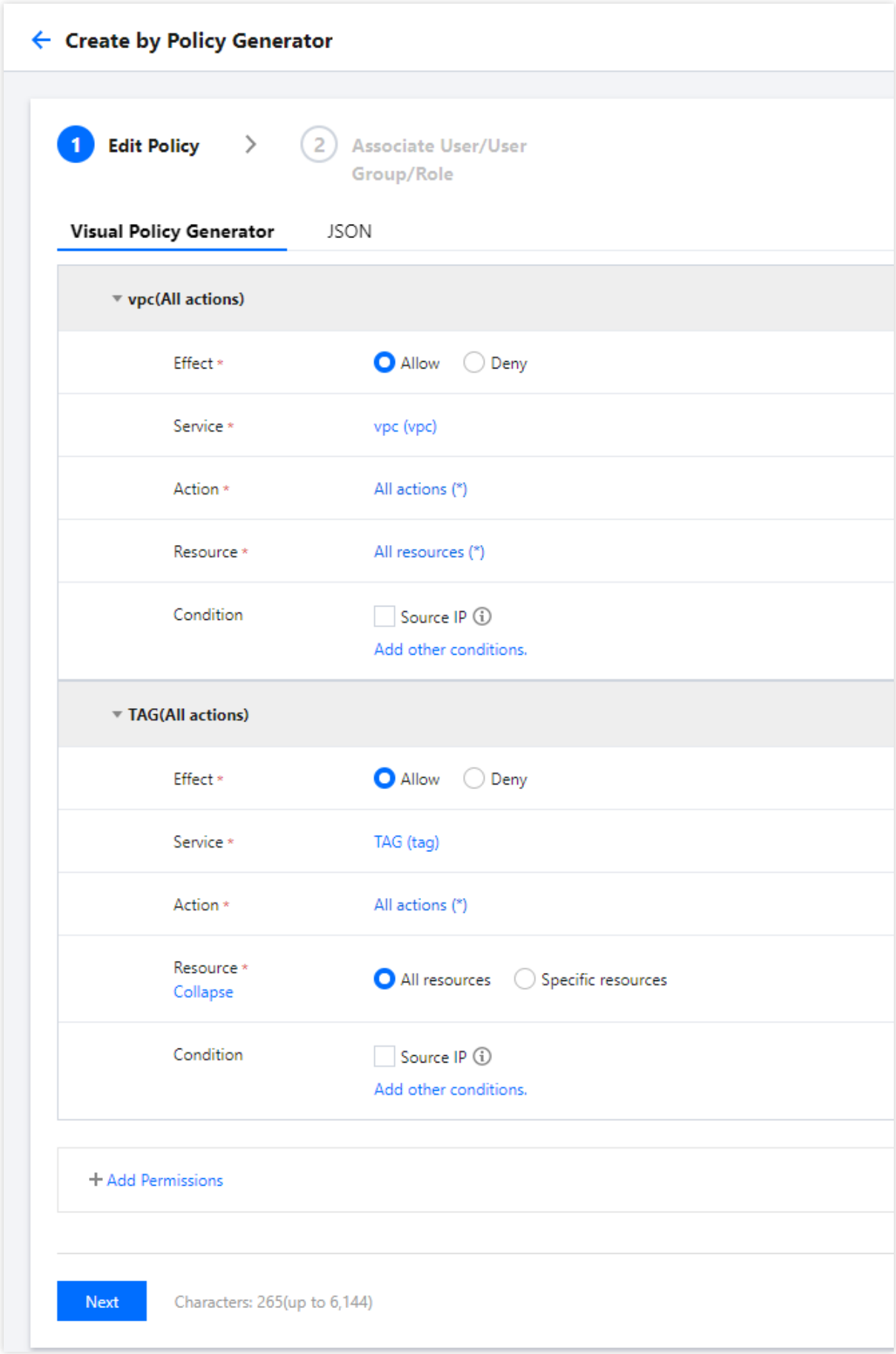
A sub-account needs to be authorized in the following two aspects before it can use CKafka:
1. CKafka needs to get permissions to access other Tencent Cloud service resources, such as viewing VPCs and tags. Therefore, a role (and its permission policy) should be passed to the CKafka service by associating the `ckafka_PassRole` policy with the sub-account. For detailed directions, see Step 1. Grant the `ckafka_PassRole` policy. For details of use cases of the policy, see Appendix.

2. For the sub-account to use CKafka, the root account needs to grant it the **full access** or **permissions of specified resources**. Based on your business needs, you can choose the scope of permissions to be granted. For detailed directions, see Step 2. Grant full access or the permissions of specified resources.

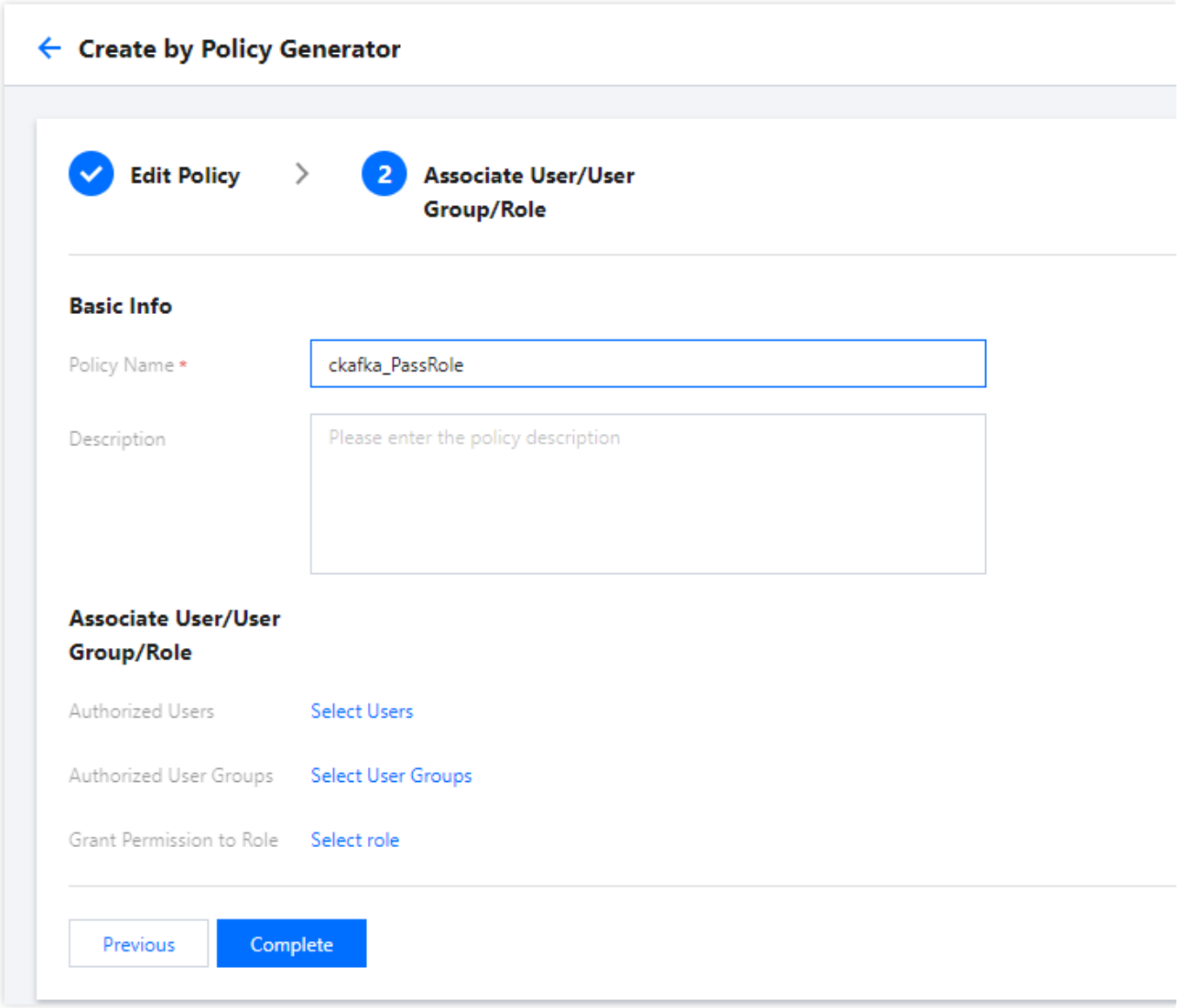
Step 1. Grant the `ckafka_PassRole` policy

Creating `ckafka_PassRole` policy

1. Log in to the CAM Console with the root account.
2. In the left sidebar, click **Policies** to go to the policy management page.
3. Click **Create Custom Policy**.
4. In the **Select Policy Creation Method** pop-up window, select **Create by Policy Generator** to enter the **Create by Policy Generator** page.
5. Enter the services, operations, resources, and other information in the policy as needed. You can refer to the figure below to generate the `ckafka_PassRole` policy. Then, click **Next**.



6. Enter the policy name `ckafka_PassRole`, associate it with the target **user**, **user group** or **role**, and click **Complete**.

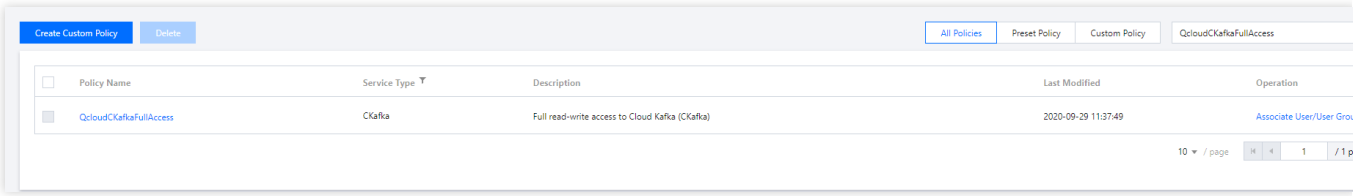


Step 2. Grant full access or the permissions of specified resources

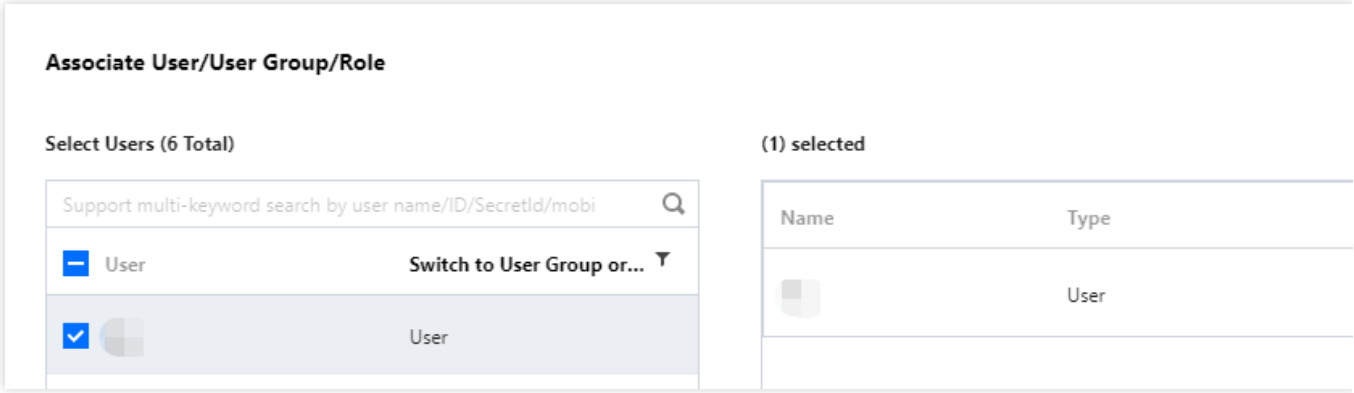
Full access

Permissions of specified resources

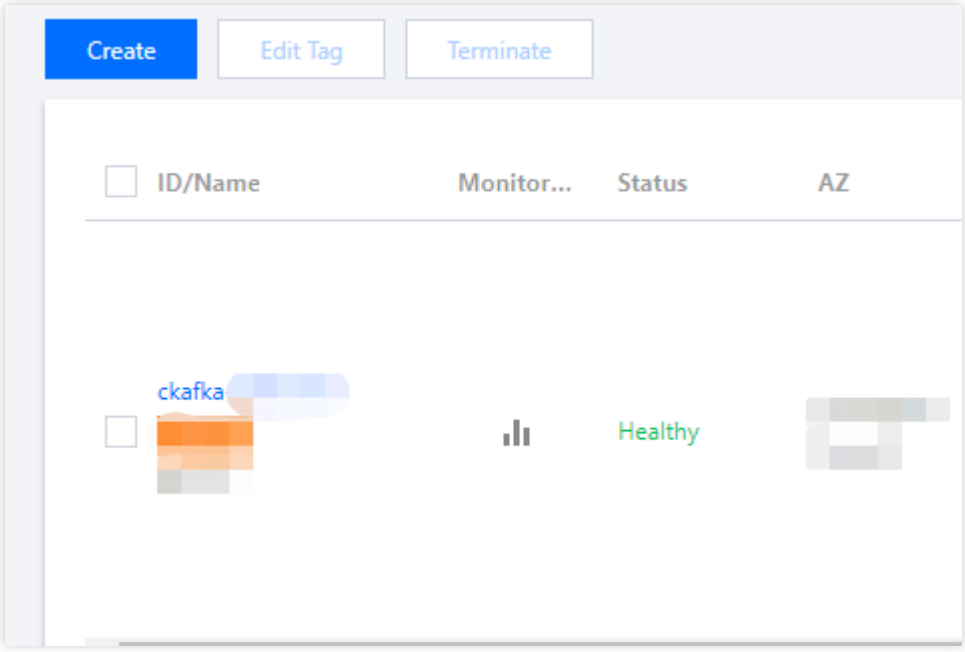
1. Log in to the CAM Console with the root account.
2. In the left sidebar, click **Policies** to go to the policy management page.
3. Search for `QcloudCKafkaFullAccess` on the right.



4. In the search results, click the **Associated Users/Groups** of `QcloudCKafkaFullAccess` and select the sub-account to be authorized.

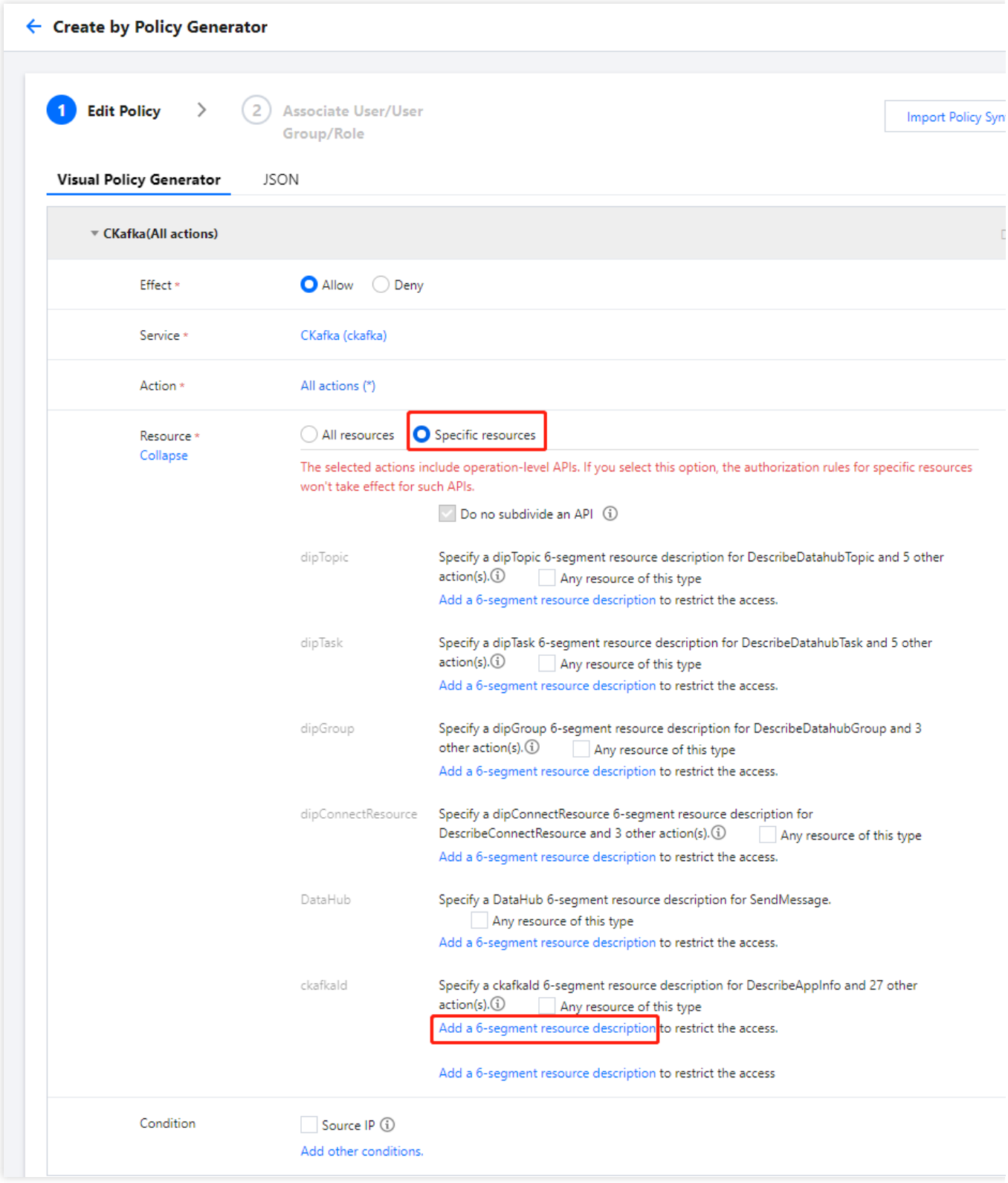


1. Log in to the CKafka console and find the CKafka instance resource that needs to be authorized.
2. Get the ID of the instance as shown below:

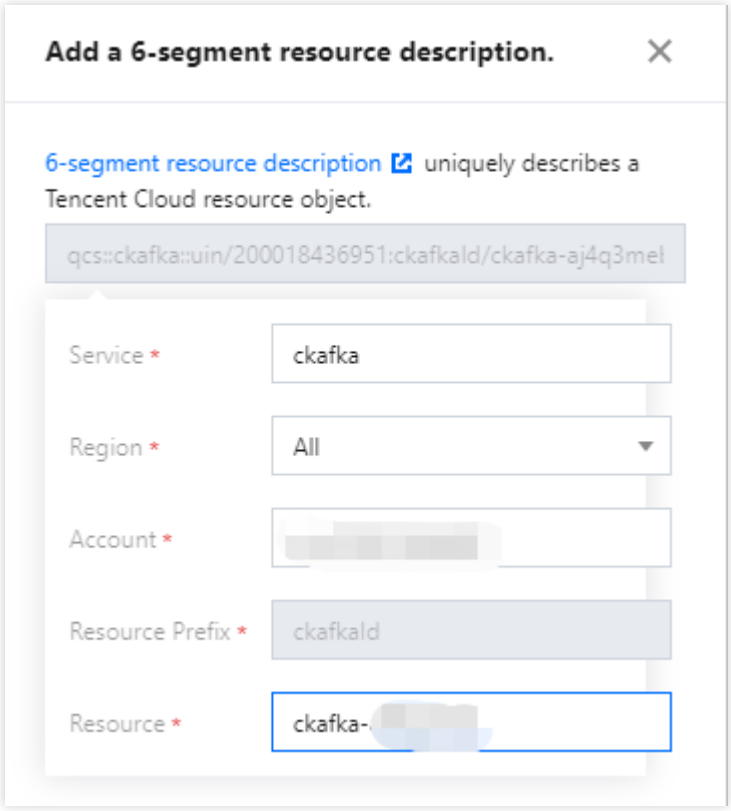


3. Log in to the CAM console and click **Policies** on the left sidebar to enter the policy management list page.

4. Click **Create Custom Policy**. In the **Select Policy Creation Method** pop-up window, select **Create by Policy Generator** to enter the **Create by Policy Generator** page.
5. Enter the services, operations, resources, and other information in the policy as needed. Then, click **Add a six-segment resource description** as shown below:



6. Enter the ID of the specified instance in the six-segment resource description:



7. Click **Next**, specify users or user groups for the policy, and click **Complete**.

Appendix

Using CKafka involves calling the following Tencent Cloud services. The root account needs to authorize sub-accounts separately for them to use CKafka features. The details are as follows:

Tencent Cloud services	Operations Affecting Ckafka
VPC	Select the VPC to which the instance access address belongs when creating an instance
Tag	Select relevant tag when creating an instance

Granting Operation-Level Permissions to Sub-Accounts

Last updated：2024-01-09 14:45:02

Overview

This document describes how to use the Tencent Cloud root account to authorize sub-accounts at the operation level. You can grant different read and write permissions to sub-accounts as needed.

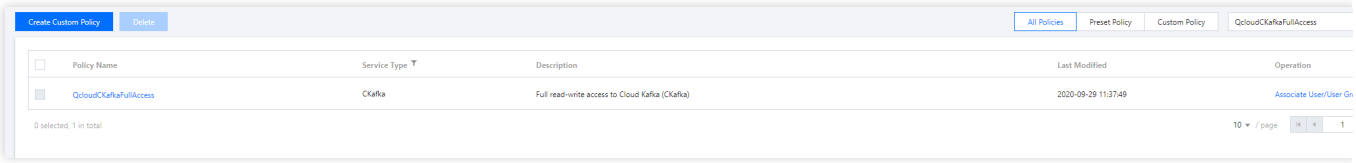
Directions

Full access permission

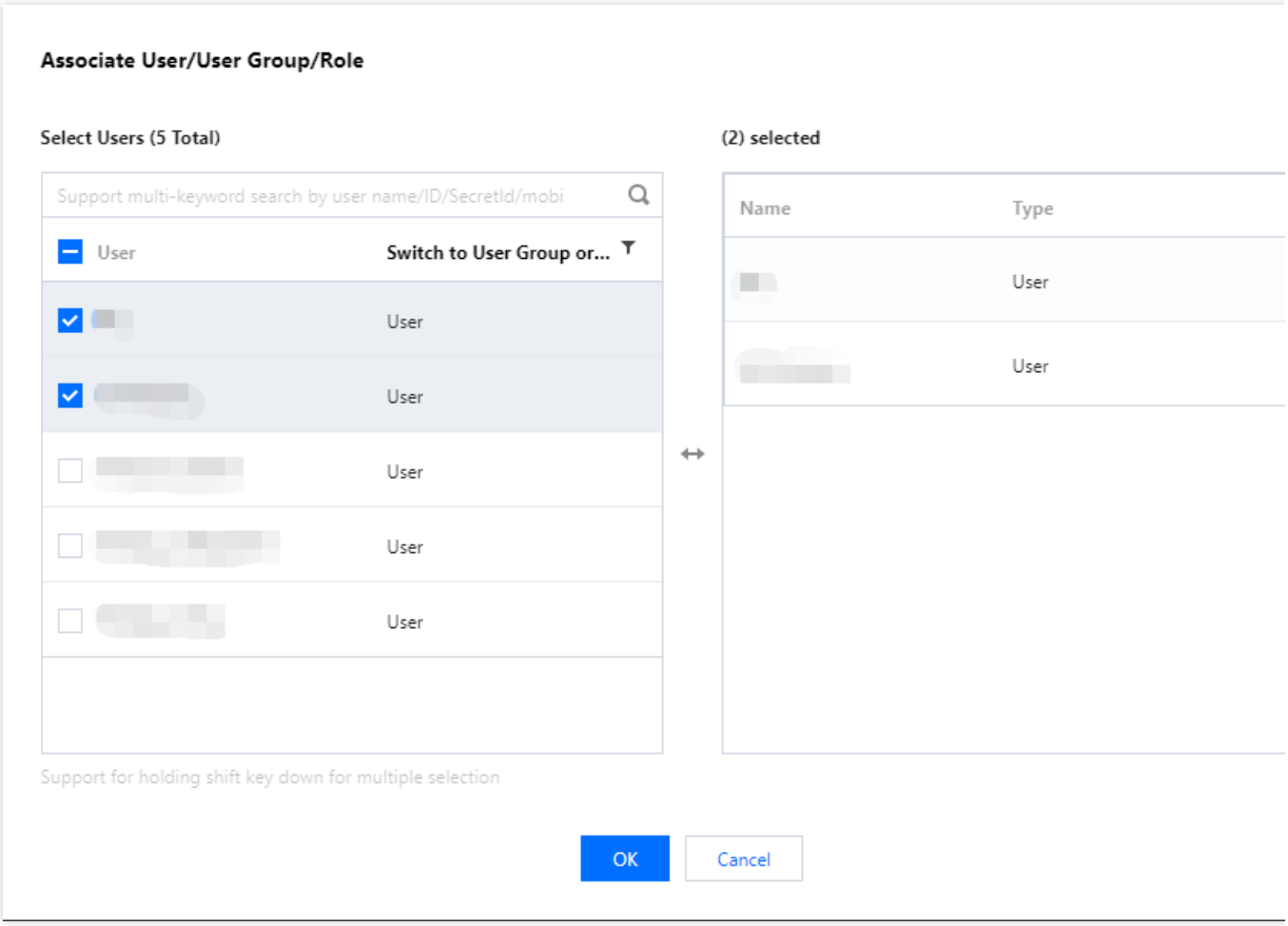
Note:

After granting full access permissions to a sub-account, the sub-account will have **full read and write capabilities** to **all resources** under the root account.

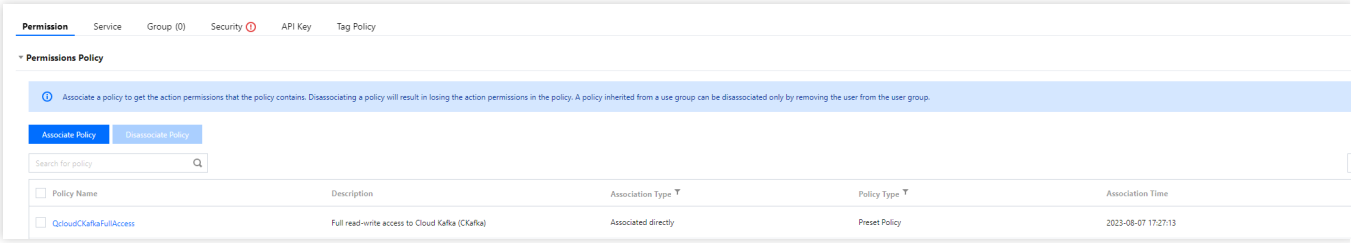
1. Log in to the CAM Console with the root account.
2. In the left sidebar, click **Policies** to go to the policy management page.
3. Search for **QcloudCKafkaFullAccess** on the right.



