

消息队列 CKafka 版

SDK 文档

产品文档



腾讯云

【版权声明】

©2013-2024 腾讯云版权所有

本文档著作权归腾讯云单独所有，未经腾讯云事先书面许可，任何主体不得以任何形式复制、修改、抄袭、传播全部或部分本文档内容。

【商标声明】

及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。

【服务声明】

本文档意在向客户介绍腾讯云全部或部分产品、服务的当时的整体概况，部分产品、服务的内容可能有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或默示的承诺或保证。

文档目录

SDK 文档

SDK 概览

Java SDK

VPC 网络接入

VPC 网络 SASL_SCRAM 方式接入

公网 SASL_PLAINTEXT 方式接入

公网 SASL_SSL 方式接入

Python SDK

VPC 网络接入

公网 SASL_PLAINTEXT 方式接入

公网 SASL_SSL 方式接入

Go SDK

VPC 网络接入

公网 SASL_PLAINTEXT 方式接入

PHP SDK

VPC 网络接入

公网 SASL_PLAINTEXT 方式接入

C++ SDK

VPC 网络接入

公网 SASL_PLAINTEXT 方式接入

Node.js SDK

VPC 网络接入

公网 SASL_PLAINTEXT 方式接入

连接器相关 SDK

数据上报 SDK

SDK 文档

SDK 概览

最近更新时间：2024-01-09 15:00:32

消息队列 CKafka 版支持多语言 SDK，客户端可以通过 VPC 网络和公网访问两种方式接入 CKafka 并收发消息。两种接入方式对应的协议说明如下：

网络	VPC 网络	公网域名接入
协议	PLAINTEXT SASL_PLAINTEXT SASL_SSL（专业版支持）	SASL_PLAINTEXT SASL_SSL（专业版支持）

各语言 SDK 的使用方式如下：

SDK 类型	文档
Java SDK	VPC 网络接入 公网 SASL_PLAINTEXT 方式接入 公网 SASL_SSL 方式接入 VPC 网络 SASL_SCRAM 方式接入
Python SDK	VPC 网络接入 公网 SASL_PLAINTEXT 方式接入 公网 SASL_SSL 方式接入
Go SDK	VPC 网络接入 公网 SASL_PLAINTEXT 方式接入
PHP SDK	VPC 网络接入 公网 SASL_PLAINTEXT 方式接入
C++ SDK	VPC 网络接入 公网 SASL_PLAINTEXT 方式接入
Node.js SDK	VPC 网络接入 公网 SASL_PLAINTEXT 方式接入

Java SDK

VPC 网络接入

最近更新时间：2024-05-31 14:40:49

操作场景

该任务以 Java 客户端为例指导您使用 VPC 网络接入消息队列 CKafka 版并收发消息。

前提条件

[安装1.8或以上版本 JDK](#)

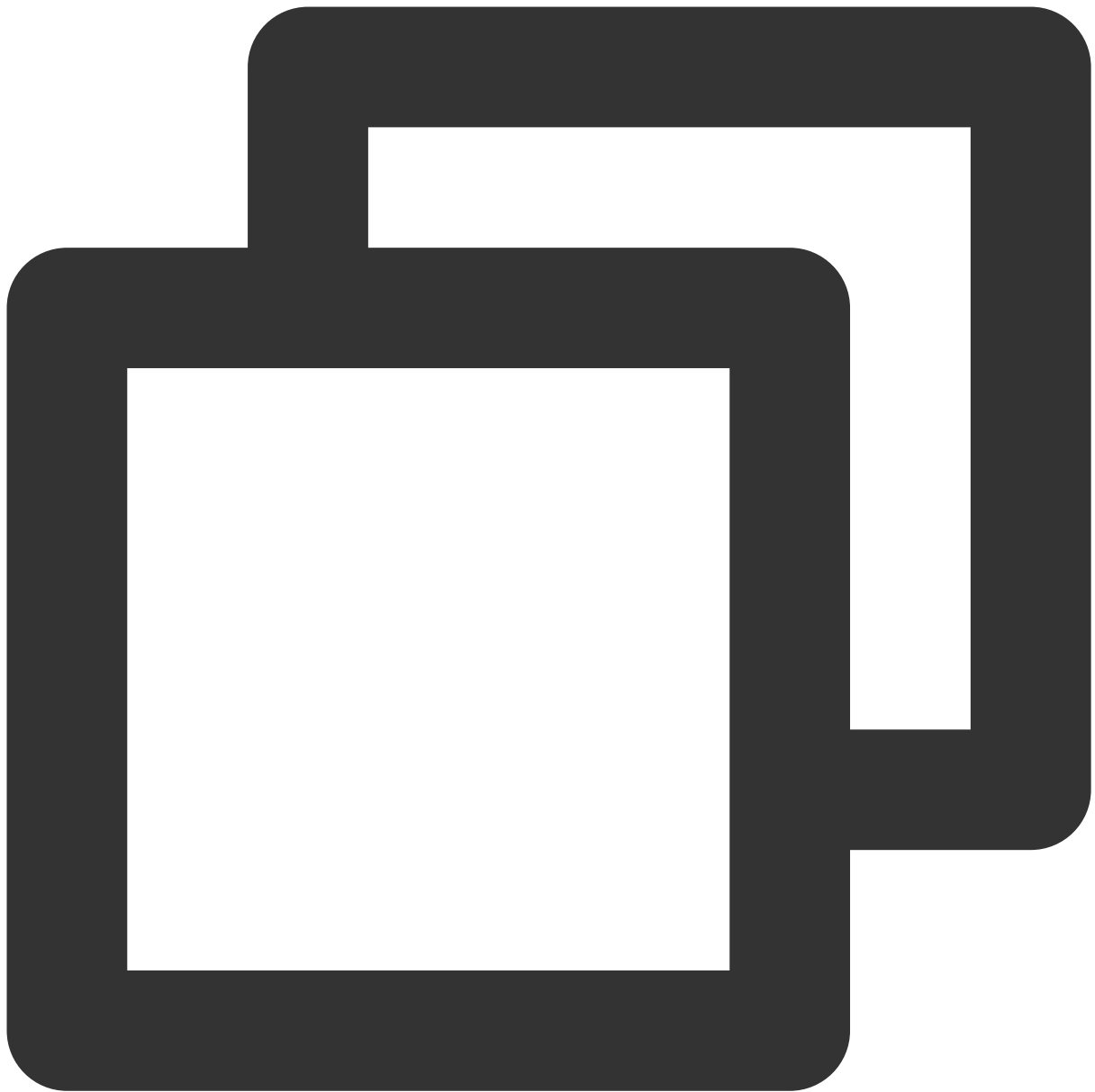
[安装2.5或以上版本 Maven](#)