4. In the search results, click the **Associated Users/Groups** of **QcloudCKafkaFullAccess** and select the sub-account to be authorized.



5. Click **OK** to complete the authorization, which will be displayed in the **Policy List** of the user.

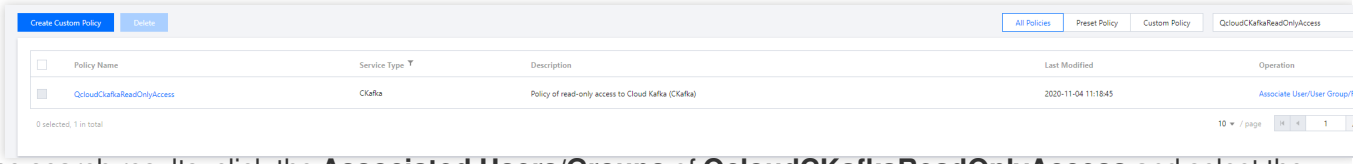


Read-only permission

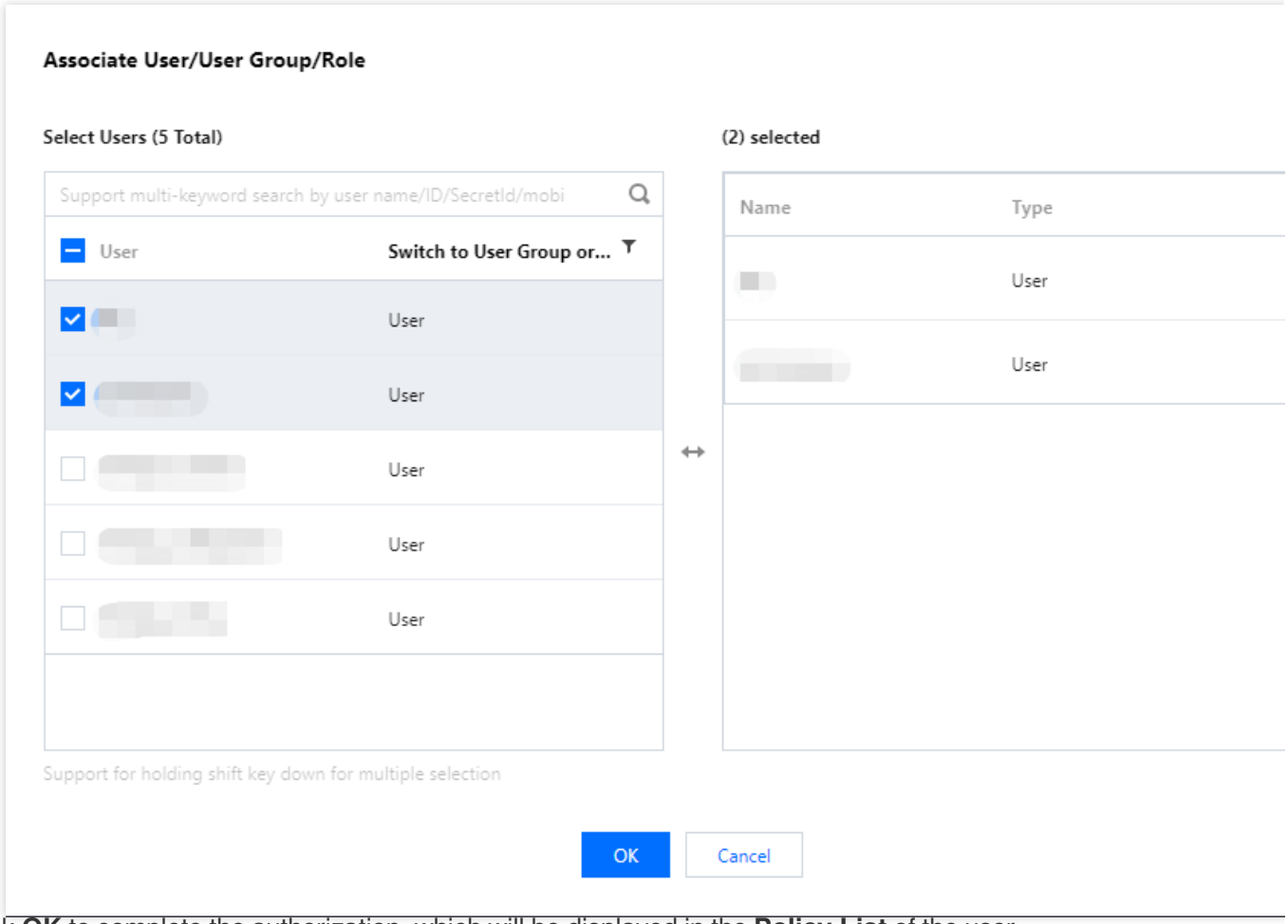
Note:

After granting the read-only permission to a sub-account, the sub-account will have **read-only capability** to **all resources** under the root account.

1. Log in to the CAM Console with the root account.
2. In the left sidebar, click **Policies** to go to the policy management page.
3. Search for **QcloudCKafkaReadOnlyAccess** on the right.

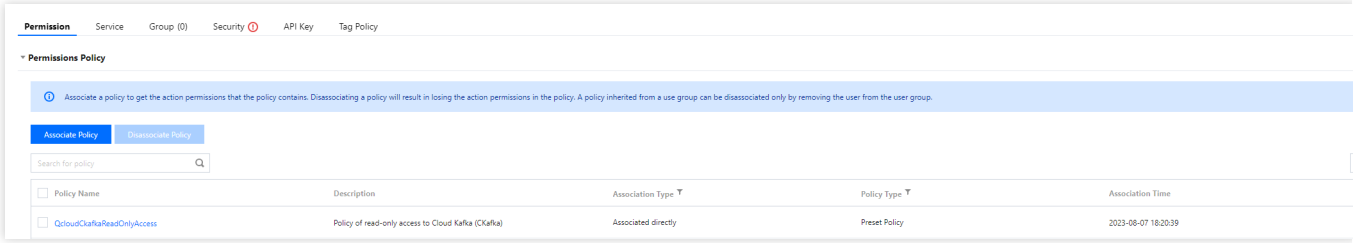


4. In the search results, click the **Associated Users/Groups** of **QcloudCKafkaReadOnlyAccess** and select the sub-account to be authorized.



5. Click **OK** to complete the authorization, which will be displayed in the **Policy List** of the user.

6.



Other methods

Resource-Level Authorization

Tag-Level Authorization

Granting Resource-Level Permissions to Sub-Accounts

Last updated：2024-01-09 14:45:02

Overview

This document describes how to use the root account to authorize sub-accounts at the resource level. After successful authorization, the sub-accounts will have the capability to control a certain resource.

Prerequisites

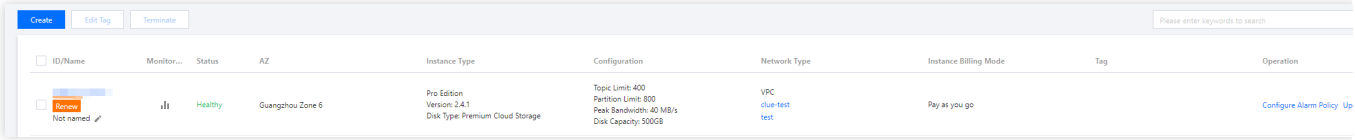
You must have a Tencent Cloud root account and have activated the Cloud Access Management (CAM) service.
Your root account must have at least one sub-account, and you have completed the authorization as instructed in Getting Access Authorization.
You must have at least one CKafka instance.

Directions

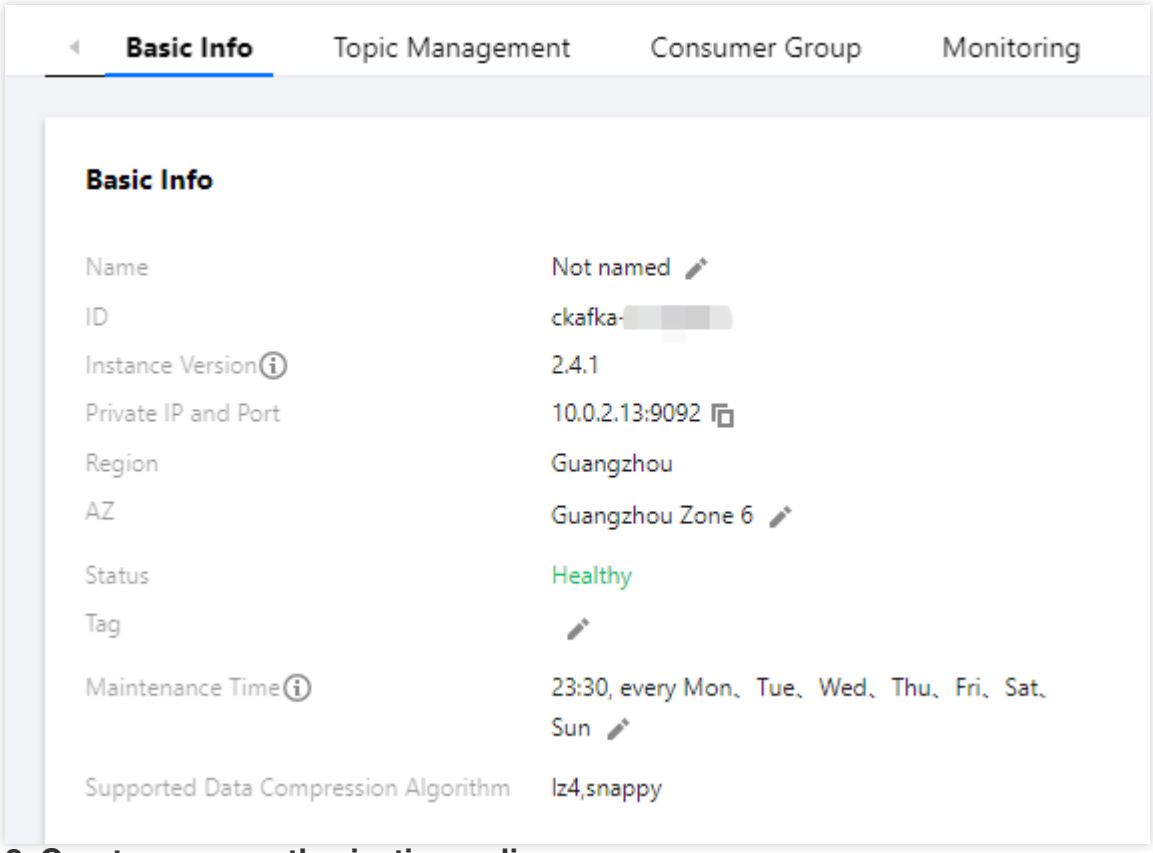
By using the policy feature in the CAM console, you can grant a sub-account access to the CKafka resources owned by the root account. Taking cluster resource as an example, the following describes the detailed steps for **granting the sub-account access to CKafka resources**, which also apply to other types of resources.

Step 1. Obtain the CKafka cluster ID

1. Log in to the CKafka console with **root account**, select an existing cluster instance, and click it to enter the details page.

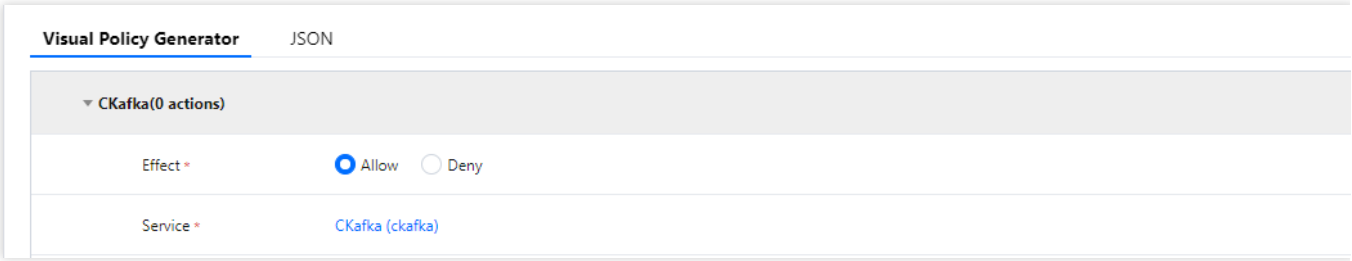


2. In **Basic Info**, the field **ID** indicates the ID of the current CKafka cluster.

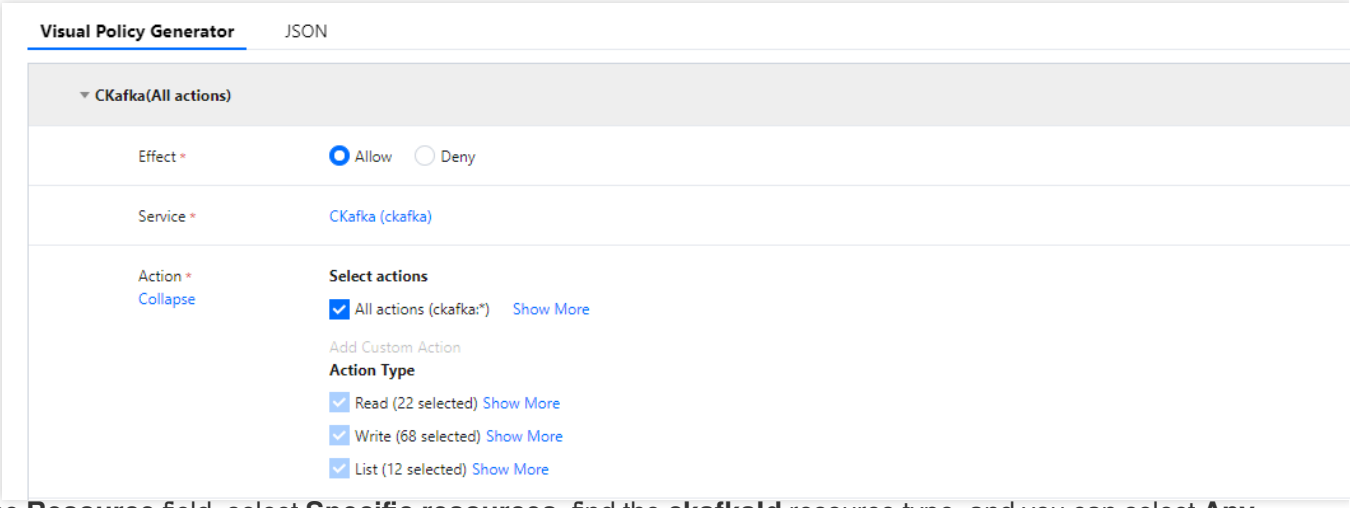


Step 2. Create a new authorization policy

1. Log in to the CAM console and click **Policies** on the left sidebar.
2. Click **Create Custom Policy** > **Create by Policy Generator**.
3. In the visual policy generator, select **Allow** for **Effect**, enter CKafka in **Service** to filter, and select **CKafka (ckafka)**.

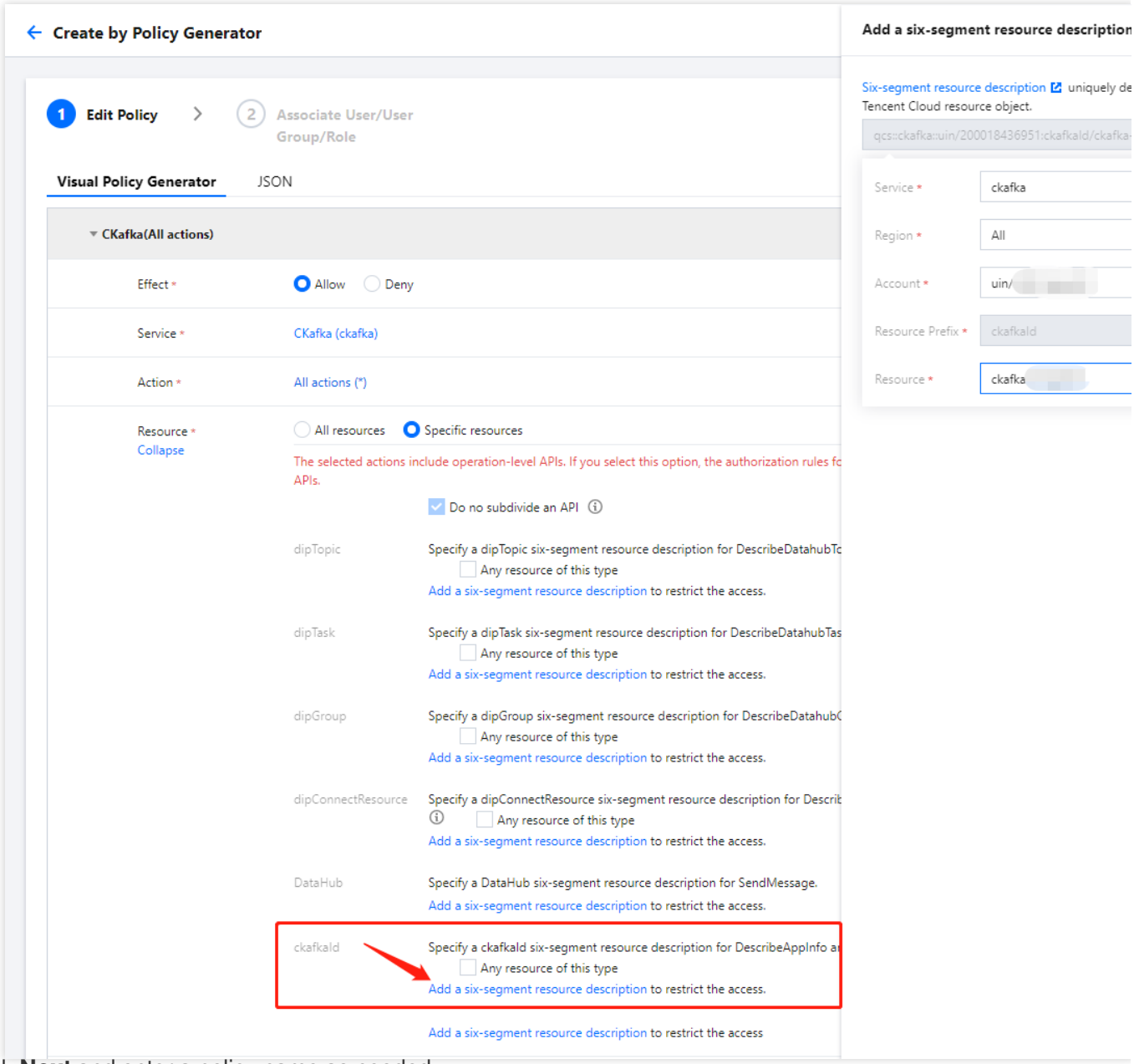


4. Select **All actions** in **Action**, and you can also select the action type as needed.



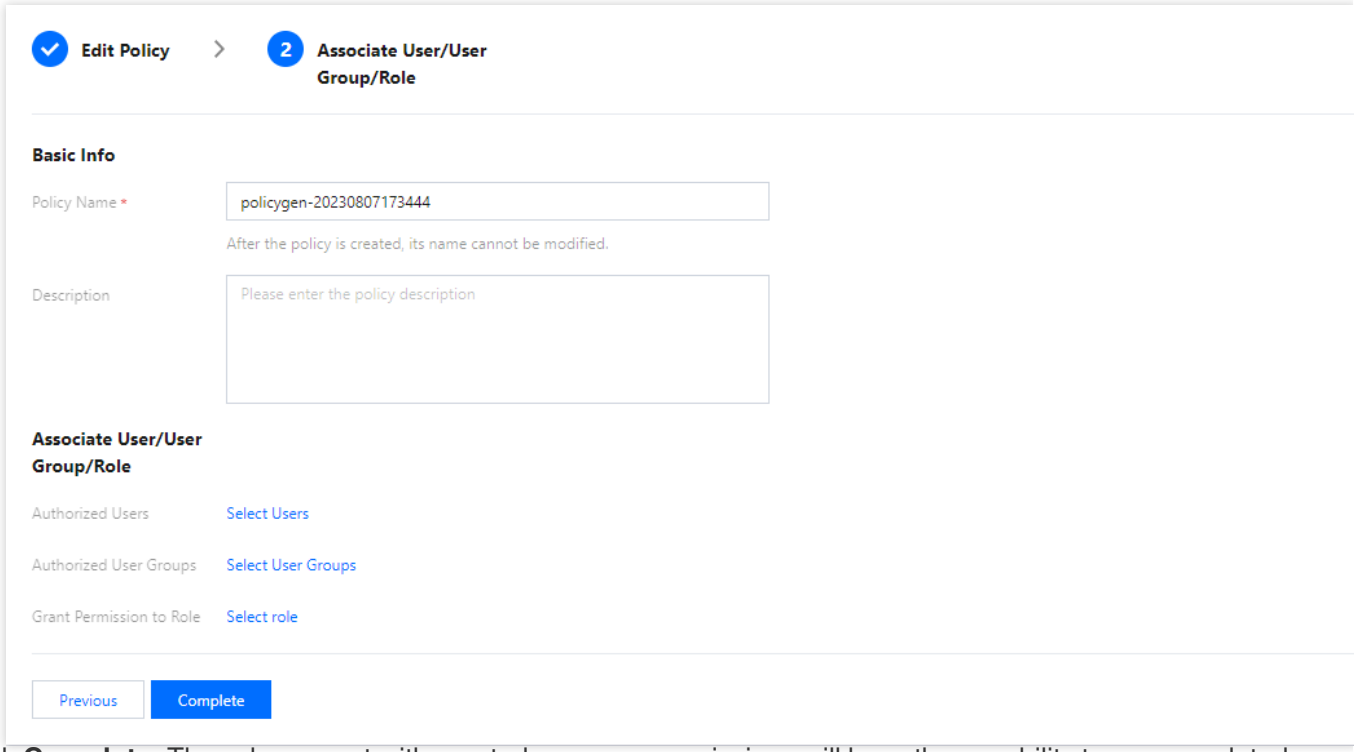
5. In the **Resource** field, select **Specific resources**, find the **ckafkaId** resource type, and you can select **Any resource of this type** on the right to authorize all cluster resources, or click **Add a six-segment resource description** to authorize specific cluster resources.

6. If you click **Add a six-segment resource description**, enter the **cluster ID** for **Resource** in the pop-up dialog box. For how to obtain the cluster ID, see Step 1.



7. Click **Next** and enter a policy name as needed.

8. Click **Select Users** or **Select User Groups** to select the users or user groups that need to be granted resource permissions.



9. Click **Complete**. The sub-account with granted resource permissions will have the capability to access related resources.

Other authorization methods

Operation-Level Authorization

Tag-Level Authorization

Granting Tag-Level Permissions to Sub-Accounts

Last updated：2024-01-09 14:45:02

Overview

This document describes how to use the root account to authorize sub-accounts at the tag level. After successful authorization, the sub-accounts will have the capability to control a certain resource under the authorized tag.

Prerequisites

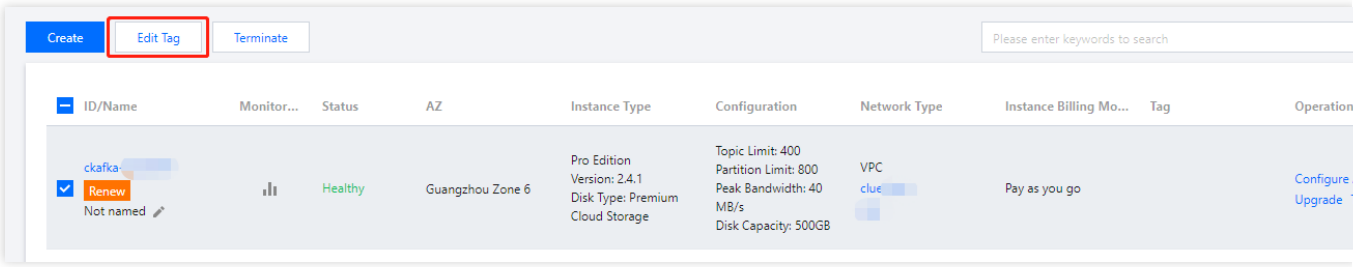
You must have a Tencent Cloud root account and have activated the Cloud Access Management (CAM) service.
Your root account must have at least one sub-account, and you have completed the authorization as instructed in Getting Access Authorization.
You must have at least one CKafka cluster instance.
You must have at least one **tag**, if you don't have one, you can go to the Tag console > **Tag List** to create a new one.

Directions

By using the policy feature in the CAM console, you can grant a sub-account full access to the tagged CKafka resources owned by the root account through the tag authorization. The following describes the detailed steps for **granting the sub-account access to CKafka resources by tag**

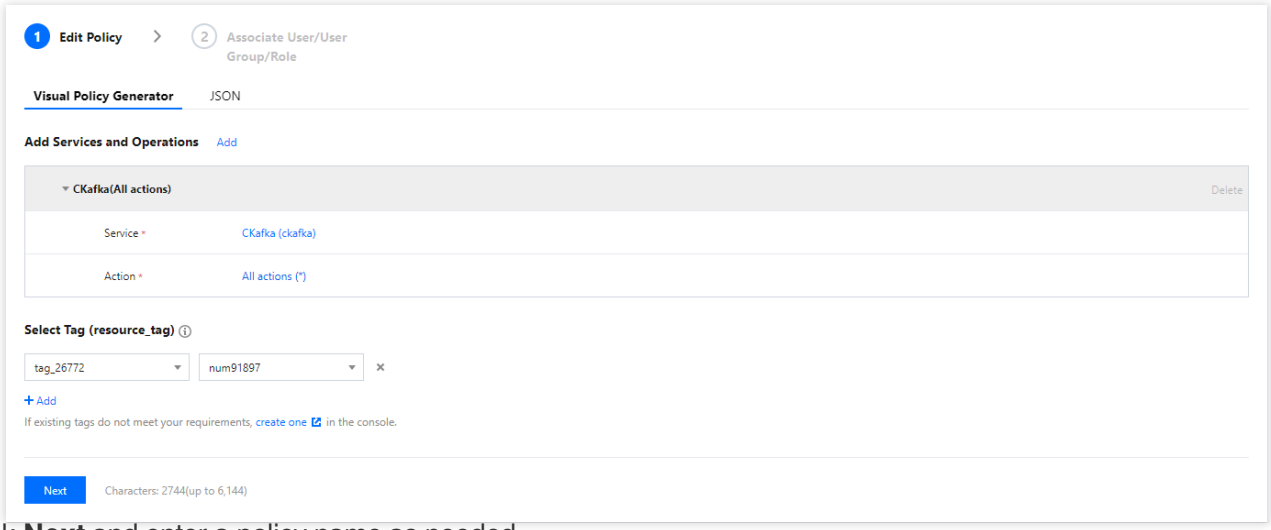
Step 1. Bind tags to resources

1. Log in to the CKafka console with **root account**, and enter the instance list page.
2. Select the target instance, click **Edit Tag** in the upper left corner, and bind the resource tag to the instance.

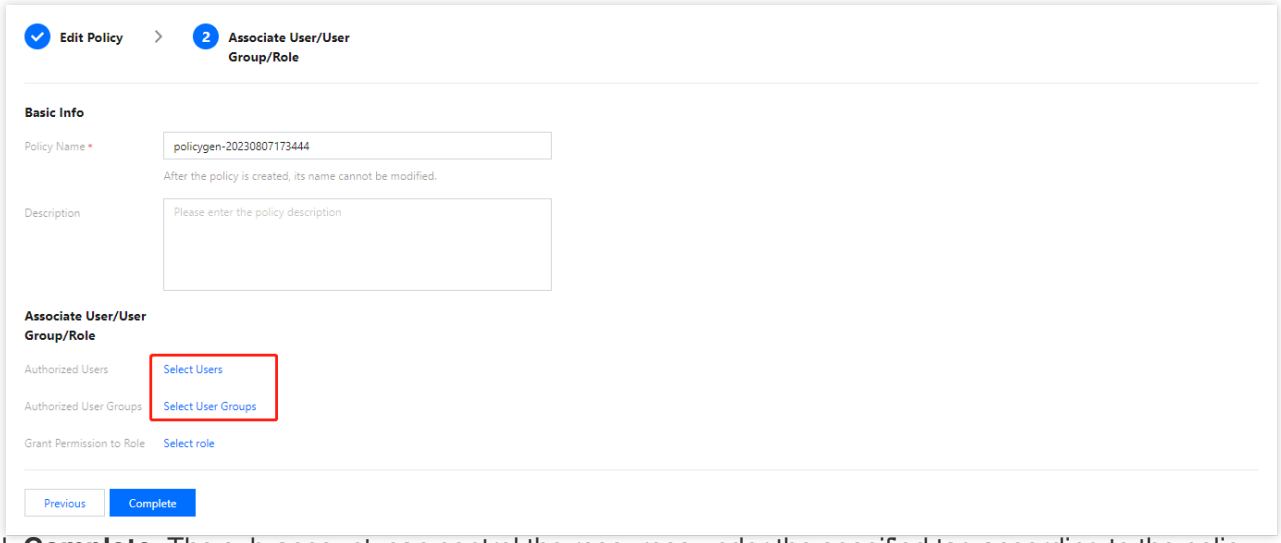


Step 2. Authorize by Tag

1. Log in to the CAM console and click **Policies** on the left sidebar.
2. Click **Create Custom Policy** > **Authorize by Tag**.
3. In the visual policy generator, enter CKafka in **Service** to filter, and select **CKafka (ckafka). Then, select** All actions in **Action**, and you can also select the action type as needed.



4. Click **Next** and enter a policy name as needed.
5. Click **Select Users** or **Select User Groups** to select the users or user groups that need to be granted resource permissions.

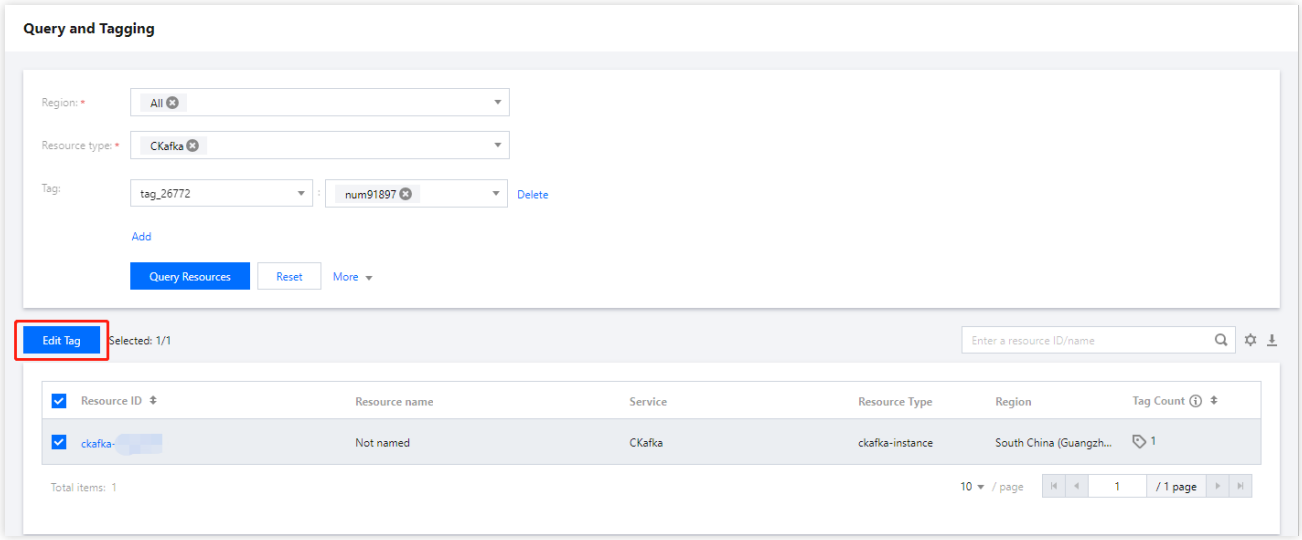


6. Click **Complete**. The sub-account can control the resources under the specified tag according to the policy.

Managing Resource Tags

You can also manage resource tags in a unified manner in the **Tag console**. The detailed operations are as follows.

1. Log in to the **Tag console**.
2. Select **Resource Tag** in the left navigation bar, select query conditions as needed, and select **CKafka** > **ckafka-instance** in **Resource type**.
3. Click **Query Resources**.
4. Select the required resources in the result and click **Edit Tag** to bind or unbind tags in batches.



Other authorization methods

Operation-Level Authorization

Resource-Level Authorization

VPC Access
Step 1. Create an Instance

Last updated：2024-01-09 14:45:02

Overview

This document describes how to create an instance and deploy a VPC in the CKafka console.

Prerequisites

You have signed up for a Tencent Cloud account.
You have created a VPC.

Directions

1. Log in to the CKafka console.
2. Select **Instance List** on the left sidebar, click **Create** to go to the instance purchase page, and enter the purchase information as needed.
Billing Mode: Pro Edition instances support **monthly subscription** and **pay-as-you-go** billing modes, while Standard Edition instances only support **monthly subscription**.
Specs Type: Select the Standard or Pro Edition based on your business needs.
Kafka Version: Select a Kafka version based on your business needs. For more information, see Suggestions for CKafka Version Selection.
Region: Select a region close to the resource for client deployment.
AZ:
Standard Edition: This edition does not support multi-AZ deployment.
Pro Edition: If multi-AZ deployment is supported in the current region, you can select up to four AZs for deployment. For more information, see Multi-AZ Deployment.
Product Specs: Select a model based on the peak bandwidth and disk capacity.
Message Retention Period: Select a value between 1 and 2,160 hours.
When the disk capacity is insufficient (that is, the disk utilization reaches 90%), previous messages will be deleted in advance to ensure service availability.
Instance Name: When purchasing multiple instances, you can create instances in batches by its numeric suffix (which is numbered in ascending order) or its designated pattern string. For more information, see Naming with Consecutive

Numeric Suffixes or Designated Pattern String.

3. VPC: Select a suitable VPC based on your business needs.

If you want to use other VPCs, follow the steps in Adding Routing Policy to modify the routing rules.

4. Click **Buy Now**. The created instance will be displayed in the instance list in about 3–5 minutes.

Step 2. Create a Topic

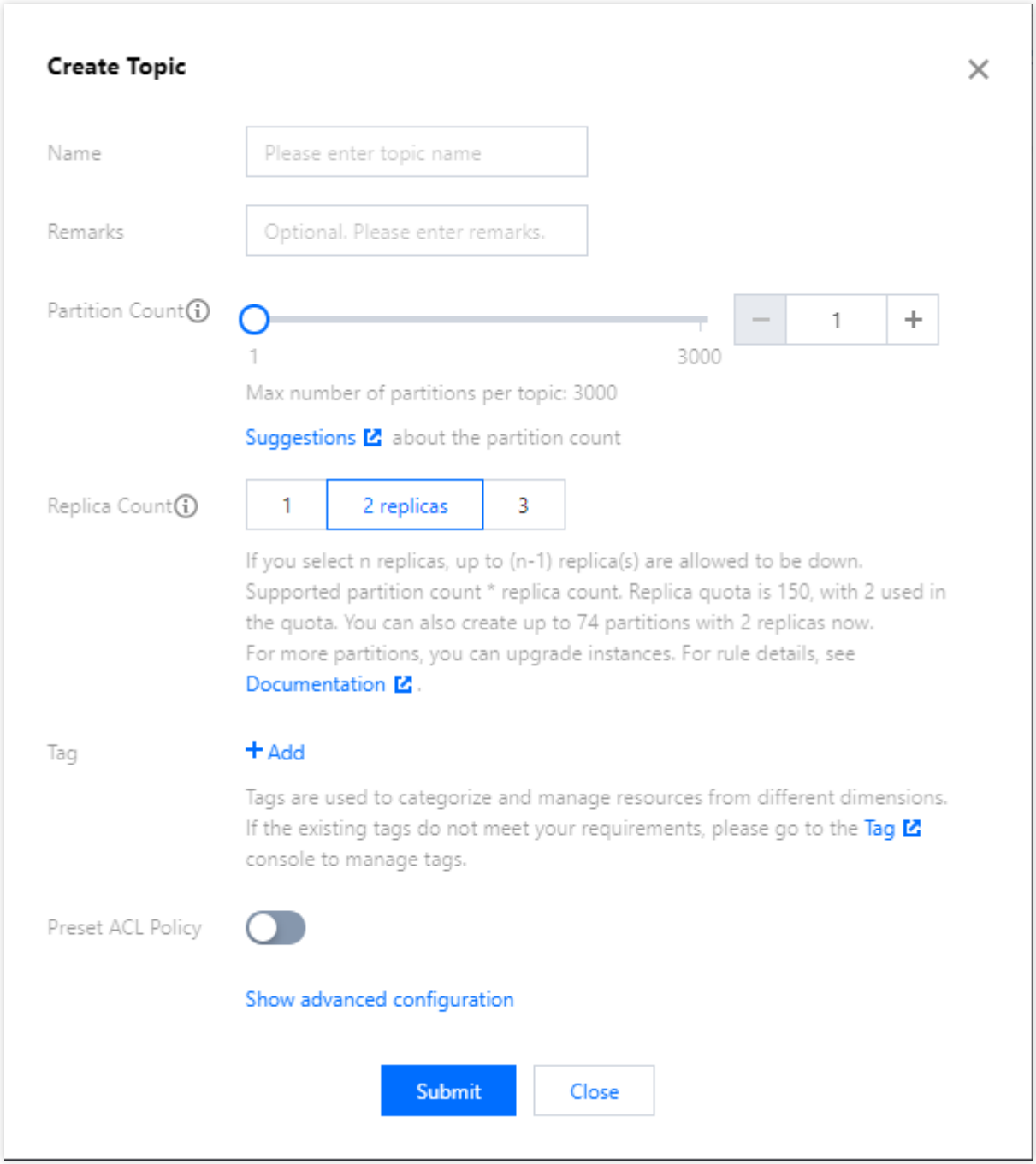
Last updated：2024-01-09 14:45:02

Overview

This document describes how to create a topic under an existing instance in the CKafka console.

Directions

1. Log in to the CKafka console.
2. On the **Instance List** page, click the **ID/Name** of the instance created in Step 1. Create an Instance to enter the instance details page.
3. On the instance details page, click **Topic Management** at the top of the page, and click **Create**.
4. In the **Create Topic** dialog box, set parameters as needed.



Name: The topic name. It cannot be changed once entered and can contain only letters, digits, underscores, or symbols ("-" and ".").

Partition Count: It is a concept in physical partition, where one topic can contain one or more partitions. CKafka uses partition as an allocation unit.

Replica Count: The number of partition replicas is used to ensure the high availability of the partition. To ensure data reliability, creating a single-replica topic is not supported. Two replicas are enabled by default.

Replicas are also counted into the number of partitions. For example, if you create 1 topic with 6 partitions, and 2 replicas for each partition, then you have a total of 12 partitions (1 x 6 x 2).

Tag: Set a resource tag. For more information, see Tag Overview.

Preset ACL Policy: Select the preset ACL policy. For more information on ACL policy, see Configuring ACL Policy.

5. Click **Submit**.

Step 3. Add a VPC Route

Last updated：2024-01-09 14:45:02

Overview

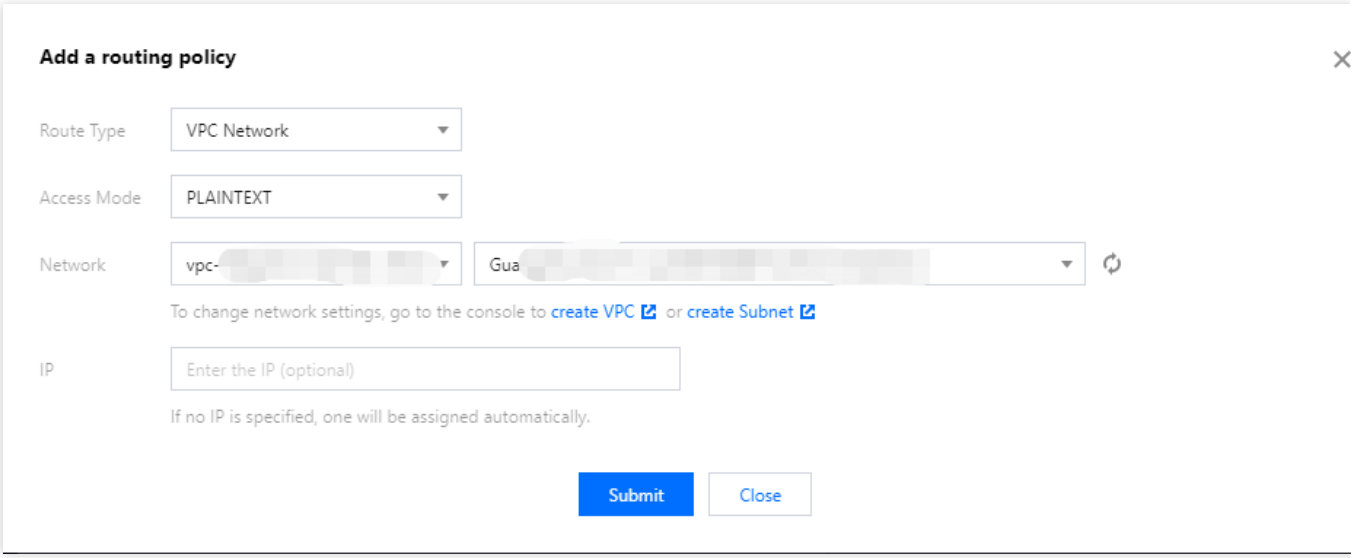
This document describes how to add a VPC route for a created instance in the CKafka console.

Prerequisites

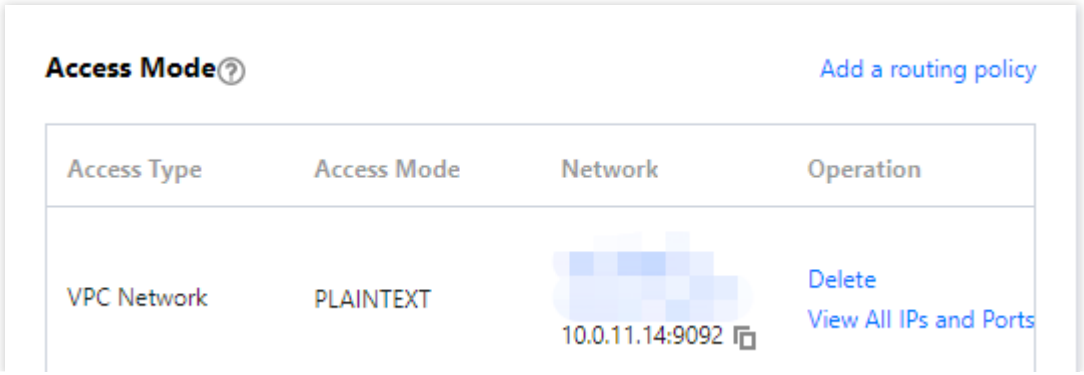
You have created an instance. For more information, see Step 1. Create an Instance

Directions

1. On the Instance List page, click the ID/name of the instance created in Step 1. Create an Instance.
2. On the instance details page, click **Add a routing policy** in the **Access Mode** section to add a VPC route.



3. Then, you can get the domain name and port for VPC access.



Step 4. Send/Receive Messages Using SDK to Receive/Send Message (Recommended)

Last updated：2024-01-09 14:45:02

Overview

This document describes how to access CKafka to receive/send messages with the SDK for Java in a VPC.
For clients in other languages, see SDK Documentation.

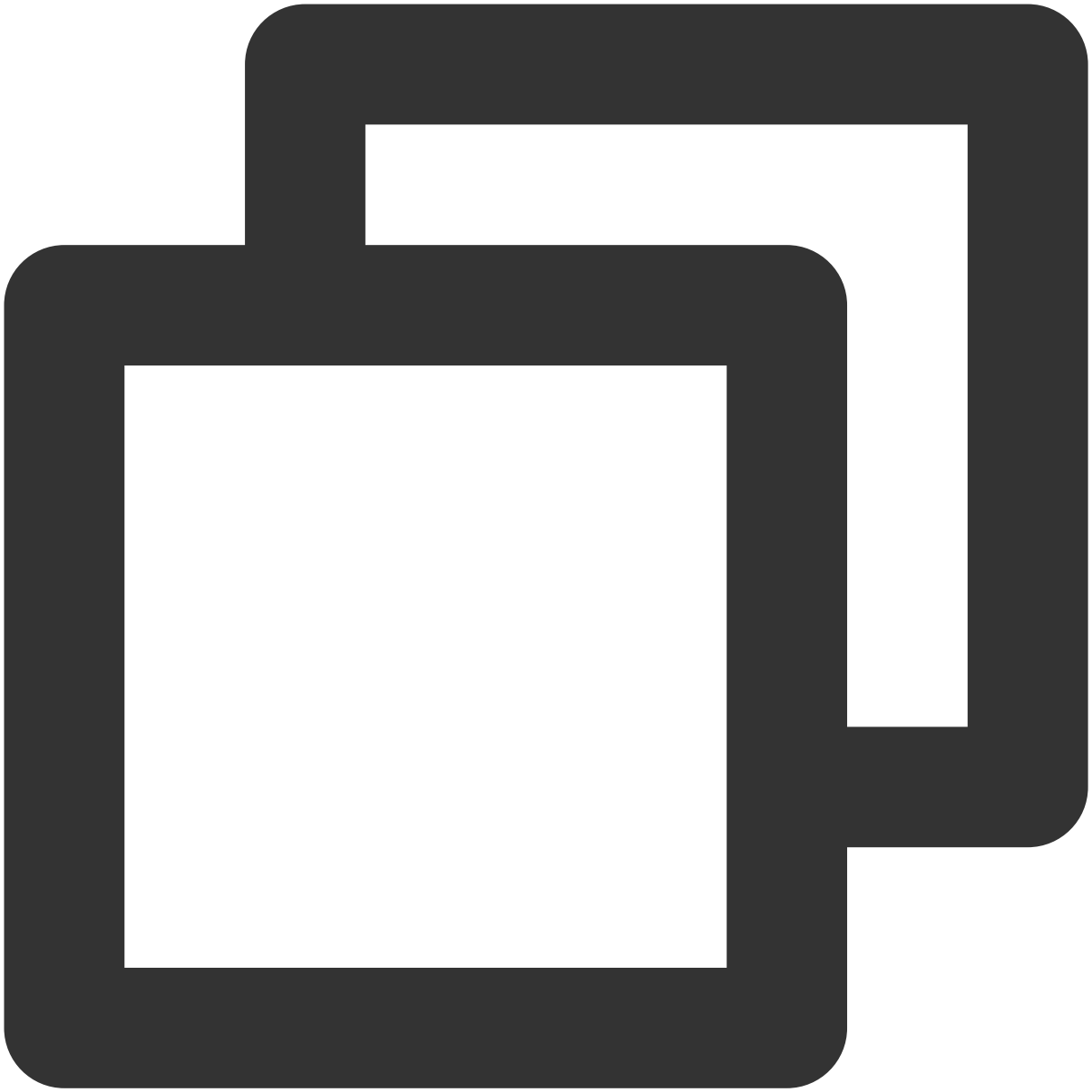
Prerequisites

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

Directions

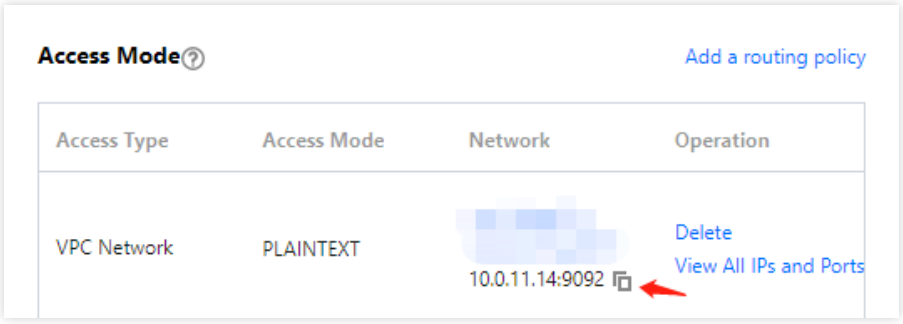
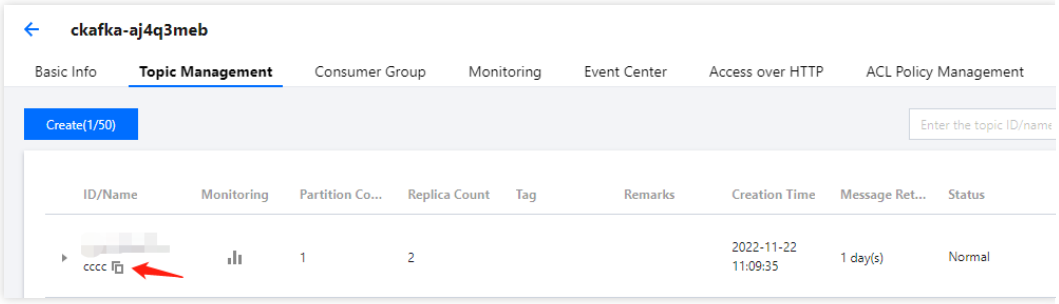
Step 1. Prepare configurations

1. Upload the downloaded demo to the Linux server under the same VPC, log in to the server, and enter the VPC directory under `javakafkademo` .
2. Modify `kafka.properties` in the `resources` directory under the VPC project.



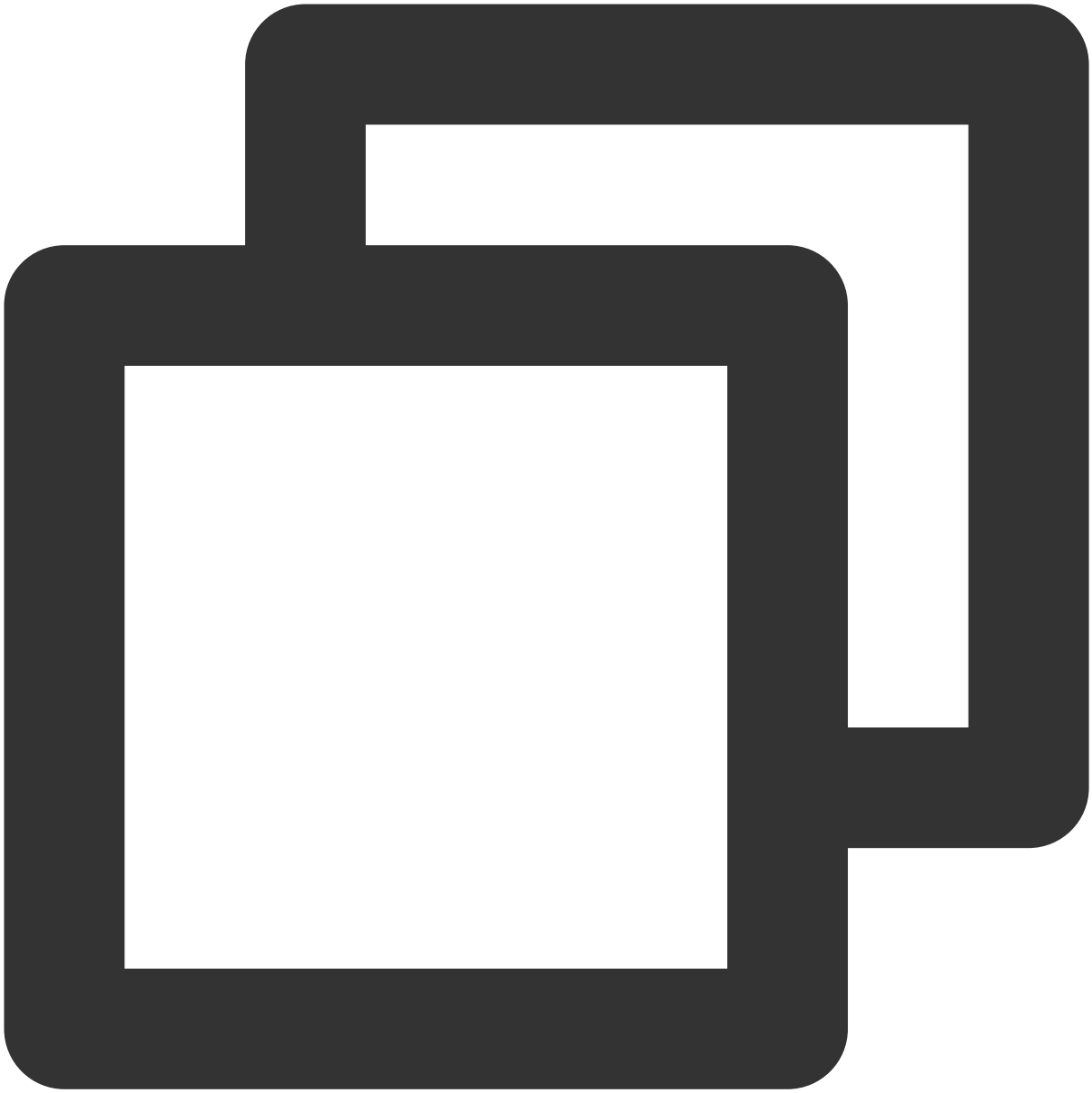
```
## Configure the accessed network by copying the information in the **Network** col
bootstrap.servers=xx.xx.xx.xx:xxxx
## Configure the topic by copying the information on the **Topic Management** page
topic=XXX
## Configure the consumer group as needed
group.id=XXX
```

Parameter	Description
bootstrap.servers	Access network, which can be copied in the **Network** column in the **Access Mode** section or

	details page in the console 
topic	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 on the **Con** page.