[下载 Demo](#)

操作步骤

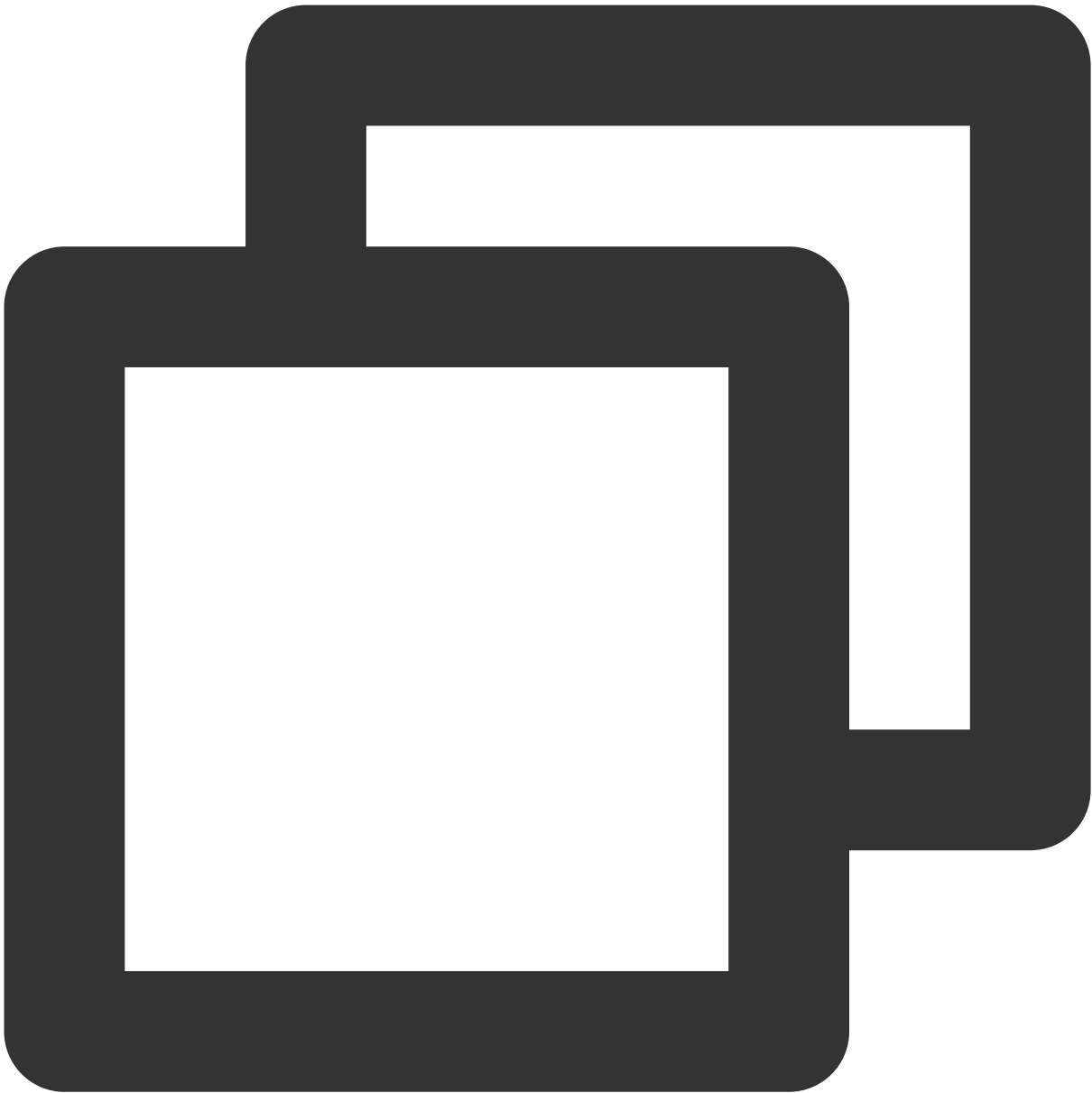
步骤1：准备配置

1. 将下载的 Demo 中的 javakafkademo 上传至 Linux 服务器。
2. 登录 Linux 服务器，进入 javakafkademo 目录，并配置相关参数。
 - 2.1 在 pom.xml 中添加以下依赖。