Step 2. Send messages

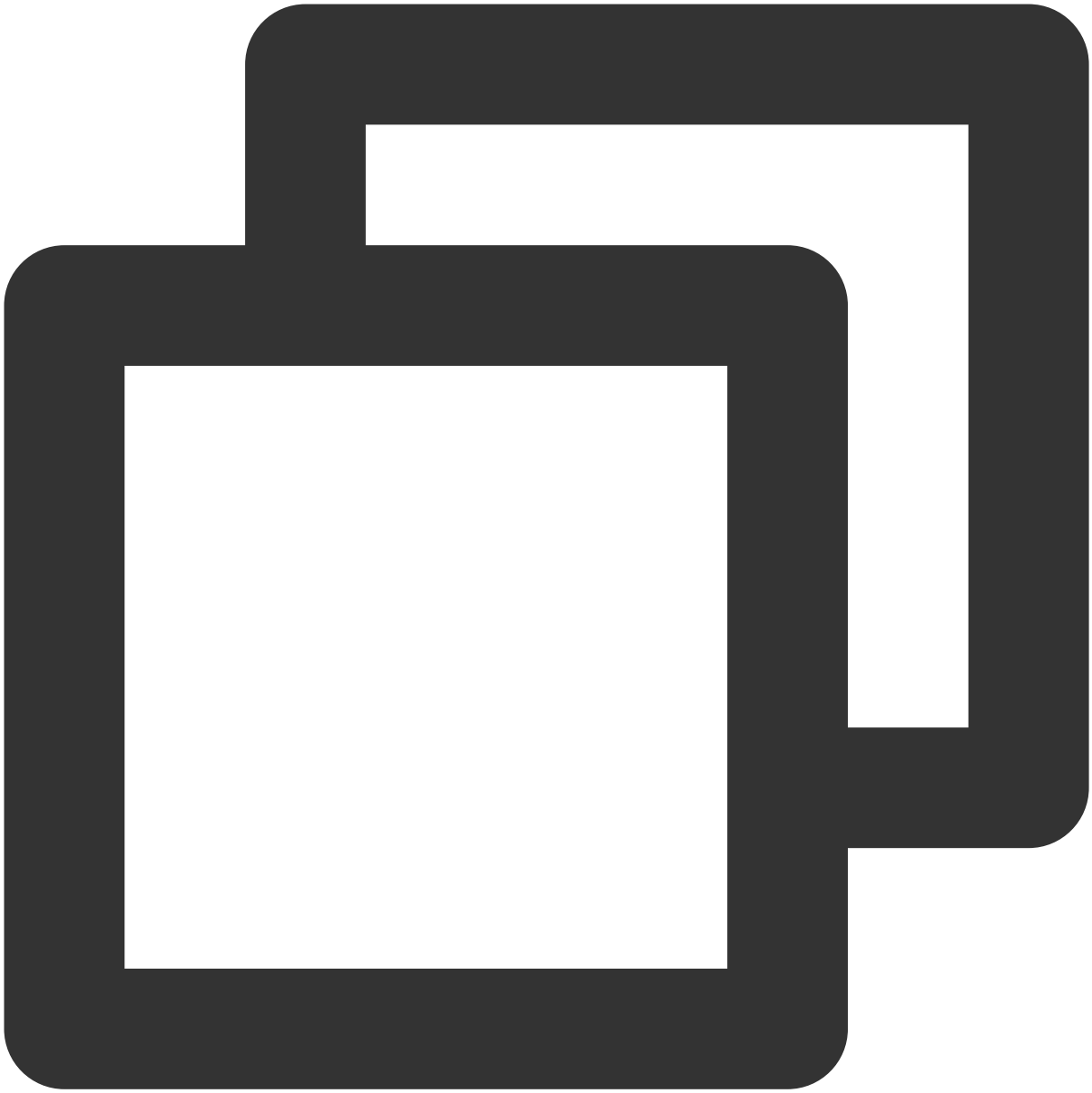
1. Compile and run the message production program `CKafkaProducerDemo.java` .



```
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 corresponding topic
        properties.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, kafkaProperties.get

        //Set the method for serializing Kafka messages. `StringSerializer` is used
```

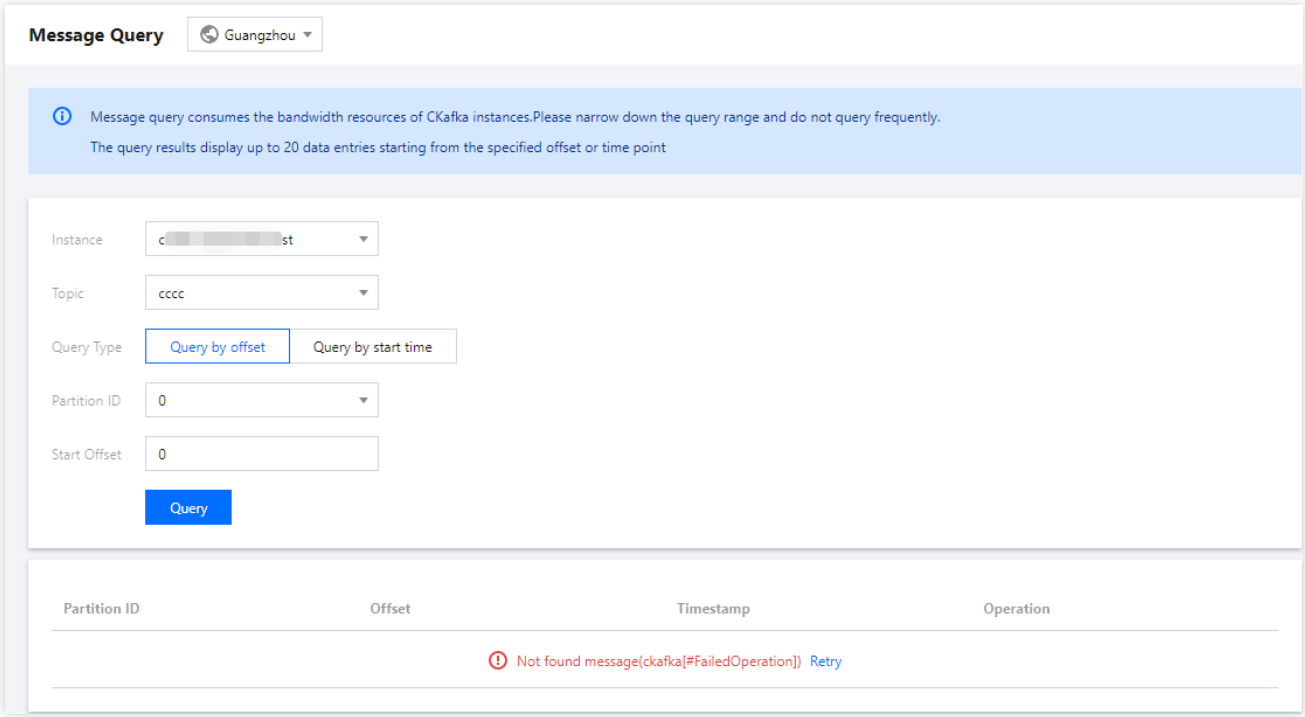
```
        properties.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
                "org.apache.kafka.common.serialization.StringSerializer");
        properties.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
                "org.apache.kafka.common.serialization.StringSerializer");
        //Set the maximum time to wait for a request
        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 interval between retries 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. View the execution result.



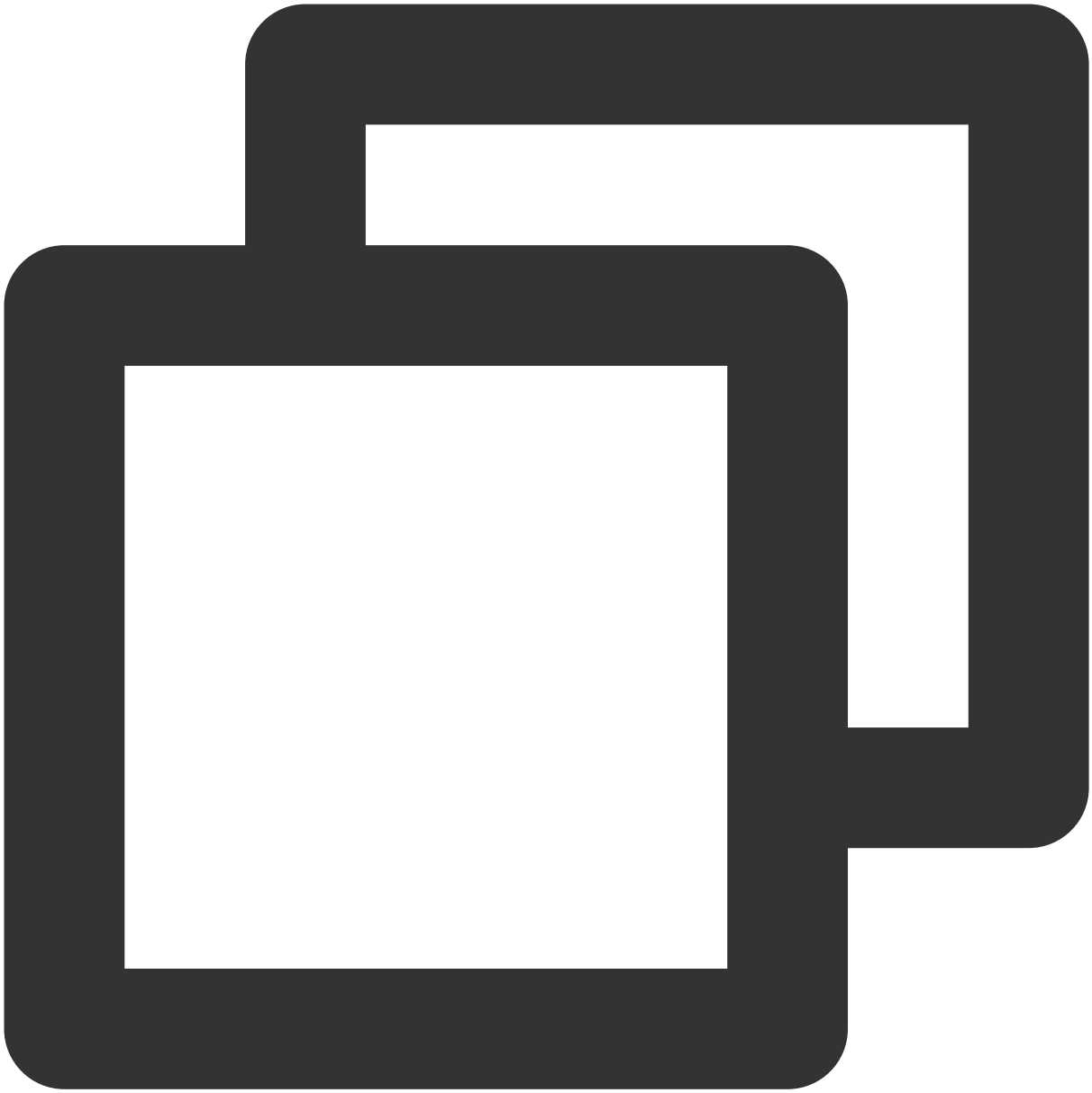
```
Produce ok:ckafka-topic-demo-0@198
Produce ok:ckafka-topic-demo-0@199
```

3. Go to the [CKafka console[(https://console.tencentcloud.com/ckafka!85c1cf838df0405887dc01b41e7972fc), select the **Topic Management** tab on the instance details page, select the target topic, and click **More** > **Message Query** to view the message just sent.



Step 3. Consume messages

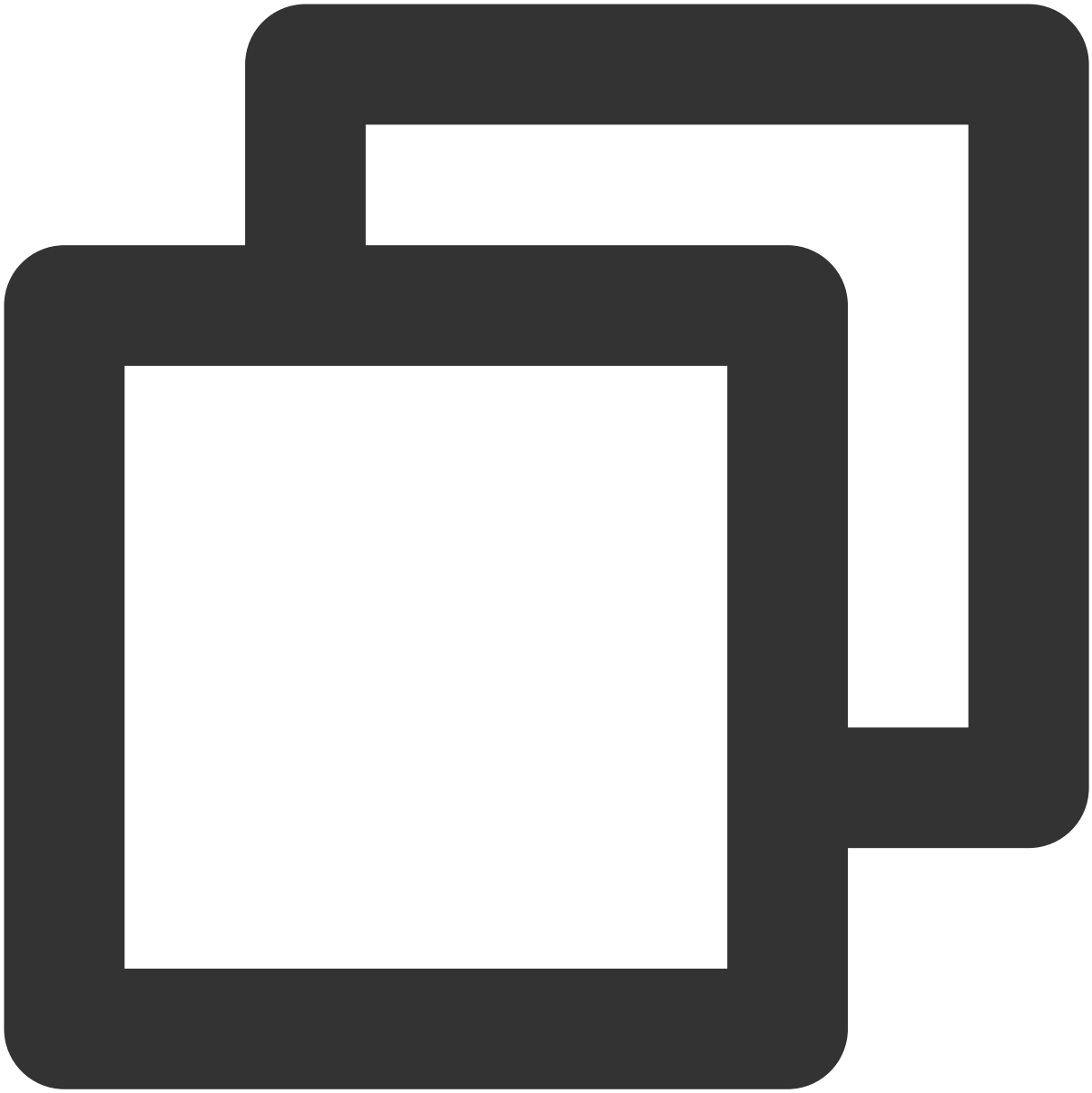
1. Compile and run the message subscription program `CKafkaConsumerDemo.java` .



```
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,
```

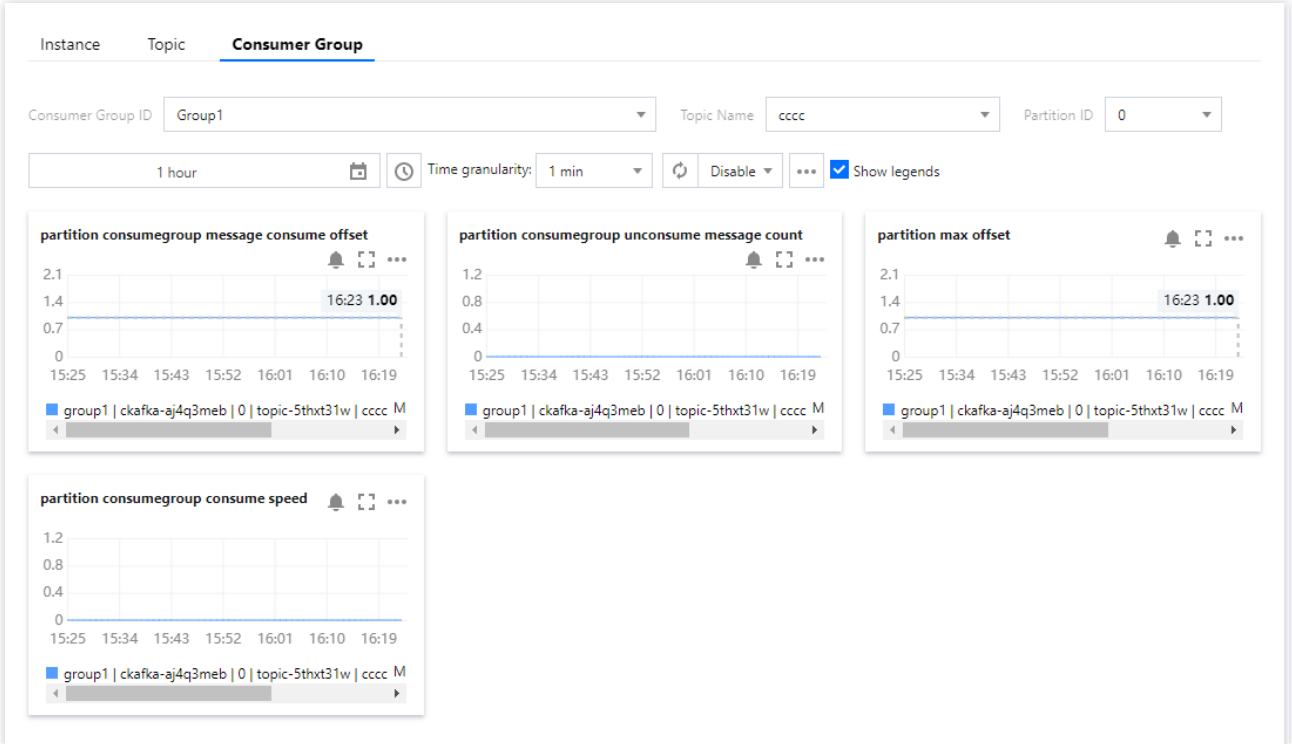
```
        props.put(ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG, 30000);
        //Set the maximum number of messages that can be polled at a time
        //Do not set this parameter to an excessively large value. If polled messag
        props.put(ConsumerConfig.MAX_POLL_RECORDS_CONFIG, 30);
        //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
        //Construct a consumer object. This generates 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. View the execution result.



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

3. 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.



Running Kafka Client (Optional)

Last updated：2024-01-09 14:45:02

Overview

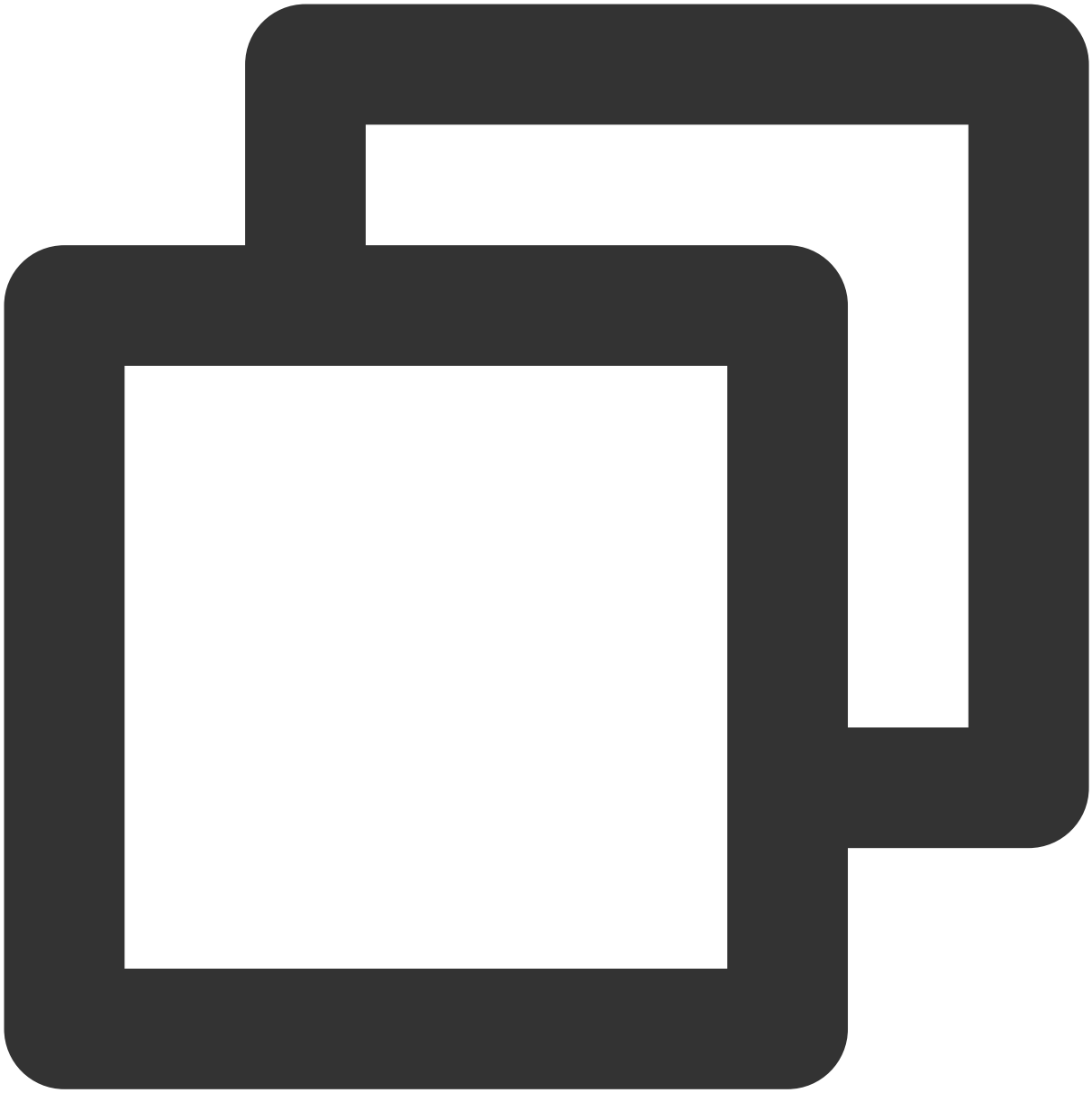
This document explains how to start using Kafka APIs after you purchase the CKafka service. After setting up a CKafka environment on a CVM instance, you need to download and decompress the Kafka installation file and perform simple testing on Kafka APIs.

Directions

Step 1. Install a JDK.

1. Check Java installation.

Open a terminal window and run this command:



```
java -version
```

If the output of the command is a Java version number, then Java is already installed in your system. If you have not installed Java yet, download and install a Java Development Kit (JDK).

2. Set up the Java environment.

Set the `JAVA_HOME` environment variable and point it to the Java installation directory on your machine.
For example, if you use Java JDK 1.8.0_20, the outputs on different operating systems are as follows:

Supported Operating	Output

Systems	
Windows	Set the environment variable JAVA_HOME to C:\\Program Files\\Java\\jdkjdk1.8.0_20
Linux	export JAVA_HOME=/usr/local/java-current
Mac OSX	export JAVA_HOME=/Library/Java/Home

Add the Java compiler path to the system path:

Supported Operating Systems	Output
Windows	Add `;C:\\Program Files\\Java\\jdk1.8.0_20\\bin` to the end of the system variable `Path`
Linux	export PATH=$PATH:$JAVA_HOME/bin/
Mac OSX	not required

Use the `java -version` command to check your Java installation.