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

2.2 创建消息队列 CKafka 版配置文件 kafka.properties。

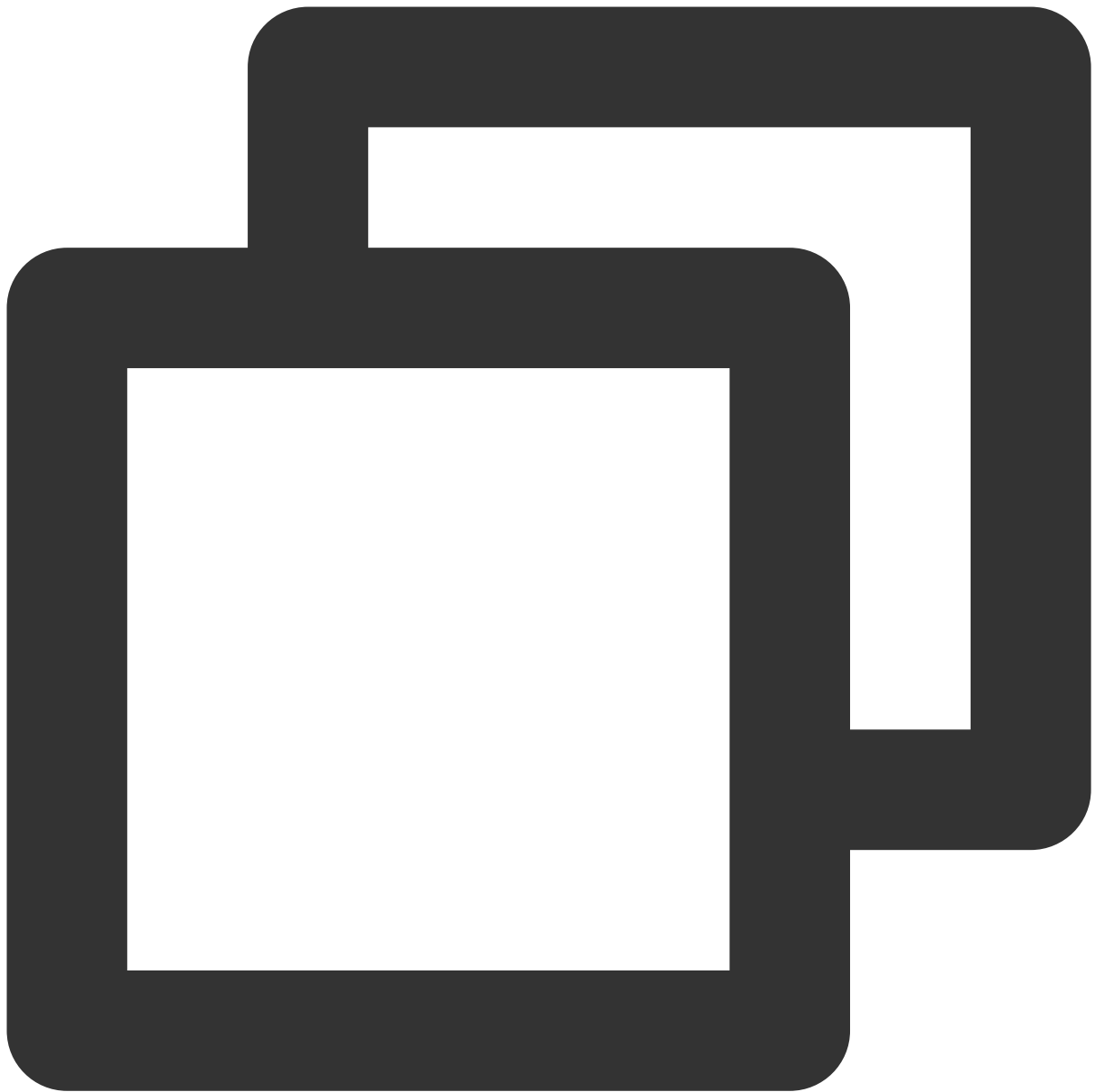


```
## 配置接入网络，在控制台的实例详情页面接入方式模块的网络列复制。  
bootstrap.servers=xx.xx.xx.xx:xxxx  
## 配置Topic，在控制台上topic管理页面复制。  
topic=XXX  
## 配置Consumer Group，您可以自定义设置  
group.id=XXX
```

参数	说明

bootstrap.servers	接入网络，在控制台的实例详情页面接入方式模块的网络列复制。 接入方式? 添加路由策略 <table><tr><th>接入类型</th><th>接入方式</th><th>网络</th><th>操作</th></tr><tr><td>VPC网络</td><td>PLAINTEXT</td><td> vpc-l subnet-5gl6yj86 1.....0092 </td><td> 删除 查看所有IP和端口 </td></tr></table>	接入类型	接入方式	网络	操作	VPC网络	PLAINTEXT	vpc-l subnet-5gl6yj86 1.....0092	删除 查看所有IP和端口												
接入类型	接入方式	网络	操作																		
VPC网络	PLAINTEXT	vpc-l subnet-5gl6yj86 1.....0092	删除 查看所有IP和端口																		
topic	Topic 名称，您可以在控制台上 topic 管理页面复制。 ckafka-p 基本信息topic管理Consumer Group监控事件中心HTTP接入ACL策略管理智能运维集群备份 新建(24/450) 请输入TopicId或名称 <table><tr><th>ID/名称</th><th>监控</th><th>分区数(个)</th><th>副本数(个)</th><th>标签</th><th>备注</th><th>创建时间</th><th>消息保留时间</th><th>状态</th><th>操作</th></tr><tr><td> topic-j cocs </td><td></td><td>3</td><td>2</td><td></td><td></td><td>2023-03-25 23:03:00</td><td>2 天</td><td>正常</td><td> 编辑 删除 更多 </td></tr></table>	ID/名称	监控	分区数(个)	副本数(个)	标签	备注	创建时间	消息保留时间	状态	操作	topic-j cocs		3	2			2023-03-25 23:03:00	2 天	正常	编辑 删除 更多
ID/名称	监控	分区数(个)	副本数(个)	标签	备注	创建时间	消息保留时间	状态	操作												
topic-j cocs		3	2			2023-03-25 23:03:00	2 天	正常	编辑 删除 更多												
group.id	您可以自定义设置，demo 运行成功后可以在 Consumer Group 页面看到该消费者。																				

3. 创建配置文件加载程序 CKafkaConfigurer.java。

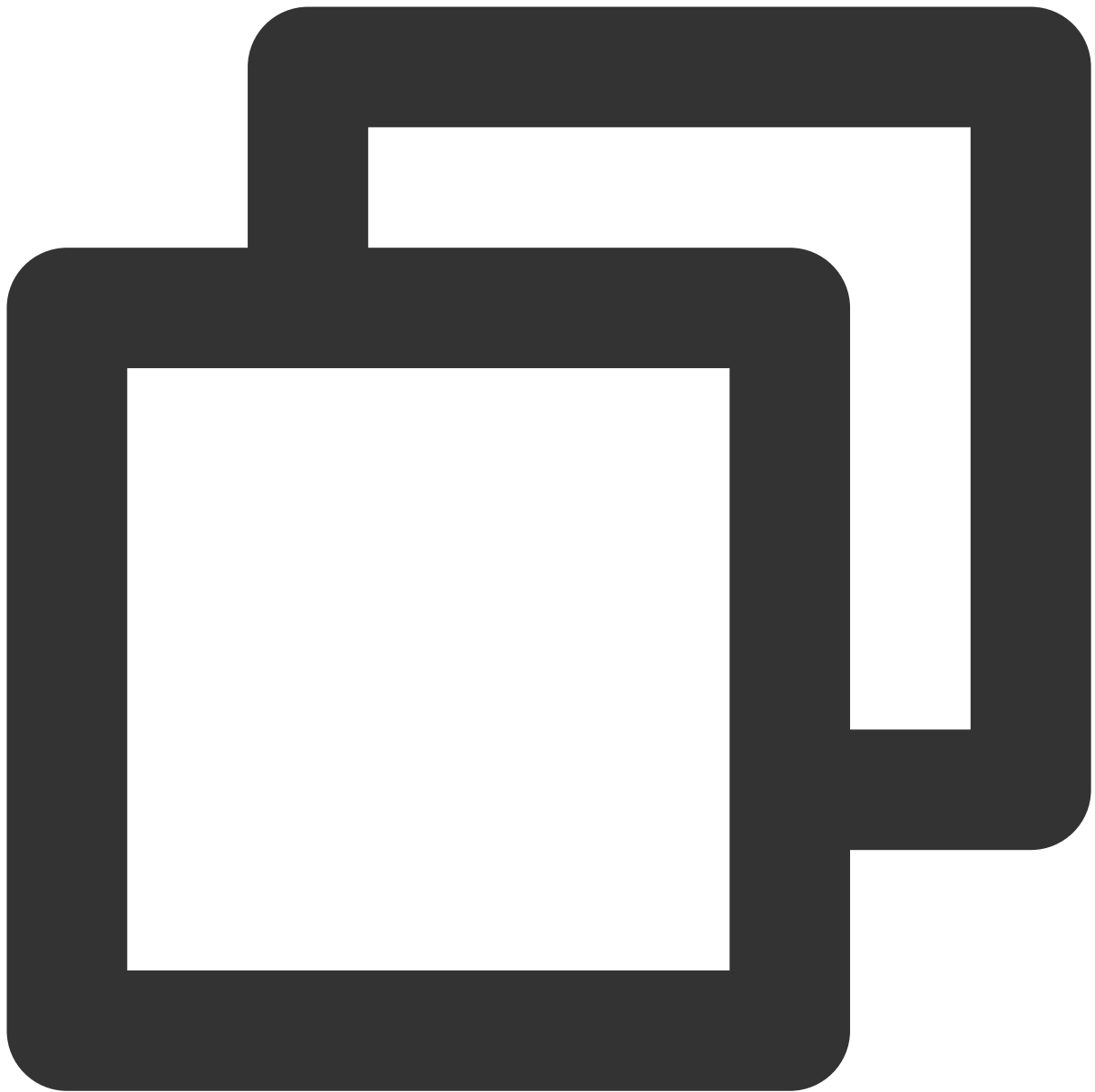


```
public class CKafkaConfigurer {  
  
    private static Properties properties;  
  
    public synchronized static Properties getCKafkaProperties() {  
        if (null != properties) {  
            return properties;  
        }  
        //获取配置文件kafka.properties的内容。  
        Properties kafkaProperties = new Properties();  
        try {
```

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

步骤2：发送消息

1. 编写生产消息程序 CKafkaProducerDemo.java。



```
public class CKafkaProducerDemo {  
  
    public static void main(String args[]) {  
        //加载kafka.properties。  
        Properties kafkaProperties = CKafkaConfigurer.getCKafkaProperties();  
  
        Properties properties = new Properties();  
        //设置接入点， 请通过控制台获取对应Topic的接入点。  
        properties.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, kafkaProperties.get  
  
        //消息队列Kafka版消息的序列化方式， 此处demo 使用的是StringSerializer。
```