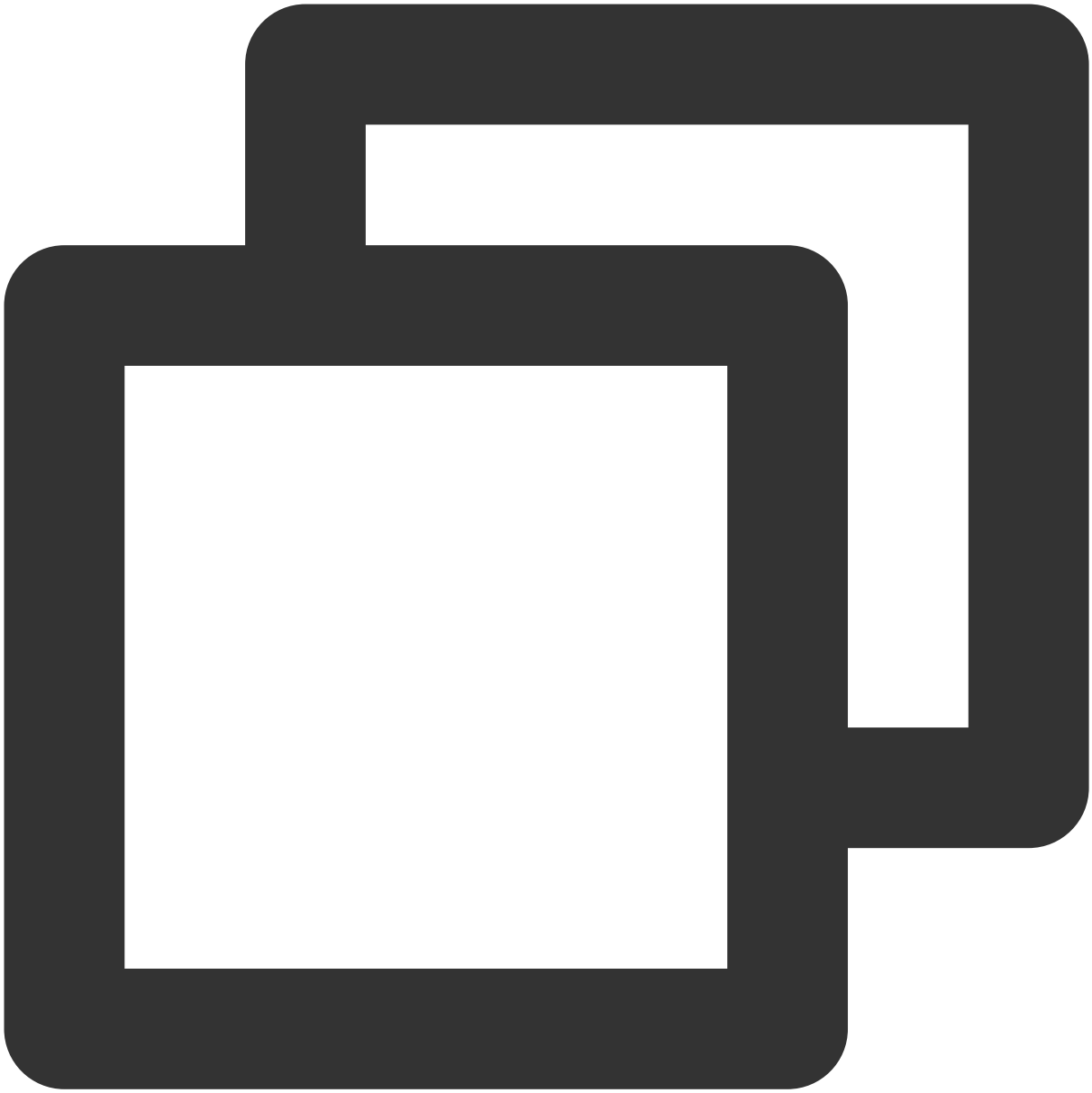
Step 2. Download the Kafka installation file.

Download and decompress the Kafka installation file.

Step 3. Test Kafka APIs.

Go to the `./bin` directory, and produce and consume a message via CLI commands.

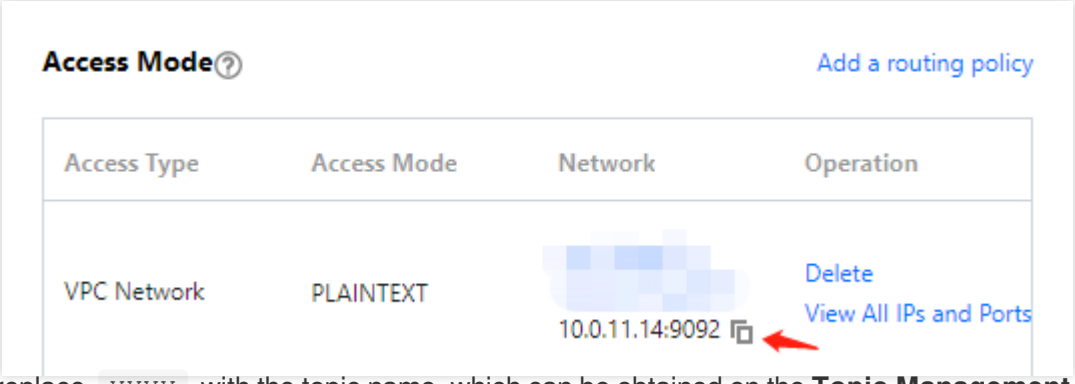
1. Open a terminal window to start a consumer.



```
bash kafka-console-consumer.sh --bootstrap-server XXXX:port --topic XXXX --consumer
```

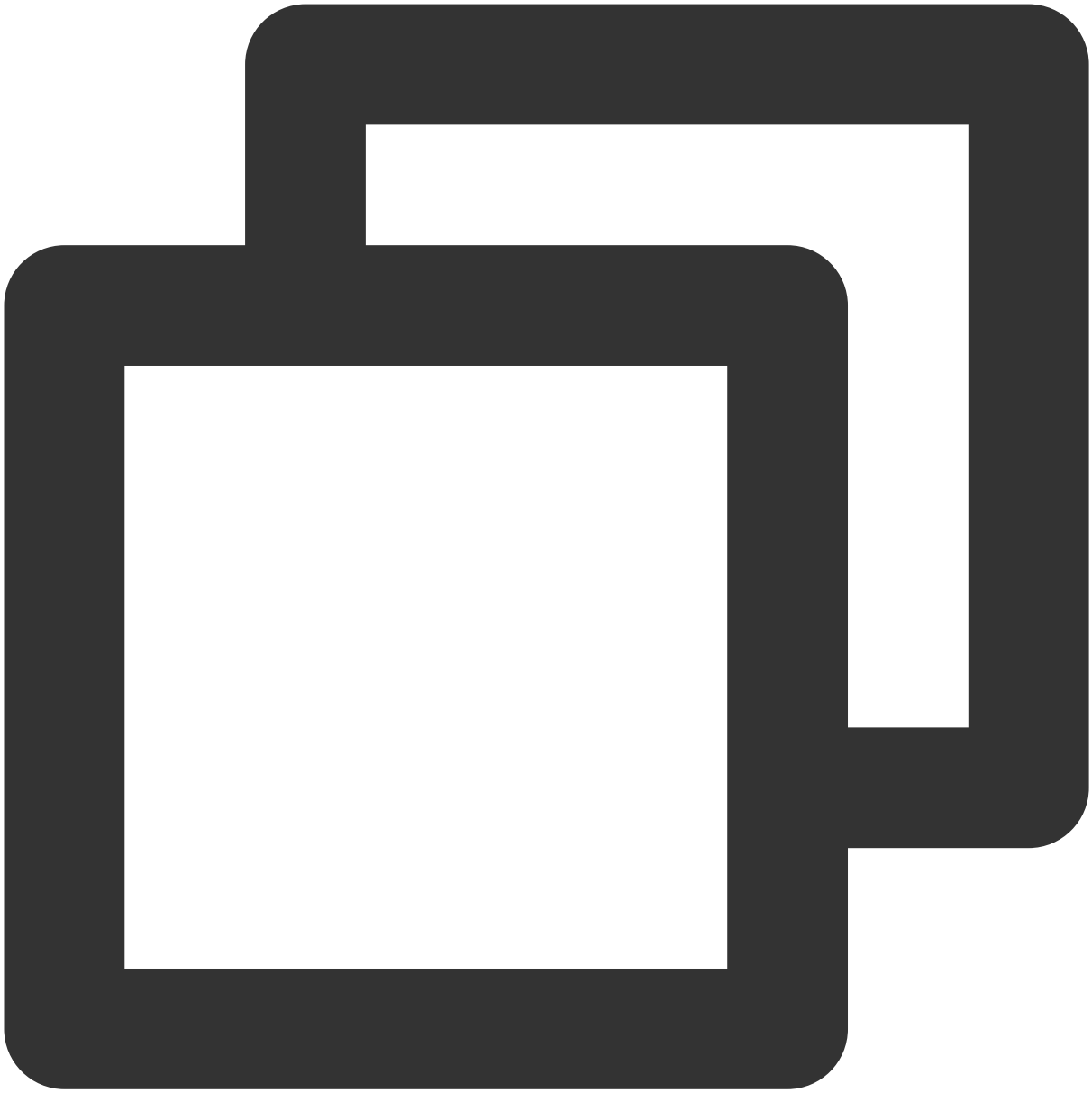
Note:

Replace `XXXX:port` with the domain name and port for VPC access, which can be obtained in the **Access Mode** section on the **Instance Details** page in the console.



topic: replace `XXXX` with the topic name, which can be obtained on the **Topic Management** page in the console.

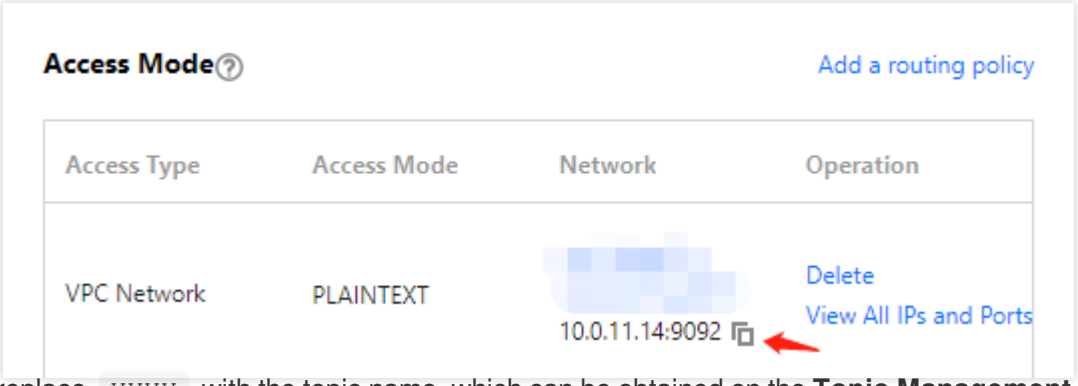
2. Open another terminal window to start a producer.



```
bash kafka-console-producer.sh --broker-list XXXX:port --topic XXXX --producer.conf
```

Note:

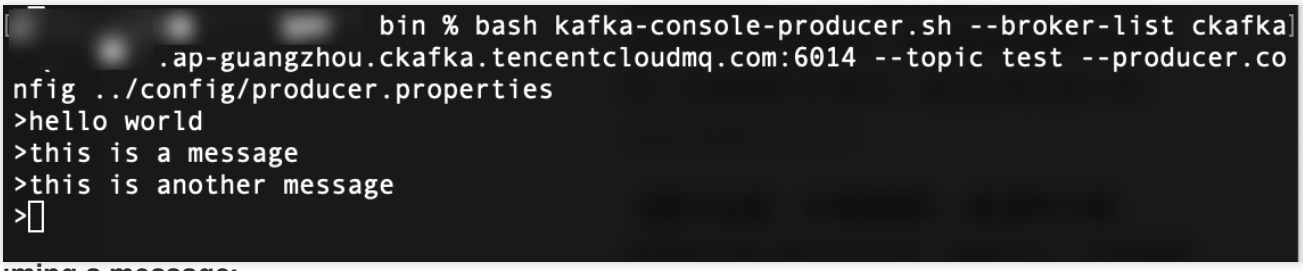
Replace `XXXX:port` with the domain name and port for VPC access, which can be obtained in the **Access Mode** section on the **Instance Details** page in the console.



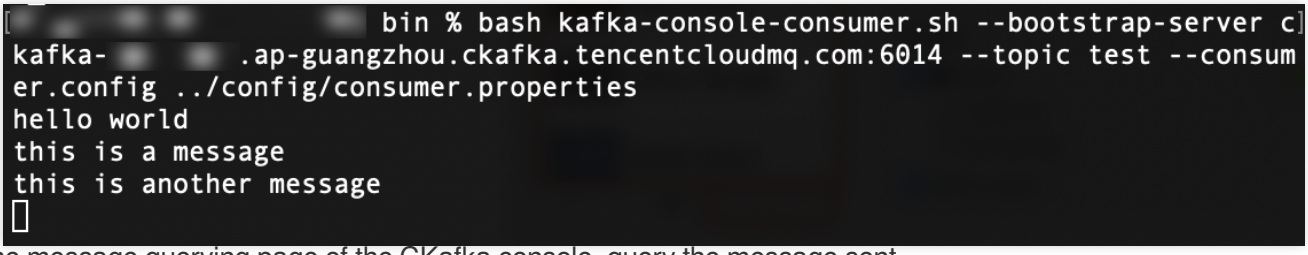
topic: replace `XXXX` with the topic name, which can be obtained on the **Topic Management** page in the console.

Enter the content of the message and press Enter.

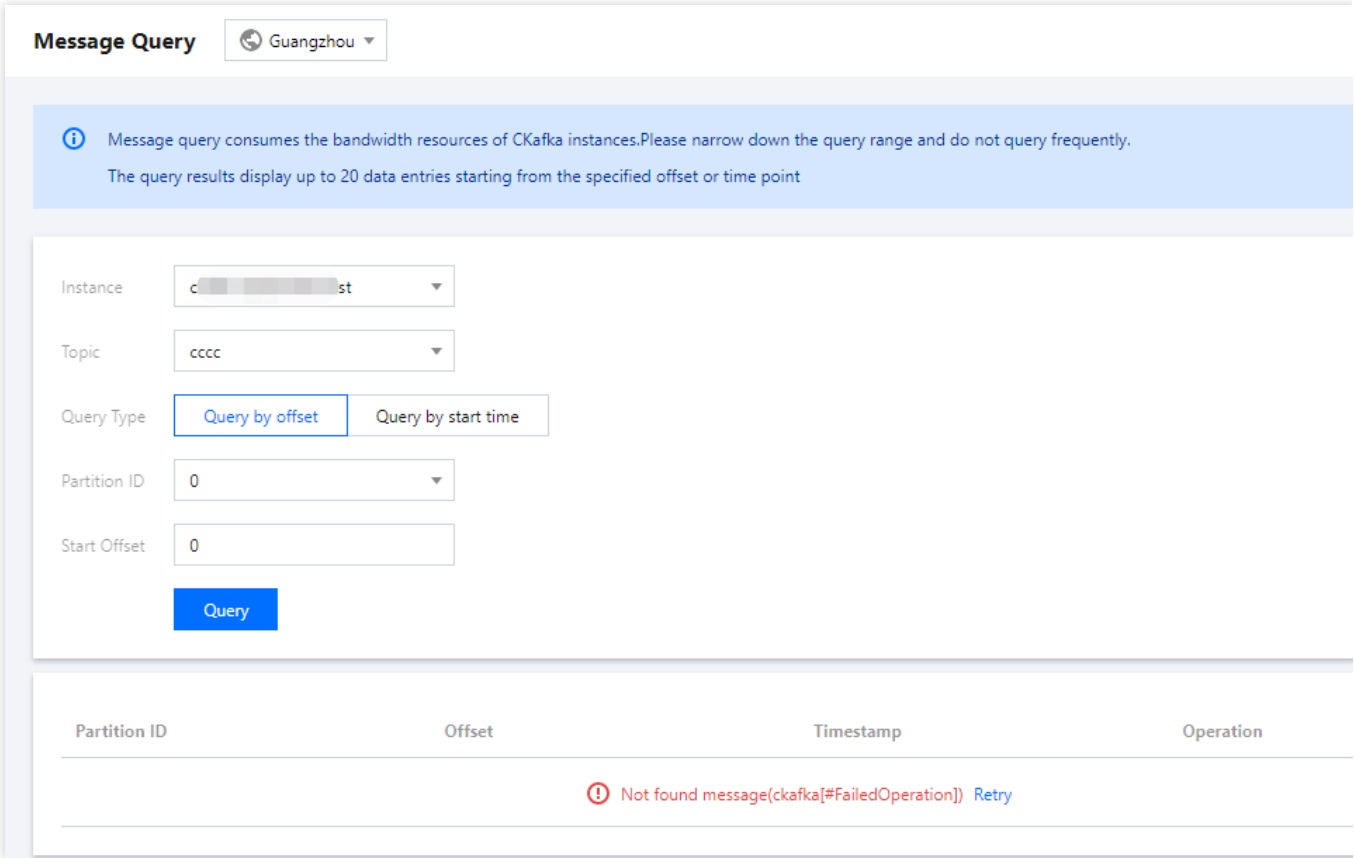
Producing a message:



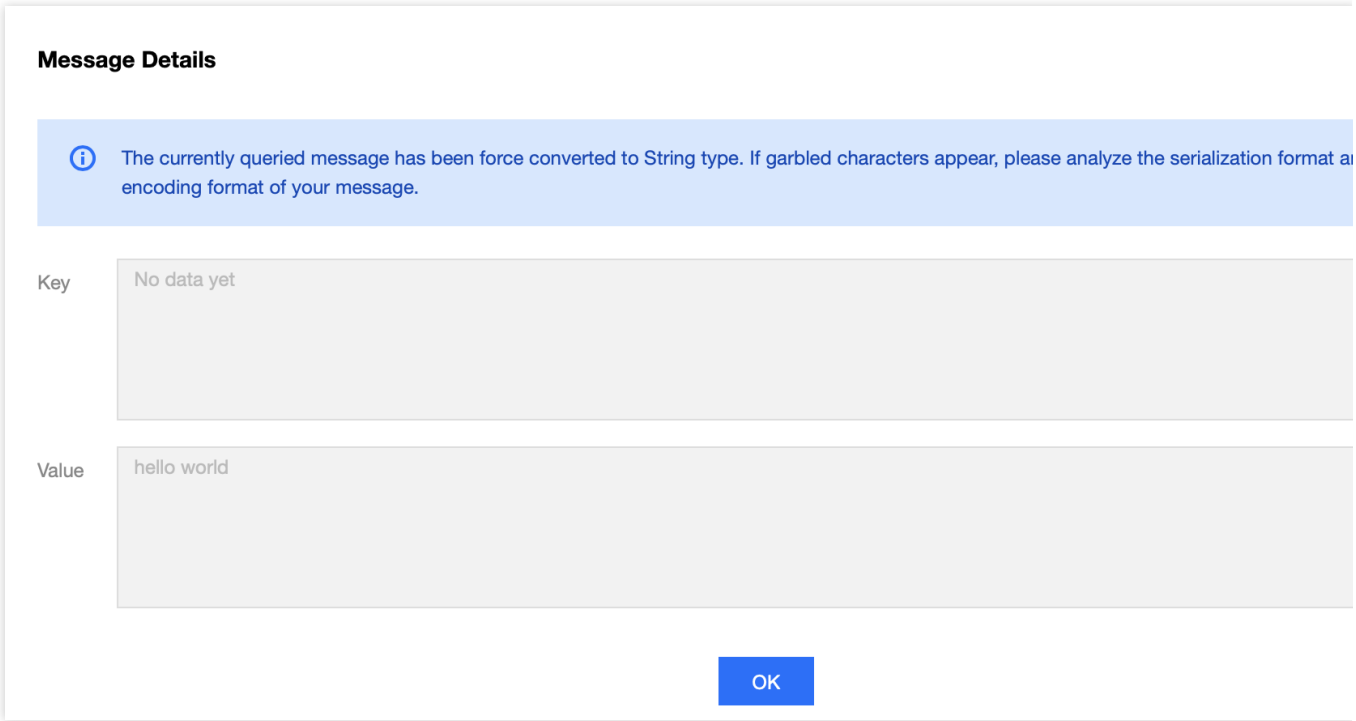
Consuming a message:



3. In the message querying page of the CKafka console, query the message sent.



The details of the message are as follows:



Access via Public Domain Name
Step 1. Create an Instance

Last updated：2024-01-09 14:45:02

Overview

This document describes how to create an instance in the CKafka console.

Prerequisites

You have signed up for a Tencent Cloud account.
You have created a VPC.

Directions

1. Log in to the CKafka console.
2. Select **Instance List** on the left sidebar, click **Create** to go to the instance purchase page, and enter the purchase information as needed.
Billing Mode: Pro Edition instances support both monthly subscription and pay-as-you-go billing, while Standard Edition instances only support monthly subscription.
Specs Type: CKafka instances are divided into Standard Edition and Pro Edition based on their specifications. For the comparison between the two editions, see Product Specifications.
Kafka Version: Select a Kafka version based on your business needs. For more information, see Suggestions for CKafka Version Selection.
Region: Select a region close to the resource for client deployment.
AZ:
Standard Edition: This edition does not support multi-AZ deployment.
Pro Edition: If multi-AZ deployment is supported in the current region, you can select up to four AZs for deployment. For more information, see Multi-AZ Deployment.
Product Specs: Select a model based on the peak bandwidth and disk capacity.
Message Retention Period: Select a value between 24 and 2,160 hours. The default value is 24 hours Expired messages will be deleted to free up disk space.
When the disk capacity is insufficient (that is, the disk utilization reaches 90%), previous messages will be deleted in advance to ensure service availability.

VPC: Select a suitable VPC based on your business needs.
Public Network Bandwidth: By default, the Standard Edition and Pro Edition instances offer 1 and 3 Mbps public network bandwidth free of charge respectively. You can pay to upgrade the public network bandwidth for your Pro Edition instance as needed For more information, see Public Network Bandwidth Management.
Instance Name: When purchasing multiple instances, you can create instances in batches by their numeric suffix (which is numbered in ascending order) or their designated pattern string. For more information, see Naming with Consecutive Numeric Suffixes or Designated Pattern String.
3. Click **Buy Now**. The created instance will be displayed in the instance list in about 3–5 minutes.

Step 2. Add a Public Route

Last updated：2024-01-09 14:45:02

Overview

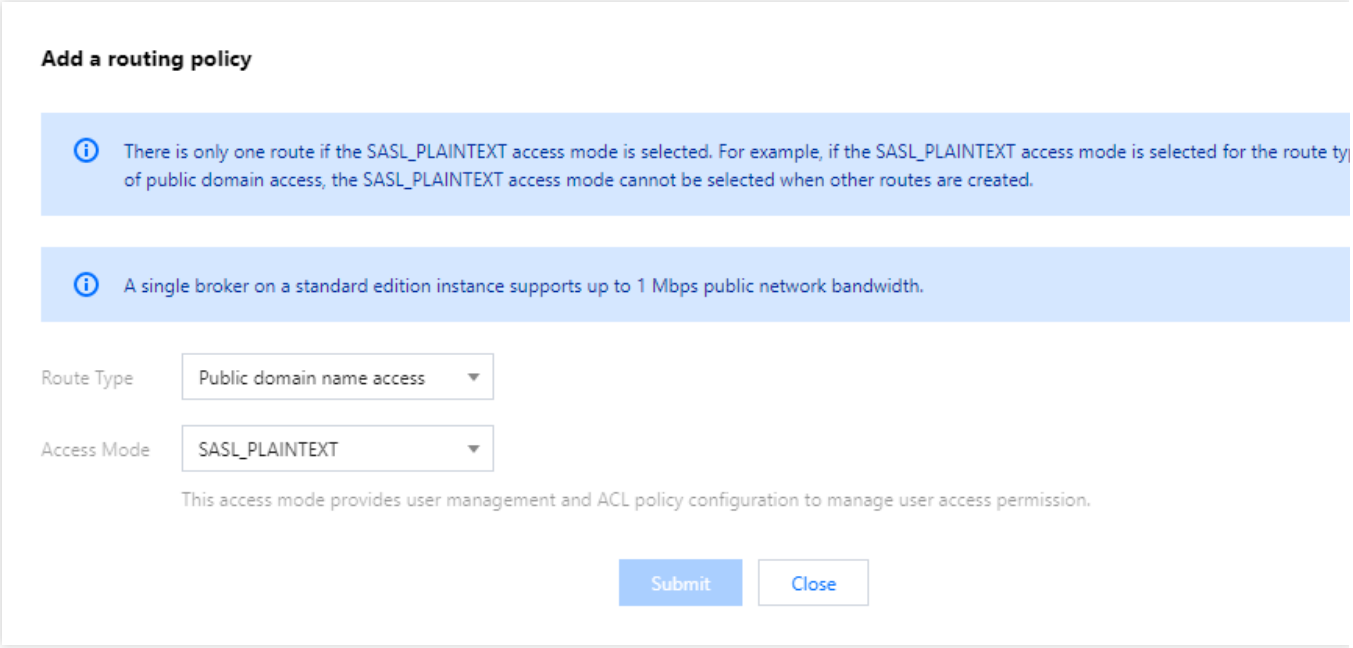
To enable public network access, you need to add a public route for the instance. This document describes how to add a public route for a created instance.

Prerequisites

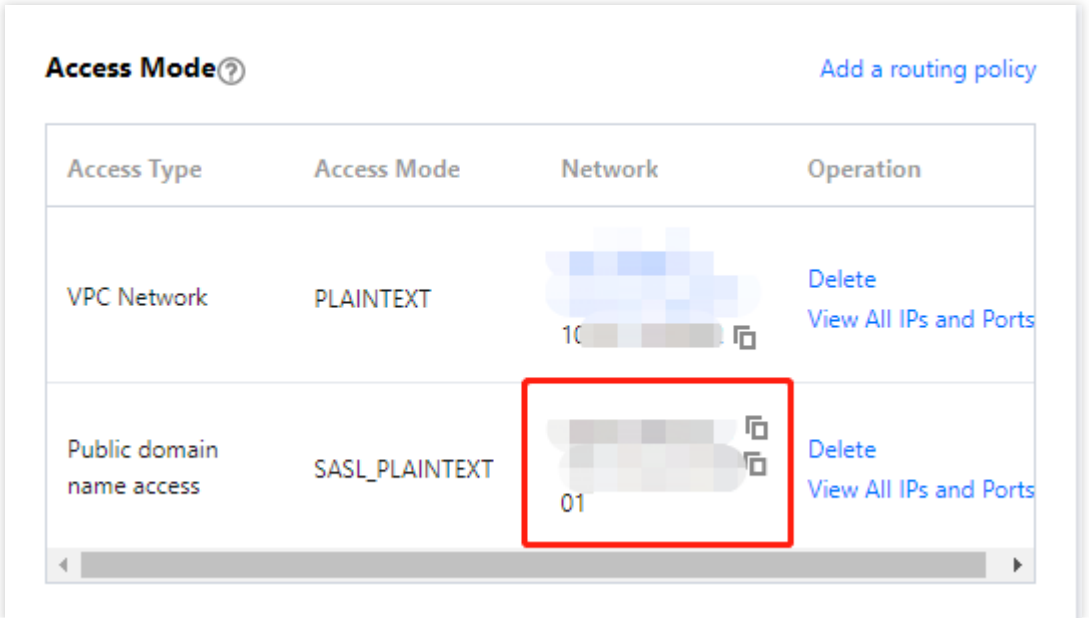
You have created an instance. For more information, see Step 1. Create an Instance.