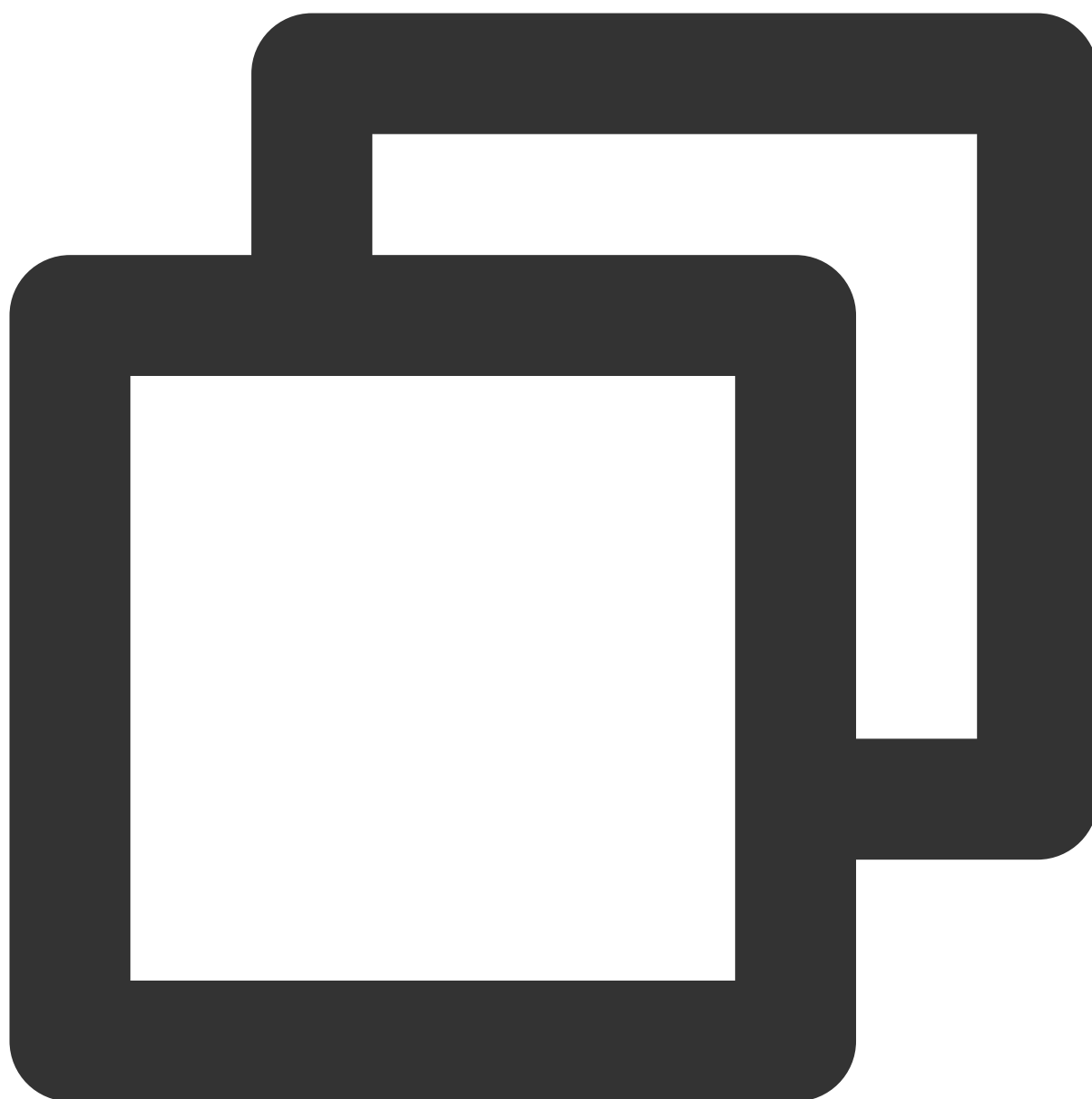
```
properties.put (ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
    "org.apache.kafka.common.serialization.StringSerializer");
properties.put (ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
    "org.apache.kafka.common.serialization.StringSerializer");
//请求的最长等待时间。
properties.put (ProducerConfig.MAX_BLOCK_MS_CONFIG, 30 * 1000);
//设置客户端内部重试次数。
properties.put (ProducerConfig.RETRIES_CONFIG, 5);
//设置客户端内部重试间隔。
properties.put (ProducerConfig.RECONNECT_BACKOFF_MS_CONFIG, 3000);
//构造Producer对象。
KafkaProducer<String, String> producer = new KafkaProducer<>(properties);

//构造一个消息队列Kafka版消息。
String topic = kafkaProperties.getProperty("topic"); //消息所属的Topic, 请在控
String value = "this is ckafka msg value"; //消息的内容。

try {
    //批量获取Future对象可以加快速度, 但注意, 批量不要太大。
    List<Future<RecordMetadata>> futureList = new ArrayList<>(128);
    for (int i = 0; i < 10; i++) {
        //发送消息, 并获得一个Future对象。
        ProducerRecord<String, String> kafkaMsg = new ProducerRecord<>(topic,
            value + ": " + i);
        Future<RecordMetadata> metadataFuture = producer.send(kafkaMsg);
        futureList.add(metadataFuture);
    }
    producer.flush();
    for (Future<RecordMetadata> future : futureList) {
        //同步获得Future对象的结果。
        RecordMetadata recordMetadata = future.get();
        System.out.println("produce send ok: " + recordMetadata.toString())
    }
} catch (Exception e) {
    //客户端内部重试之后, 仍然发送失败, 业务要应对此类错误。
    System.out.println("error occurred");
}
}
```

2. 编译并运行 CKafkaProducerDemo.java 发送消息。

3. 运行结果。



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

4. 在 [CKafka 控制台](#) 的 **topic 管理** 页面，选择对应的 Topic，单击**更多 > 消息查询**，查看刚刚发送的消息。

消息查询

消息查询会占用CKafka实例的带宽资源，建议您尽量缩小查询范围查询，不要频繁操作。

消息查询最多展示最近的20条消息。

实例

Topic

查询类型

分区ID

时间

按位点查询

按时间查询

0

2017-22:54

查询

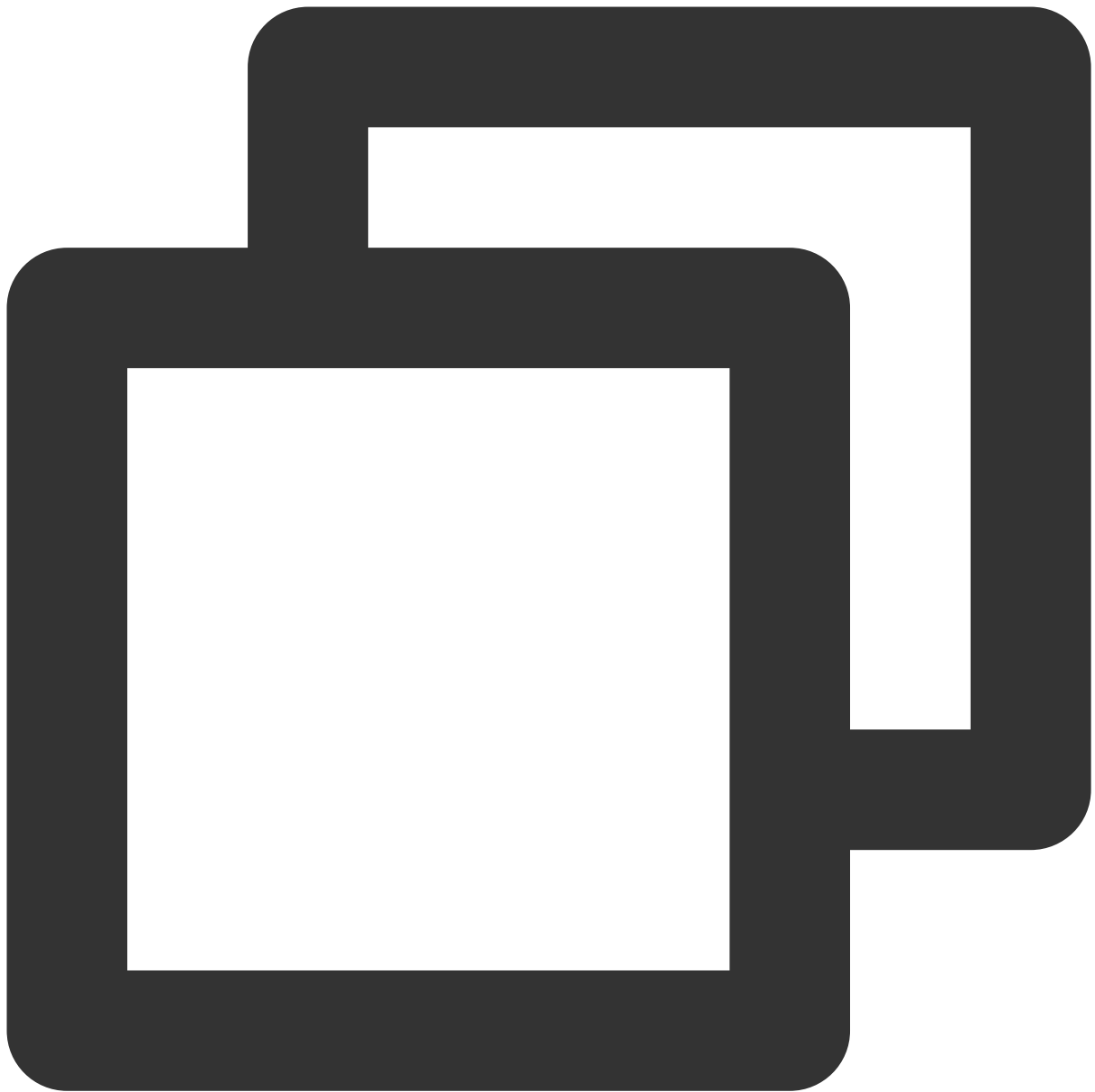
分区ID	位点	时间戳
0	628	2017-25:31
0	629	2017-25:31