Directions

1. On the Instance List page, click the ID/name of the instance created in Step 1. Create an Instance.
2. On the instance details page, click **Add a routing policy** in the **Access Mode** section to add a public network route.



3. Then you get the domain name and port for public network access.



Step 3. Create a Topic

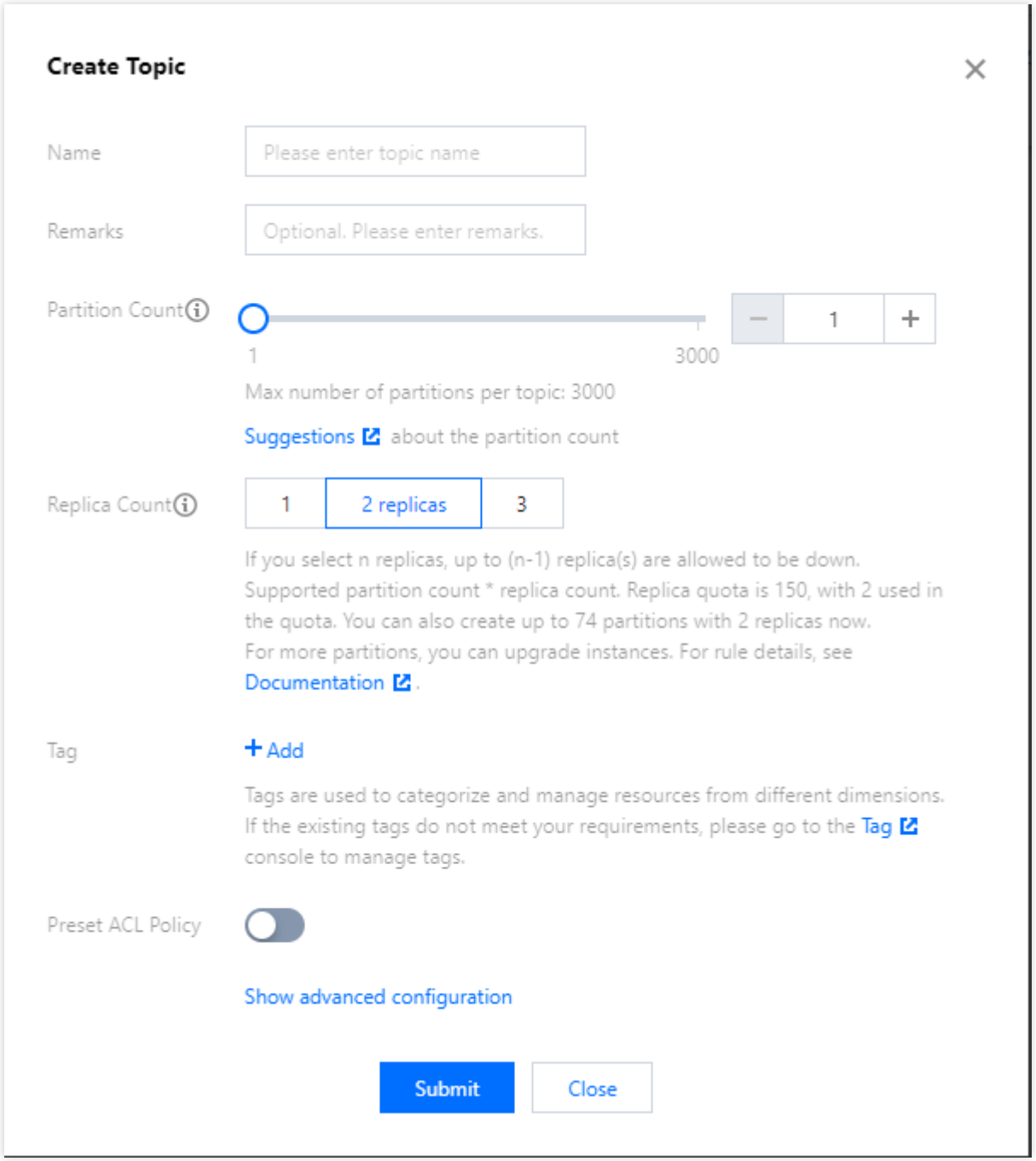
Last updated：2024-01-09 14:45:02

Overview

This document describes how to create a topic under an existing instance in the CKafka console.

Directions

1. Log in to the CKafka console.
2. On the **Instance List** page, click the **ID/Name** of the instance created in Step 1. Create an Instance to enter the instance details page.
3. On the instance details page, click **Topic Management** at the top of the page, and click **Create**.
4. In the **Create Topic** dialog box, set parameters as needed.



Name: The topic name. It cannot be changed once entered and can contain only letters, digits, underscores, or symbols ("-" and ".").

Partition Count: It is a concept in physical partition, where one topic can contain one or more partitions. CKafka uses partition as a message allocation unit.

Replica Count: The number of partition replicas is used to ensure the high availability of the partition. To ensure data reliability, creating a single-replica topic is not supported. Two replicas are enabled by default.

Replicas are also counted into the number of partitions. For example, if you create 1 topic with 6 partitions, and 2 replicas for each partition, then you have a total of 12 partitions (1 x 6 x 2).

Tag: Set a resource tag. For more information, see Tag Overview.

Preset ACL Policy: Select the preset ACL policy. For more information on ACL policy, see Configuring ACL Policy.

5. Click **Submit**.

Step 4. Configure an ACL Policy

Last updated：2024-01-09 14:45:02

Overview

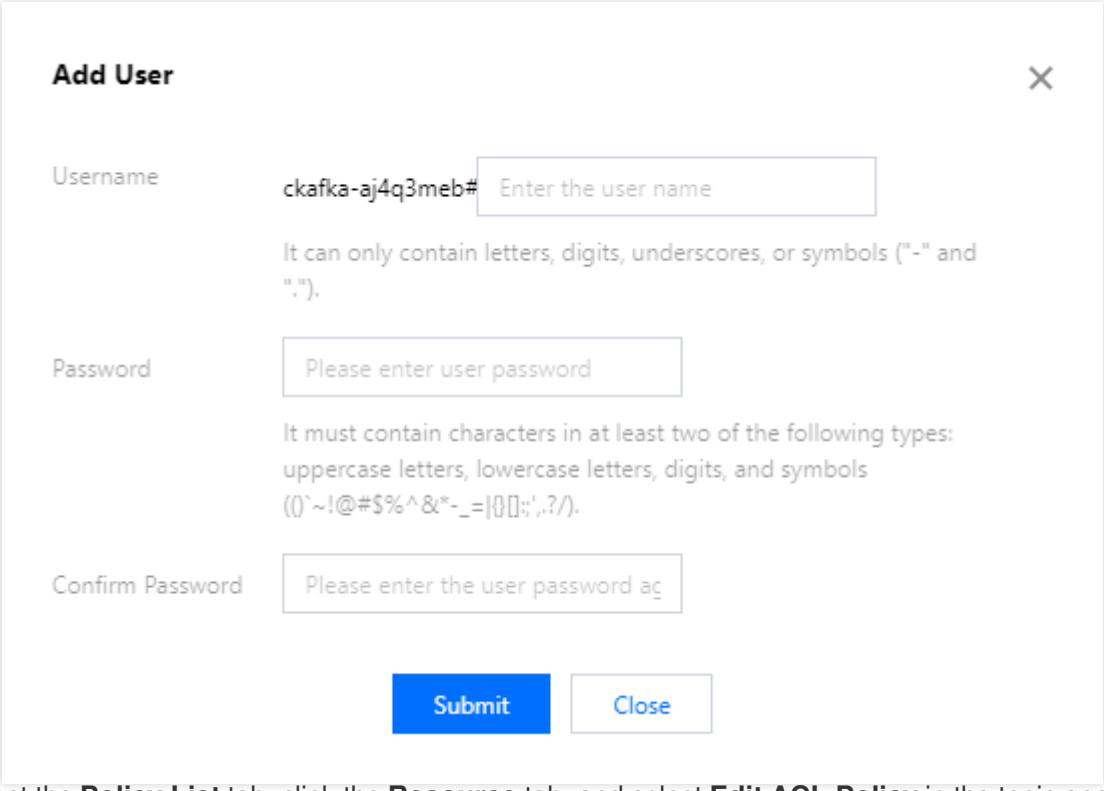
To enable public network access, you need to configure an ACL policy for a topic. This document describes how to configure an ACL policy for a created topic in the CKafka console.

Prerequisites

You have created a topic. For more information, see Step 3. Create a Topic

Directions

1. On the instance details page, select **ACL Policy Management** > **User Management** and click **Create** to add a user and set the username and password.



2. Select the **Policy List** tab, click the **Resource** tab, and select **Edit ACL Policy** in the topic operation column created in Step 3. Create a Topic to add read and write permissions for the user.

Step 5. Send/Receive Messages Using SDK to Receive/Send Message (Recommended)

Last updated：2024-01-09 14:45:02

Overview

This document describes how to access CKafka to receive/send messages with the SDK for Java on the public network. For clients in other languages, see SDK Documentation.

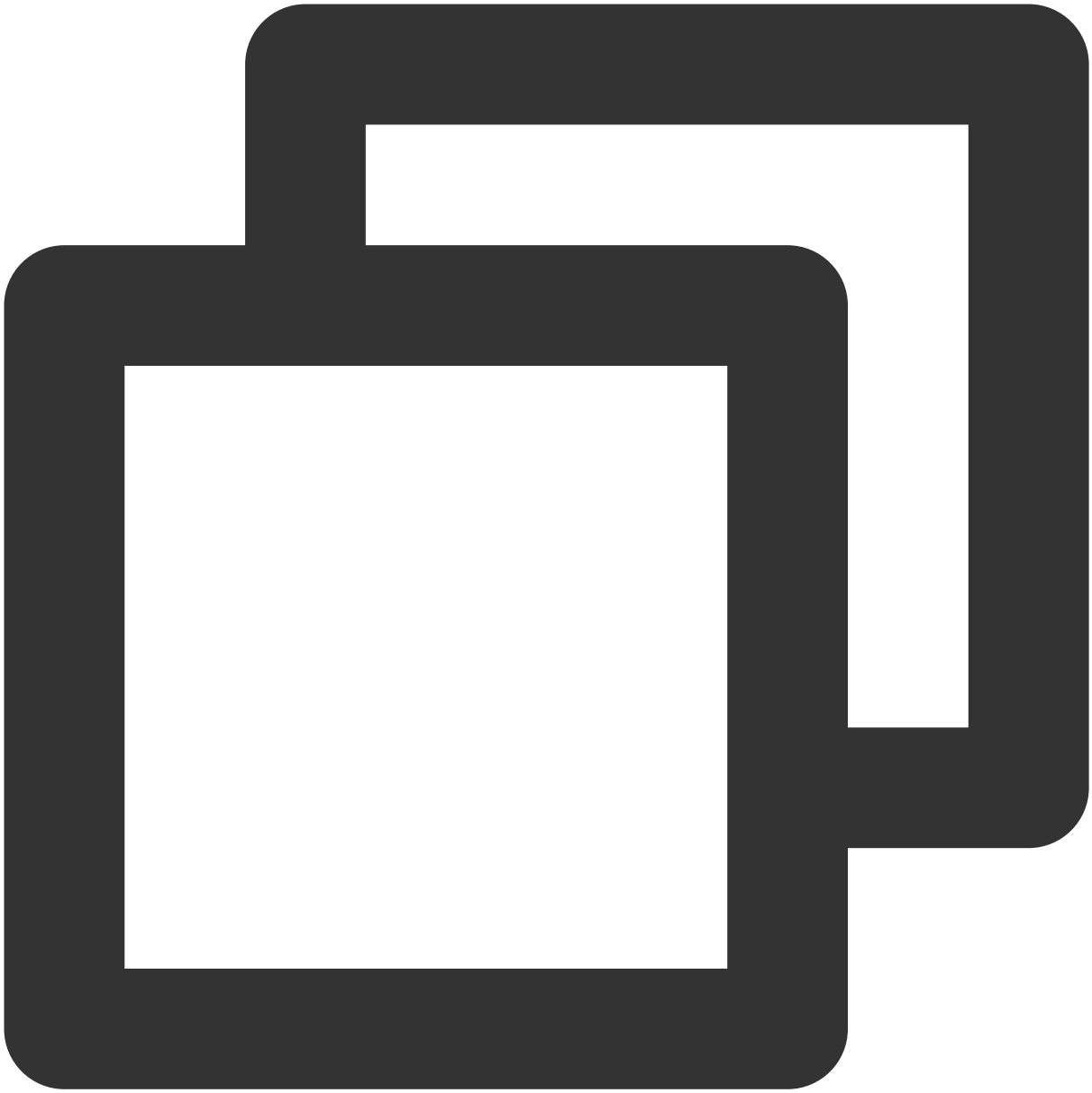
Prerequisites

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. Decompress the downloaded demo and enter the `PUBLIC_SASL` directory under `javakafkademo` .
2. Modify a JAAS configuration file named `ckafka_client_jaas.conf` .

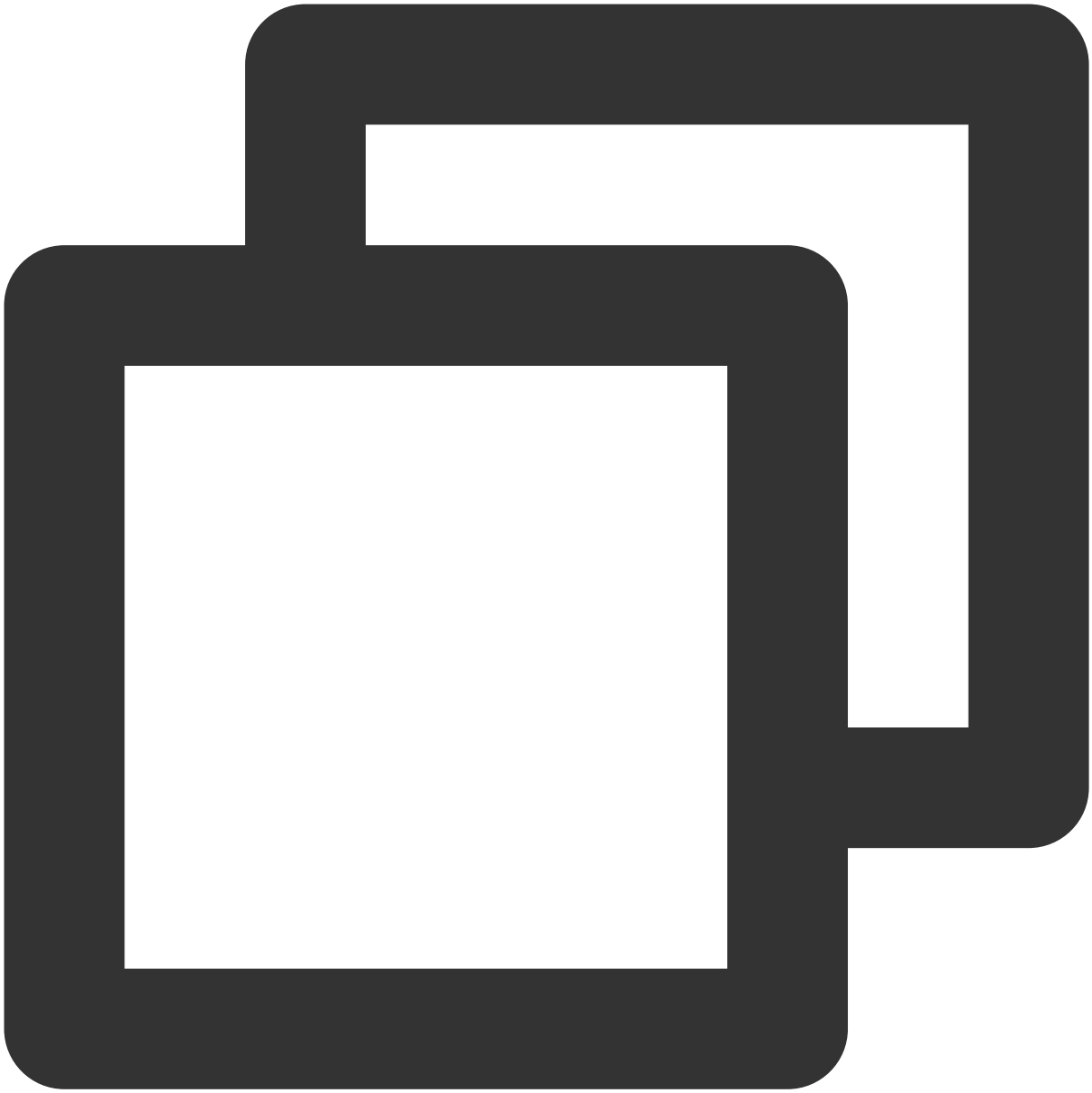


```
KafkaClient {
org.apache.kafka.common.security.plain.PlainLoginModule required
username="yourinstance#yourusername"
password="yourpassword";
};
```

Note:

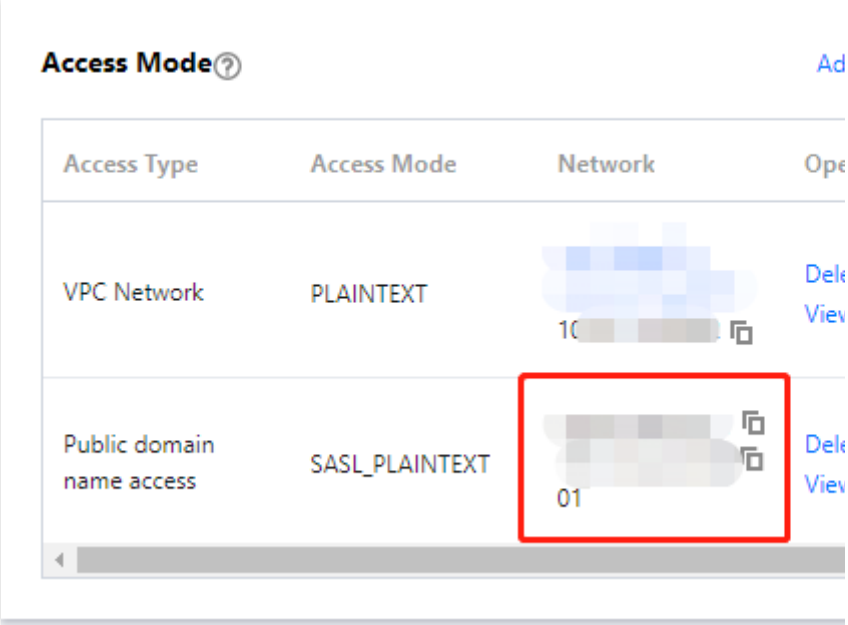
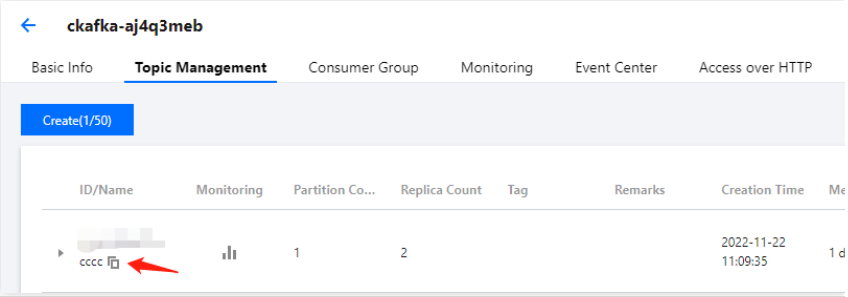
Set `username` to a value in the format of `instance ID` + `#` + `configured username`, and `password` to a configured password.

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



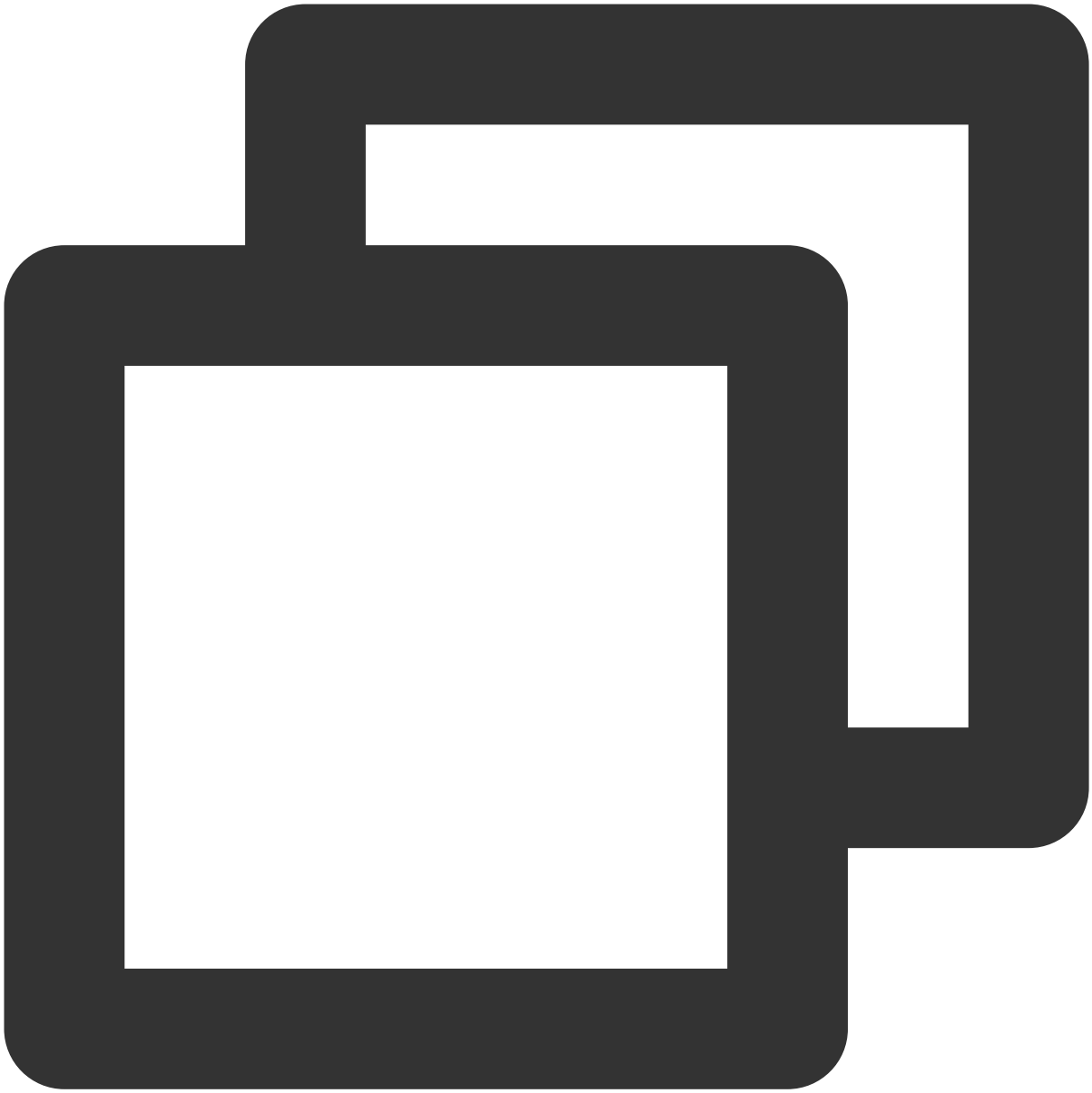
```
## Configure the accessed network by copying the information in the **Network** col
bootstrap.servers=ckafka-xxxxxxx
## Configure the topic by copying the information on the **Topic Management** page
topic=XXX
## Configure the consumer group as needed
group.id=XXX
## The path of the JAAS configuration file named `ckafka_client_jaas.conf`
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 on the **Topic Management** page in the co 
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`

Step 2. Send messages

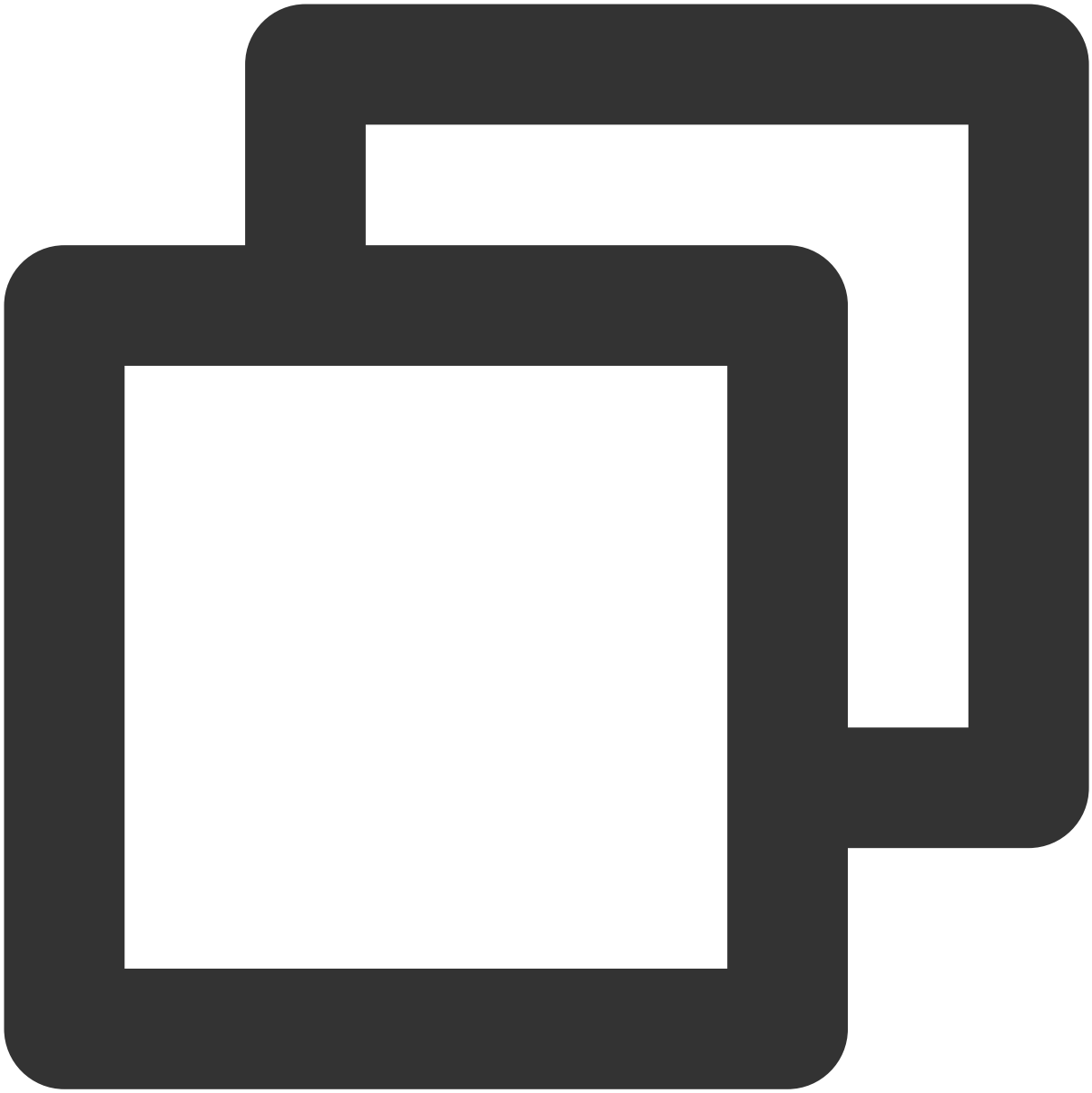
1. Compile and run the message sending program `CKafkaSaslProducerDemo.java` .