步骤3：消费消息

- 1. 创建 Consumer 订阅消息程序 CKafkaConsumerDemo.java。

版权所有：腾讯云计算（北京）有限责任公司

第14 共253页



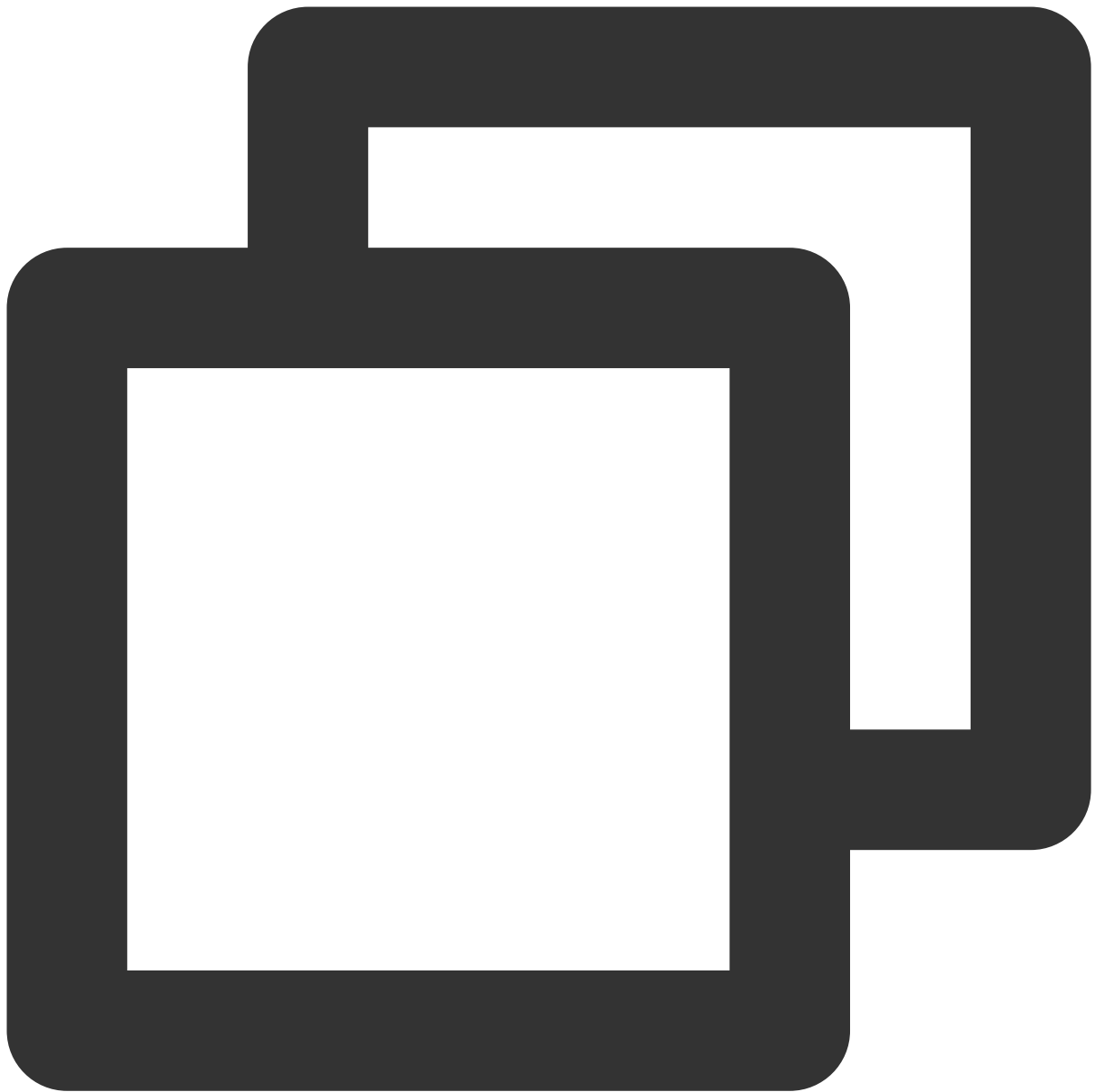
```
public class CKafkaConsumerDemo {  
  
    public static void main(String args[]) {  
        //加载kafka.properties。  
        Properties kafkaProperties = CKafkaConfigurer.getCKafkaProperties();  
  
        Properties props = new Properties();  
        //设置接入点， 请通过控制台获取对应Topic的接入点。  
        props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, kafkaProperties.getPrope  
        //两次Poll之间的最大允许间隔。  
        //消费者超过该值没有返回心跳，服务端判断消费者处于非存活状态，服务端将消费者从Consumer G
```

```
props.put (ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG, 30000);
//每次Poll的最大数量。
//注意该值不要改得太大，如果Poll太多数据，而不能在下次Poll之前消费完，则会触发一次负载均衡
props.put (ConsumerConfig.MAX_POLL_RECORDS_CONFIG, 30);
//消息的反序列化方式。
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");
//属于同一个组的消费实例，会负载消费消息。
props.put (ConsumerConfig.GROUP_ID_CONFIG, kafkaProperties.getProperty("group.id"));
//构造消费对象，也即生成一个消费实例。
KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props);
//设置消费组订阅的Topic，可以订阅多个。
//如果GROUP_ID_CONFIG是一样，则订阅的Topic也建议设置成一样。
List<String> subscribedTopics = new ArrayList<>();
//如果需要订阅多个Topic，则在这里添加进去即可。
//每个Topic需要先在控制台进行创建。
String topicStr = kafkaProperties.getProperty("topic");
String[] topics = topicStr.split(",");
for (String topic : topics) {
    subscribedTopics.add(topic.trim());
}
consumer.subscribe(subscribedTopics);

//循环消费消息。
while (true) {
    try {
        ConsumerRecords<String, String> records = consumer.poll(1000);
        //必须在下次Poll之前消费完这些数据，且总耗时不得超过SESSION_TIMEOUT_MS_CONFIG
        //建议开一个单独的线程池来消费消息，然后异步返回结果。
        for (ConsumerRecord<String, String> record : records) {
            System.out.println(
                String.format("Consume partition:%d offset:%d", record.partition(), record.offset()));
        }
    } catch (Exception e) {
        System.out.println("consumer error!");
    }
}
}
```

2. 编译并运行 CKafkaConsumerDemo.java 消费消息。

3. 运行结果。



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

4. 在 [CKafka 控制台](#) 的 **Consumer Group** 页面，选择对应的消费组名称，在主题名称输入 Topic 名称，单击**查询详情**，查看消费详情。