```
public class CKafkaSaslProducerDemo {

    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
```

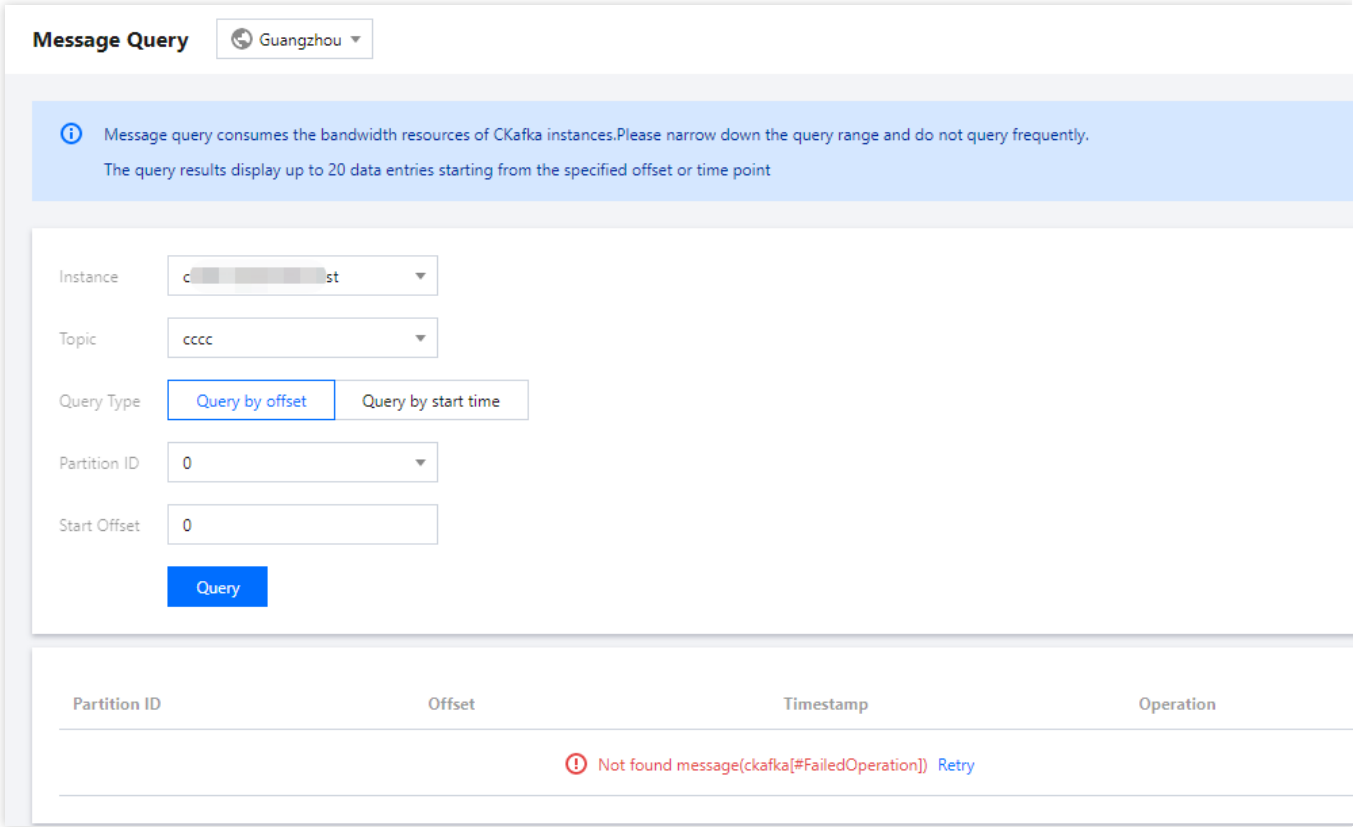
```
            props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, kafkaProperties.get

            // Set the access protocol.
            props.put(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG, "SASL_PLAINTEX
            // Set the PLAIN mechanism.
            props.put(SaslConfigs.SASL_MECHANISM, "PLAIN");

            // Set the method of serializing CKafka messages.
            props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, "org.apache.kafk
            props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, "org.apache.ka
            // 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);
            // Construct a producer object. Note that a producer object is thread-
            KafkaProducer<String, String> producer = new KafkaProducer<>(props);

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

            try {
                    // Obtaining the future objects in batches can accelerate the
                    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 Prod
                            Future<RecordMetadata> metadataFuture = producer.send(
                            futures.add(metadataFuture);

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

2. View the execution result (output).



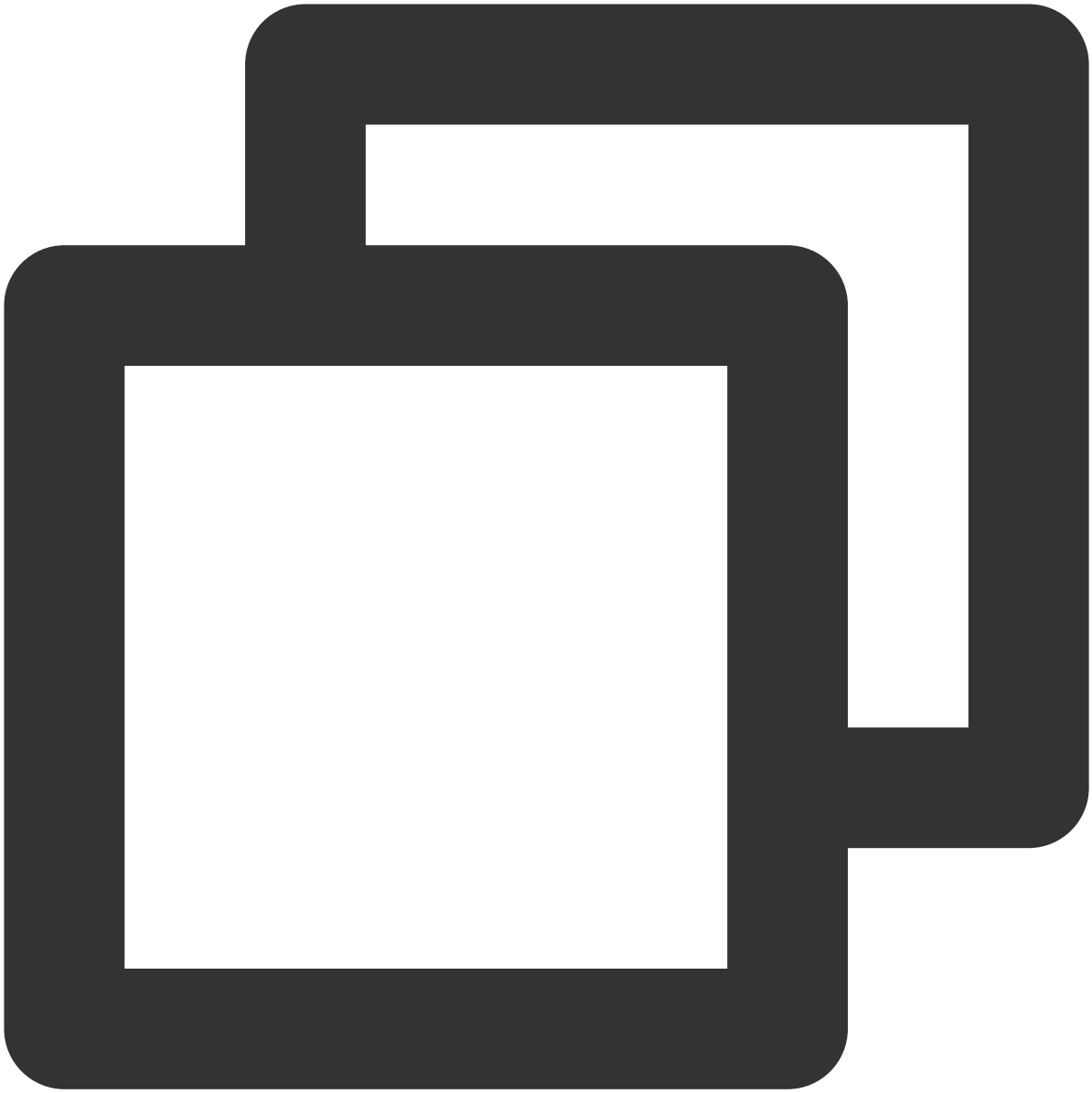
```
Produce ok:ckafka-topic-demo-0@198
Produce ok:ckafka-topic-demo-0@199
```

3. 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

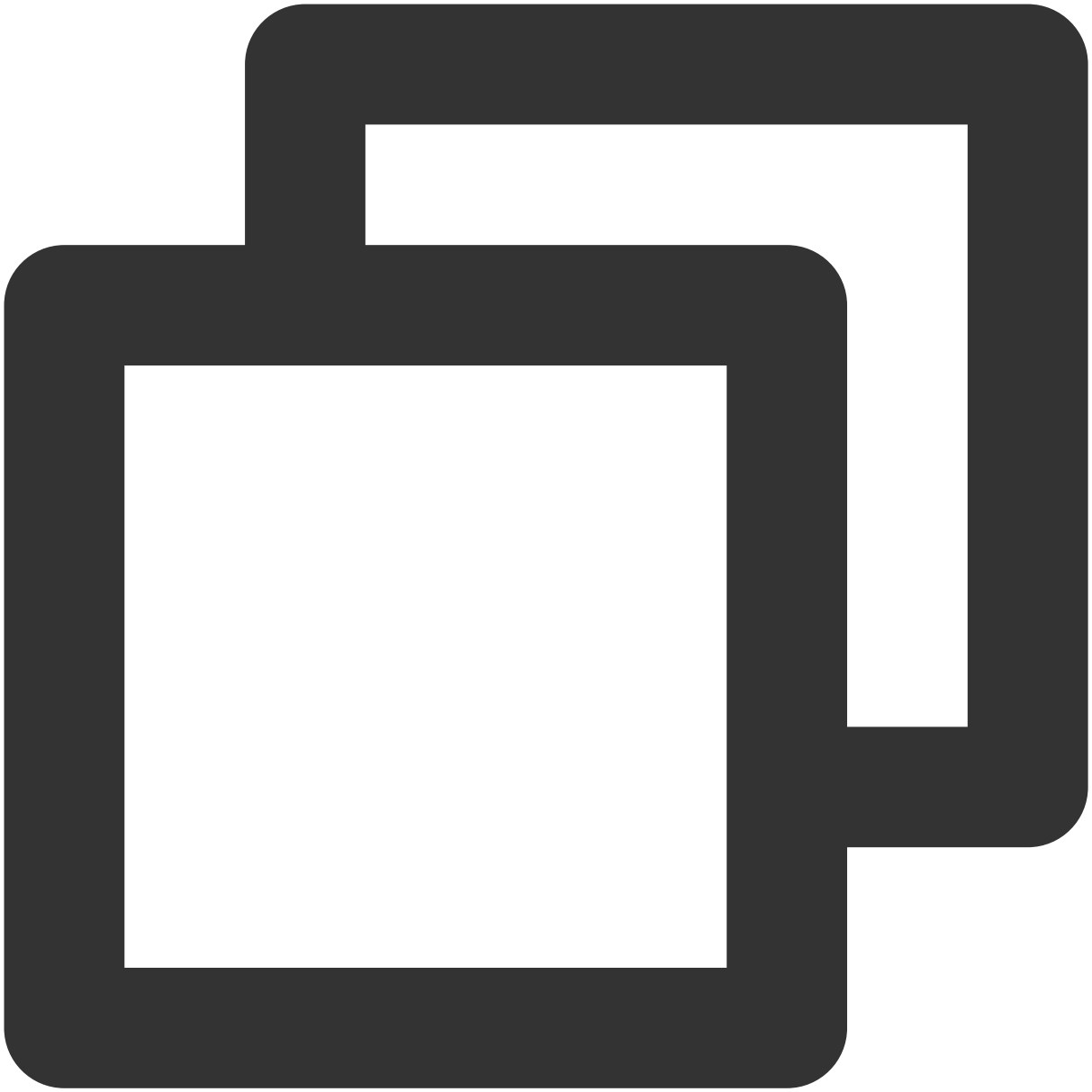
1. Compile and run the message subscription program `CKafkaSaslConsumerDemo.java` .



```
public class CKafkaSaslConsumerDemo {
    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
        props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, kafkaProperties.getPrope
```

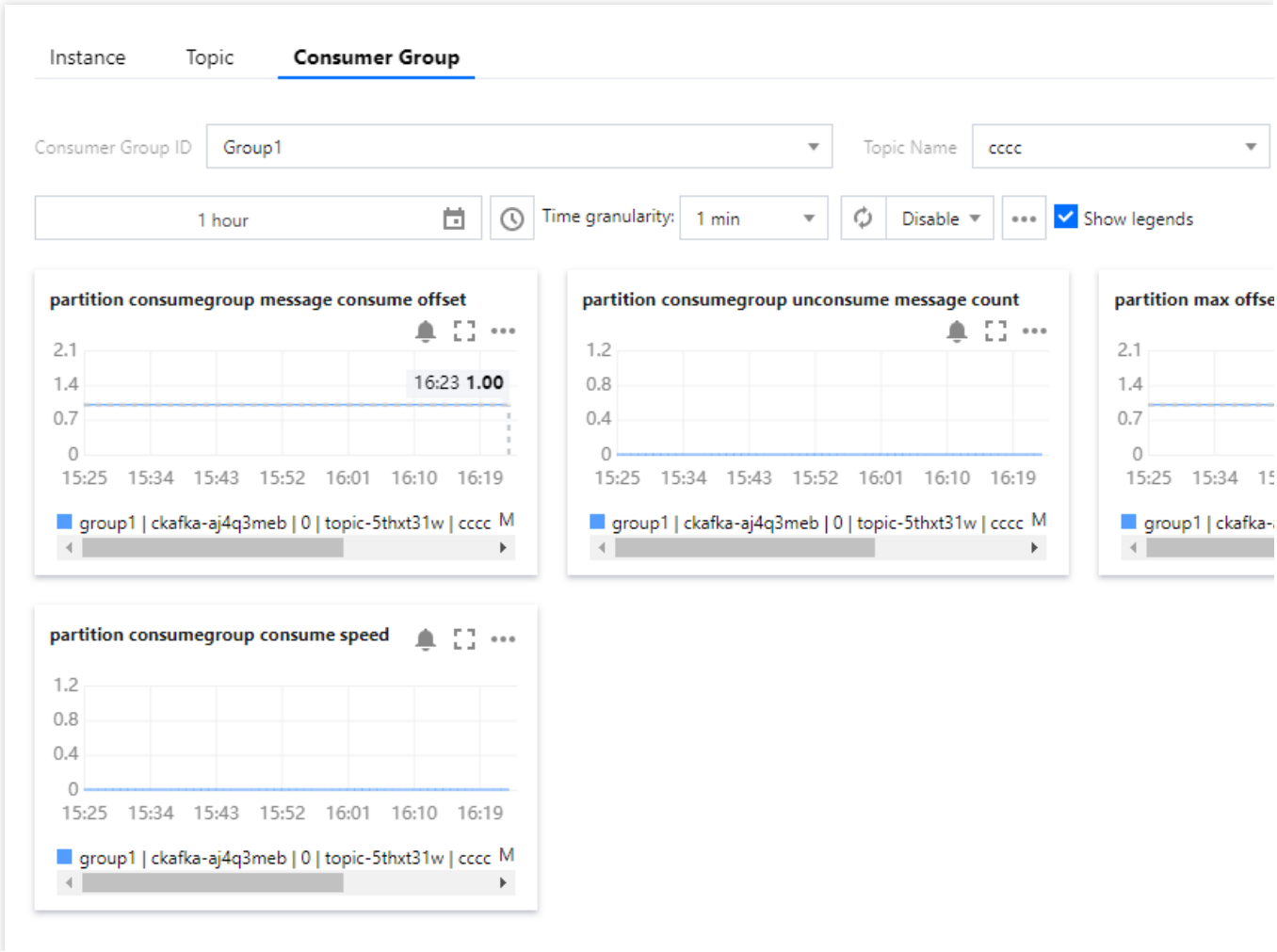
```
        // Set the access protocol.
        props.put(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG, "SASL_PLAINTEXT");
        // Set the PLAIN mechanism.
        props.put(SaslConfigs.SASL_MECHANISM, "PLAIN");
        // Set the maximum interval between two polls.
        // If the consumer does 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 messa
        props.put(ConsumerConfig.MAX_POLL_RECORDS_CONFIG, 30);
        // Set the method of deserializing messages.
        props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, "org.apache.kafka.c
        props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, "org.apache.kafka
        // Set the consumer group of the current consumer instance after you apply
        // The instances in the same consumer group consume messages in load balanc
        props.put(ConsumerConfig.GROUP_ID_CONFIG, kafkaProperties.getProperty("grou
        // Construct a consumer object. This generates a consumer instance.
        KafkaConsumer<String, String> consumer = new KafkaConsumer<String, String>(
        // Set one or more topics to which the consumer group subscribes.
        // We recommend that you configure consumer instances with the same `GROUP_
        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 tota
                for (ConsumerRecord<String, String> record : records) {
                    System.out.println(String.format("Consume partition:%d offset:%
                }
            } catch (Exception e){
                System.out.println("consumer error!");
            }
        }
    }
}
```

2. View the execution result.



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

3. 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.



Running Kafka Client (Optional)

Last updated：2024-01-09 14:45:02

Overview

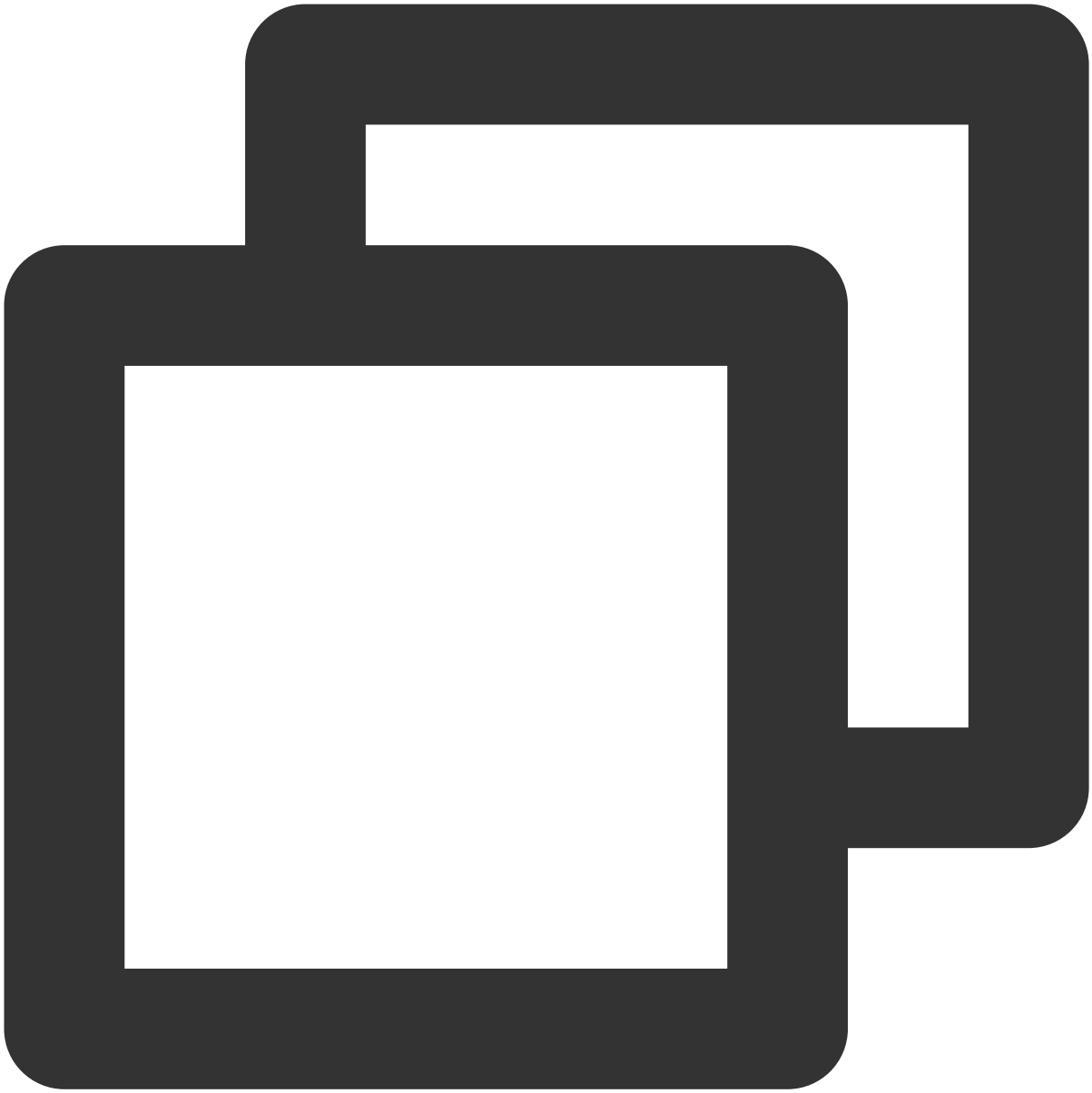
This document explains how to start using Kafka APIs after you purchase the CKafka service. You can download and decompress the Kafka installation file and perform simple testing on Kafka APIs.

Directions

Step 1. Install a JDK.

1. Check Java installation.

Open a terminal window and run this command:



```
java -version
```

If the output of the command is a Java version number, then Java is already installed in your system. If you have not installed Java yet, download and install a Java Development Kit (JDK).

2. Set up the Java environment.

Set the `JAVA_HOME` environment variable and point it to the Java installation directory on your machine.
For example, if you use Java JDK 1.8.0_20, the outputs on different operating systems are as follows:

Supported Operating	Output

Systems	
Windows	Set the environment variable JAVA_HOME to C:\\Program Files\\Java\\jdkjdk1.8.0_20
Linux	export JAVA_HOME=/usr/local/java-current
Mac OSX	export JAVA_HOME=/Library/Java/Home

Add the Java compiler path to the system path:

Supported Operating Systems	Output
Windows	Add `C:\\Program Files\\Java\\jdk1.8.0_20\\bin` to the end of the system variable `Path`
Linux	export PATH=$PATH:$JAVA_HOME/bin/
Mac OSX	not required

Use the `java -version` command to check your Java installation.

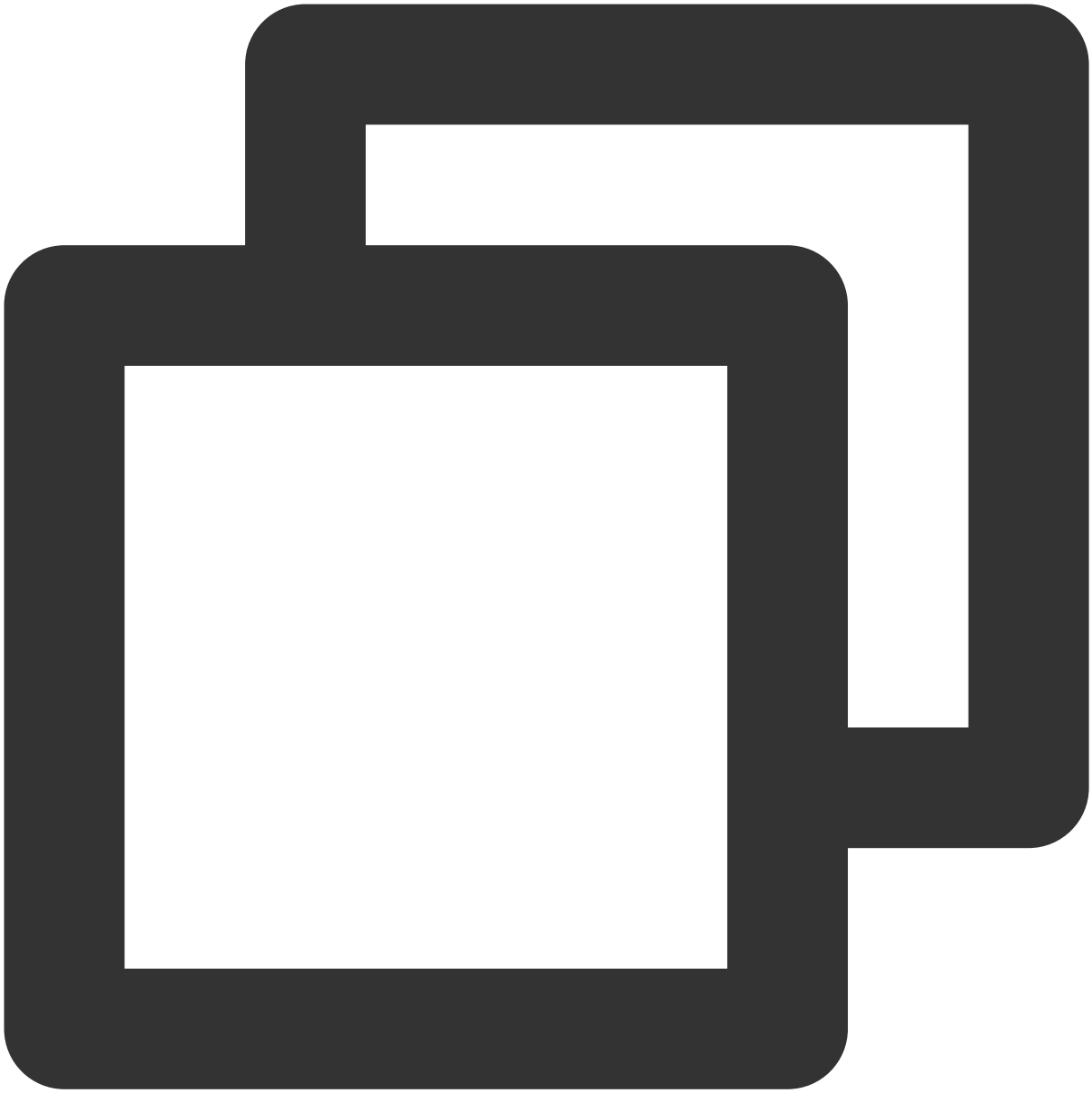
Step 2. Download the Kafka installation file.