VPC 网络 SASL_SCRAM 方式接入

最近更新时间：2024-05-31 14:40:49

操作背景

该任务以 Java 客户端为例指导您在 VPC 网络环境下，使用 SASL_SCRAM 方式接入消息队列 CKafka 版并收发消息。

说明：

SASL_SCRAM 接入方式仅北京地域且2.4.1版本实例支持，其他地域和存量实例需要 [提交工单](#) 申请。

前提条件

[安装 1.8 或以上版本 JDK](#)

[安装 2.5 或以上版本 Maven](#)

[配置 ACL 策略](#)

[下载 Demo](#)

操作步骤

步骤1：控制台配置

1. 创建接入点。

1.1 在 [实例列表](#) 页面，单击目标实例 ID，进入实例详情页。

1.2 在 **基本信息 > 接入方式** 中，单击**添加路由策略**。在打开窗口中选择：

路由类型：公网域名接入。

接入方式：SASL_SCRAM。

添加路由策略

路由类型

VPC网络

接入方式

SASL_SCRAM

网络

vpc-

请选择

IP

选填,请输入IP

提交

关闭

如果现有的网络不合适,您可以去控制台[新建私有网络](#)或[新建子网](#)

如果没有指定IP,系统会自动分配

可用区内无有效子网,请在该可用区下新建子网

2. 创建角色。

在 **ACL 策略管理**下的**用户管理**页面新建角色，设置密码。

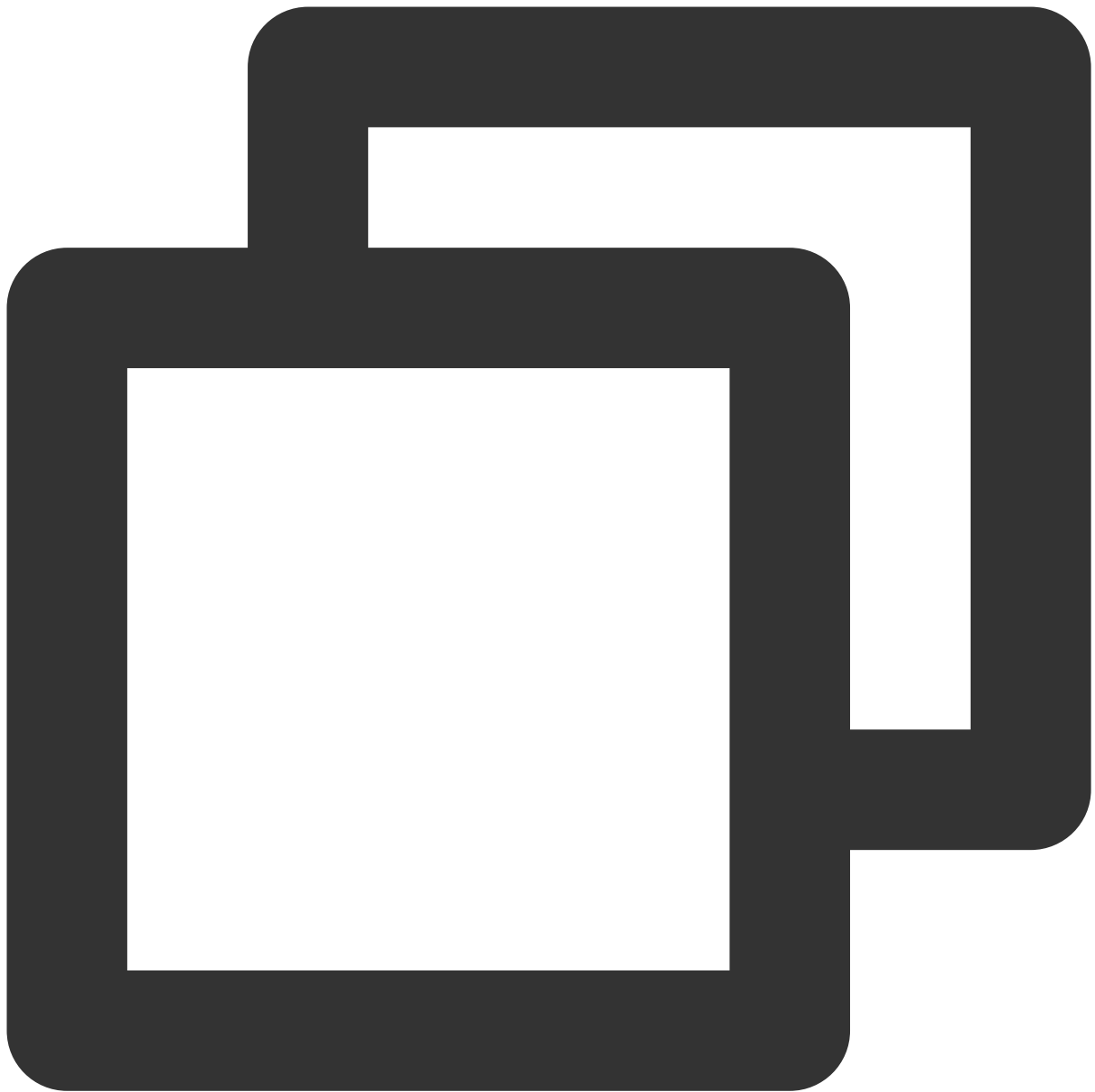
ckafka-		
基本信息 topic管理 Consumer Group 监控 事件中心 HTTP接入 ACL策略管理 智能运维 专业版		
策略列表 用户管理		
创建		
用户名	创建/更新时间	操作
ckafka- user1	2022-11-03 16:05:47 2022-11-03 16:05:47	修改密码

3. 创建 Topic。

在控制台 **Topic 管理**页面新建 Topic（参见 [创建 Topic](#)）。

步骤2：添加配置文件