Download and decompress the Kafka installation file.

Step 3. Test Kafka APIs.

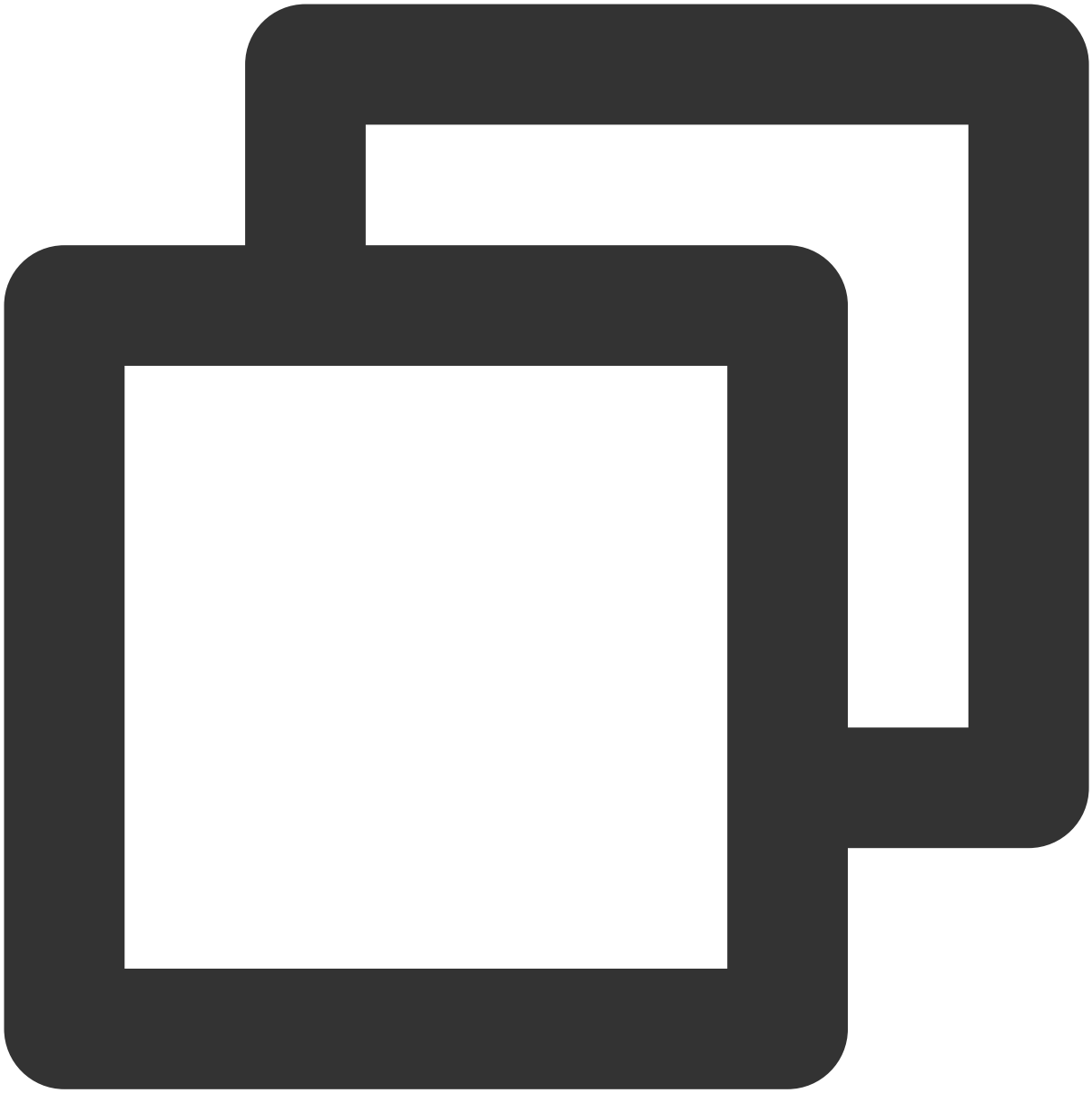
1. Configure an ACL policy locally

1.1 In the `./config` directory of the installation file, add the following content at the end of `producer.properties` and `consumer.properties` .



```
security.protocol=SASL_PLAINTEXT
sasl.mechanism=PLAIN
```

1.2 Create a file named `ckafka_client_jaas.conf` , and add the following content.

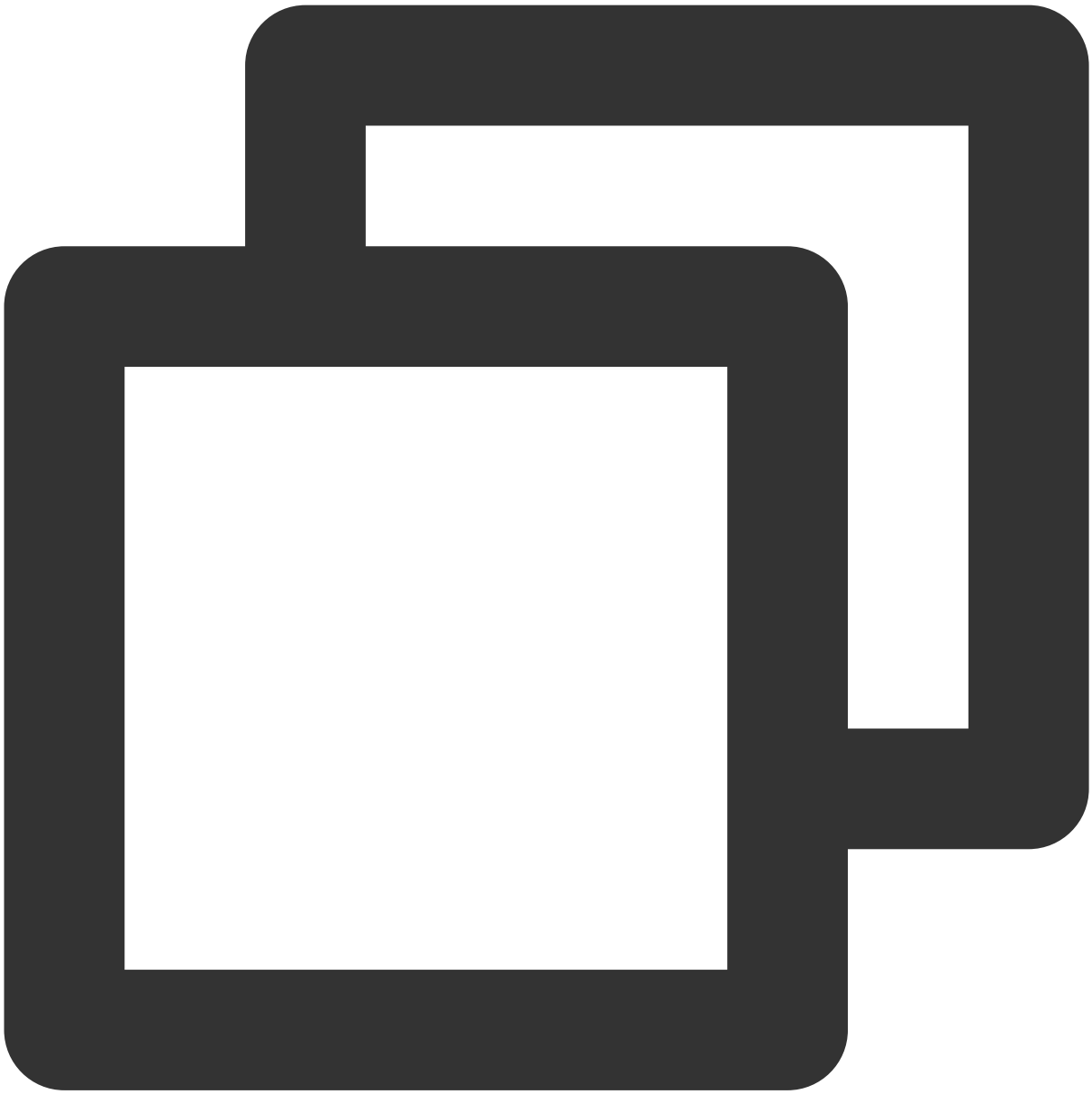


```
KafkaClient {
          org.apache.kafka.common.security.plain.PlainLoginModule required
          username="yourinstance#yourusername"
          password="yourpassword";
};
```

Note:

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

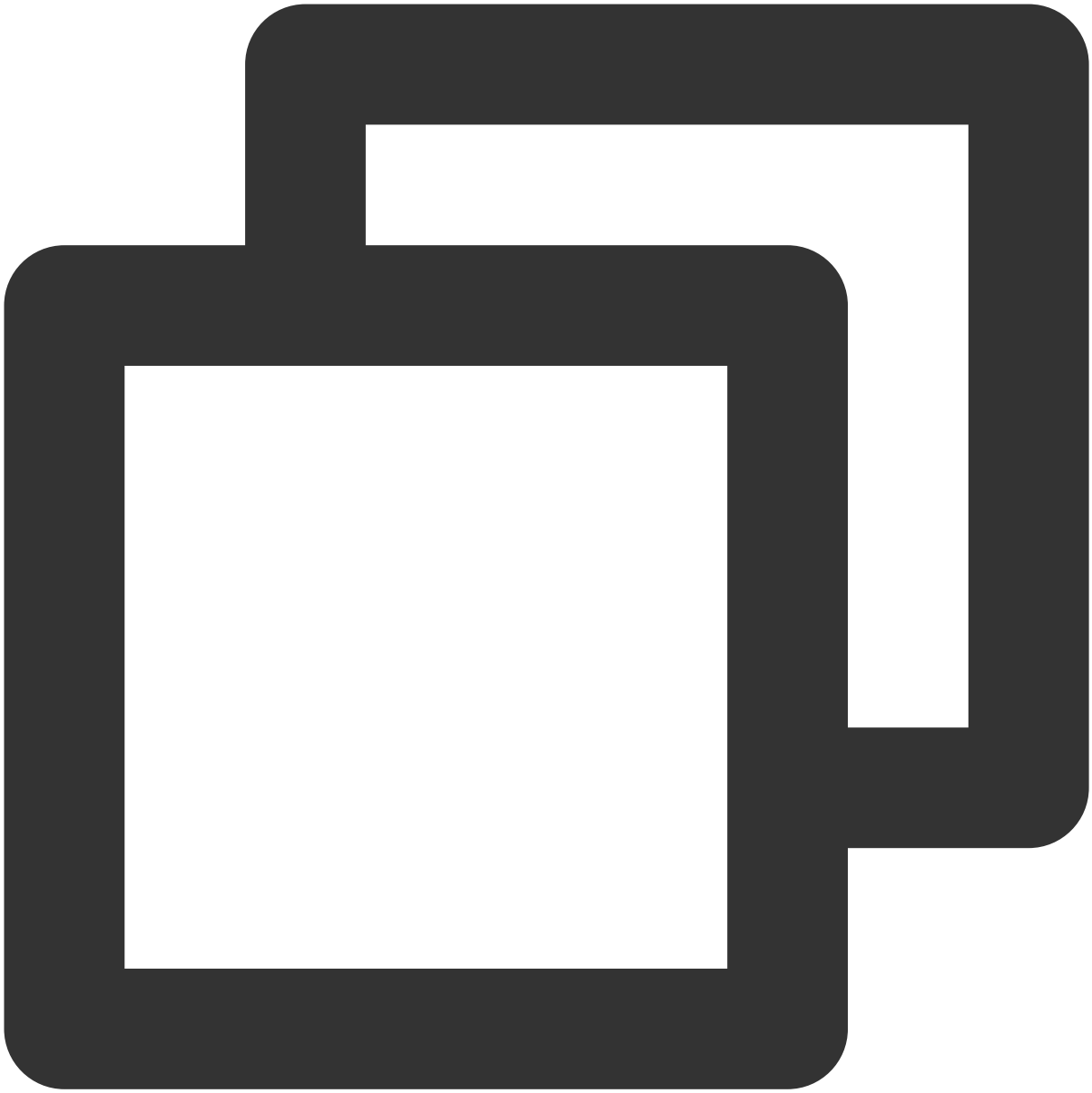
1.3 In the `./bin` directory of the installation file, add the statement of the full path of the JAAS file at the beginning of `kafka-console-producer.sh` and `kafka-console-consumer.sh` .



```
export KAFKA_OPTS="-Djava.security.auth.login.config=****/config/ckafka_client_jaas
```

2. Go to the `./bin` directory, and produce and consume a message via CLI commands.

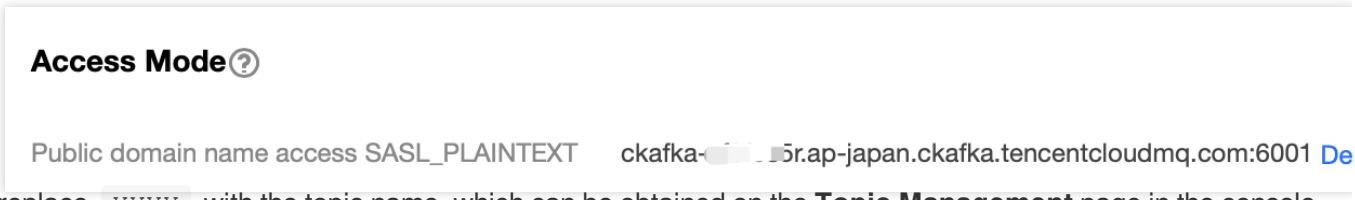
2.1 Open a terminal window to start a consumer.



```
bash kafka-console-consumer.sh --bootstrap-server XXXX:port --topic XXXX --consumer
```

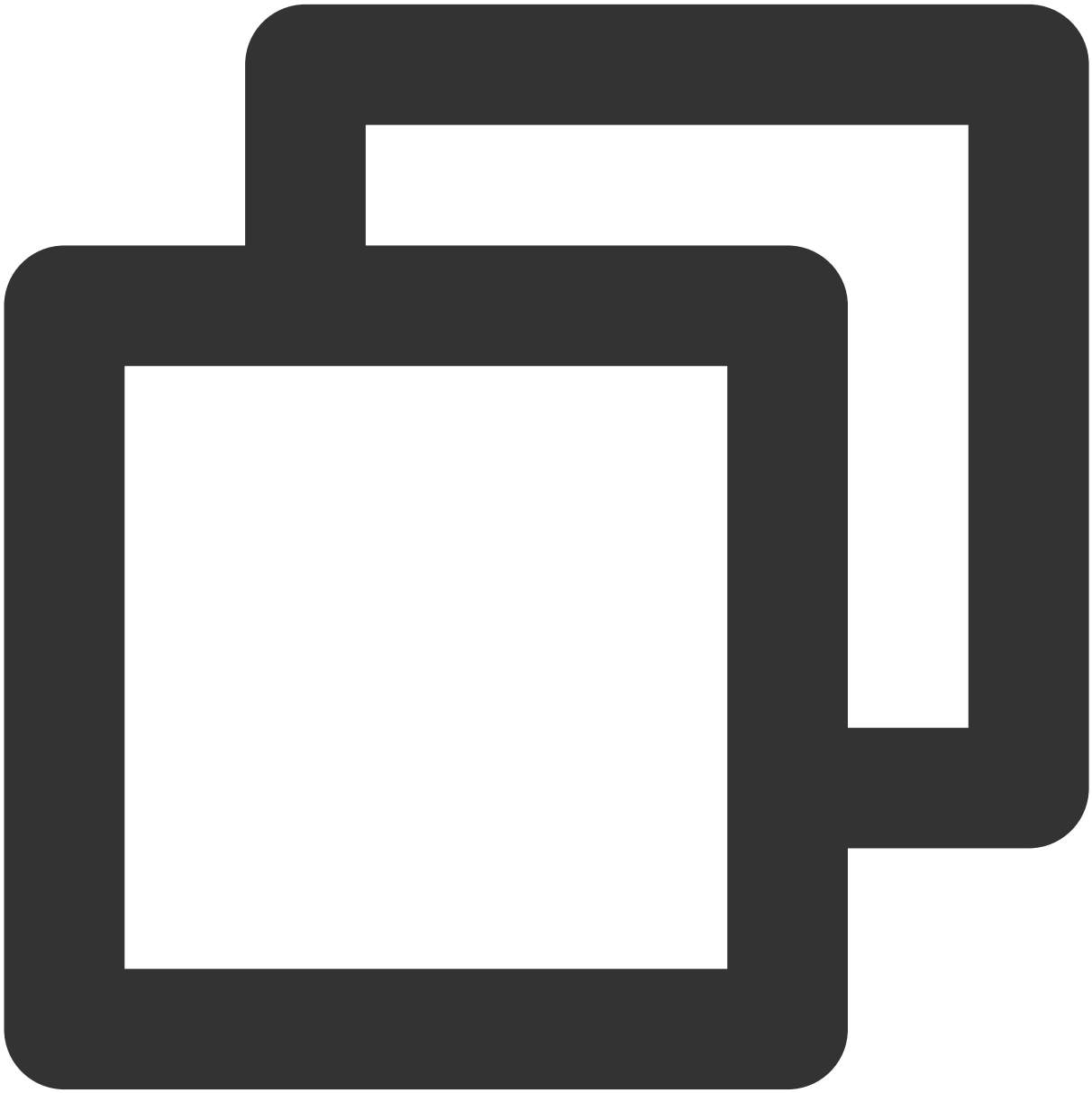
Note:

broker-list: replace `XXXX:port` with the domain name and port for public network access, which can be obtained in the **Access Mode** section on the **Instance Details** page in the console.



topic: replace `XXXX` with the topic name, which can be obtained on the **Topic Management** page in the console.

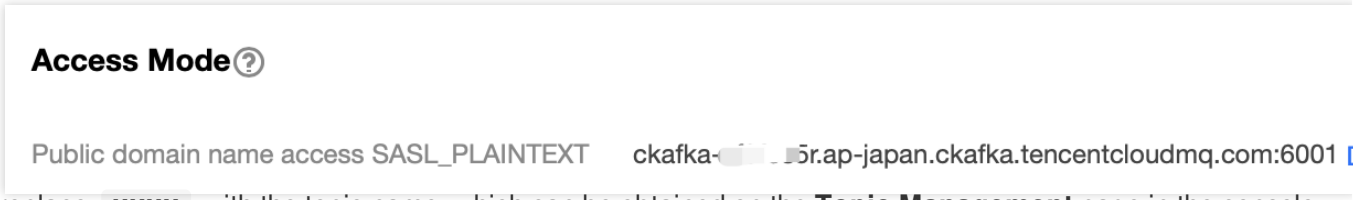
2.2 Open another terminal window to start a producer.



```
bash kafka-console-producer.sh --broker-list XXXX:port --topic XXXX --producer.con
```

Note:

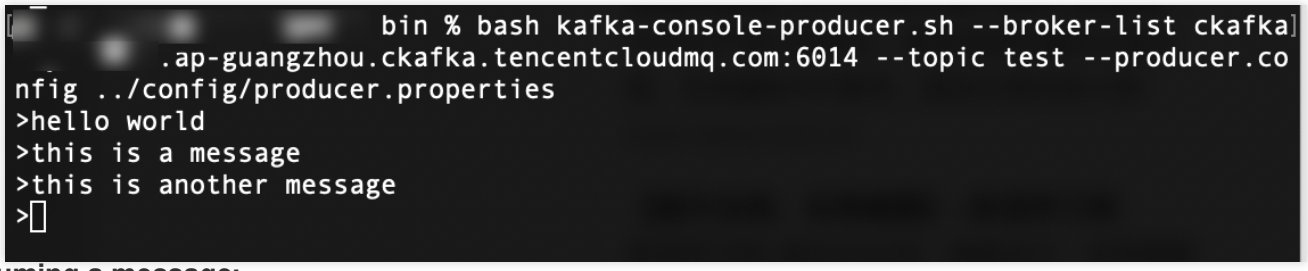
broker-list: replace `XXXX:port` with the domain name and port for public network access, which can be obtained in the **Access Mode** section on the **Instance Details** page in the console.



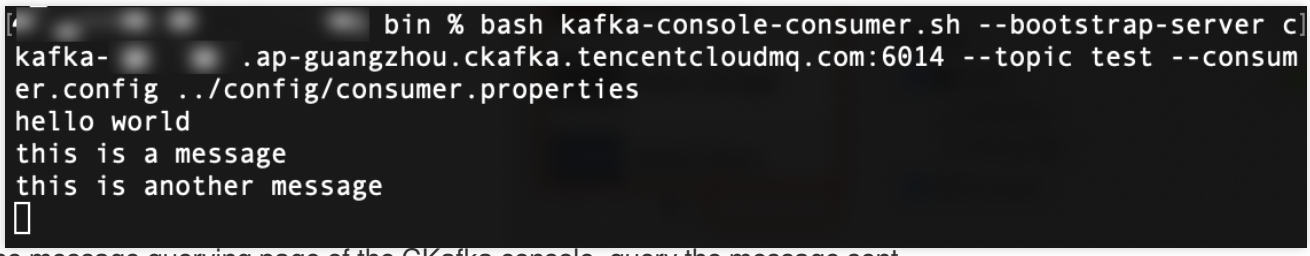
topic: replace `XXXX` with the topic name, which can be obtained on the **Topic Management** page in the console.

Enter the content of the message and press Enter.

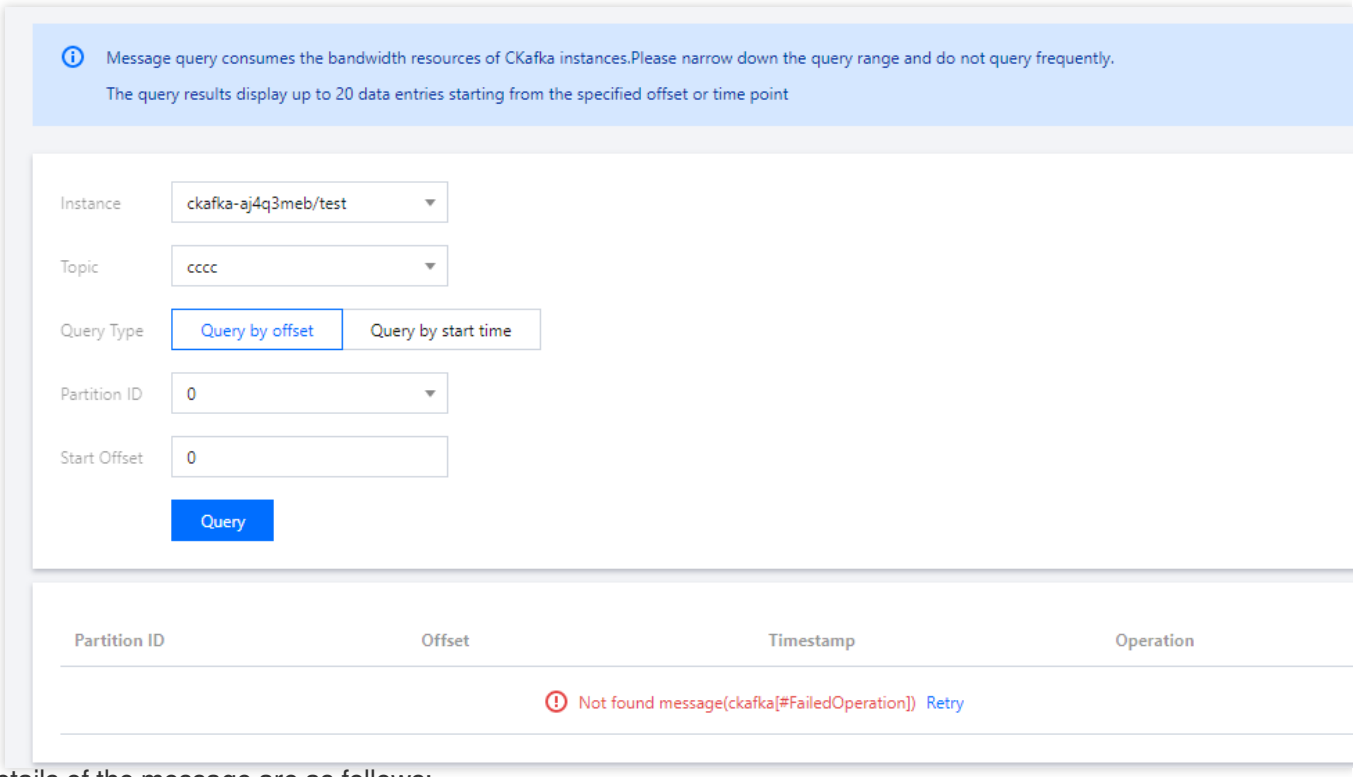
Producing a message:



Consuming a message:



3. In the message querying page of the CKafka console, query the message sent.



The details of the message are as follows: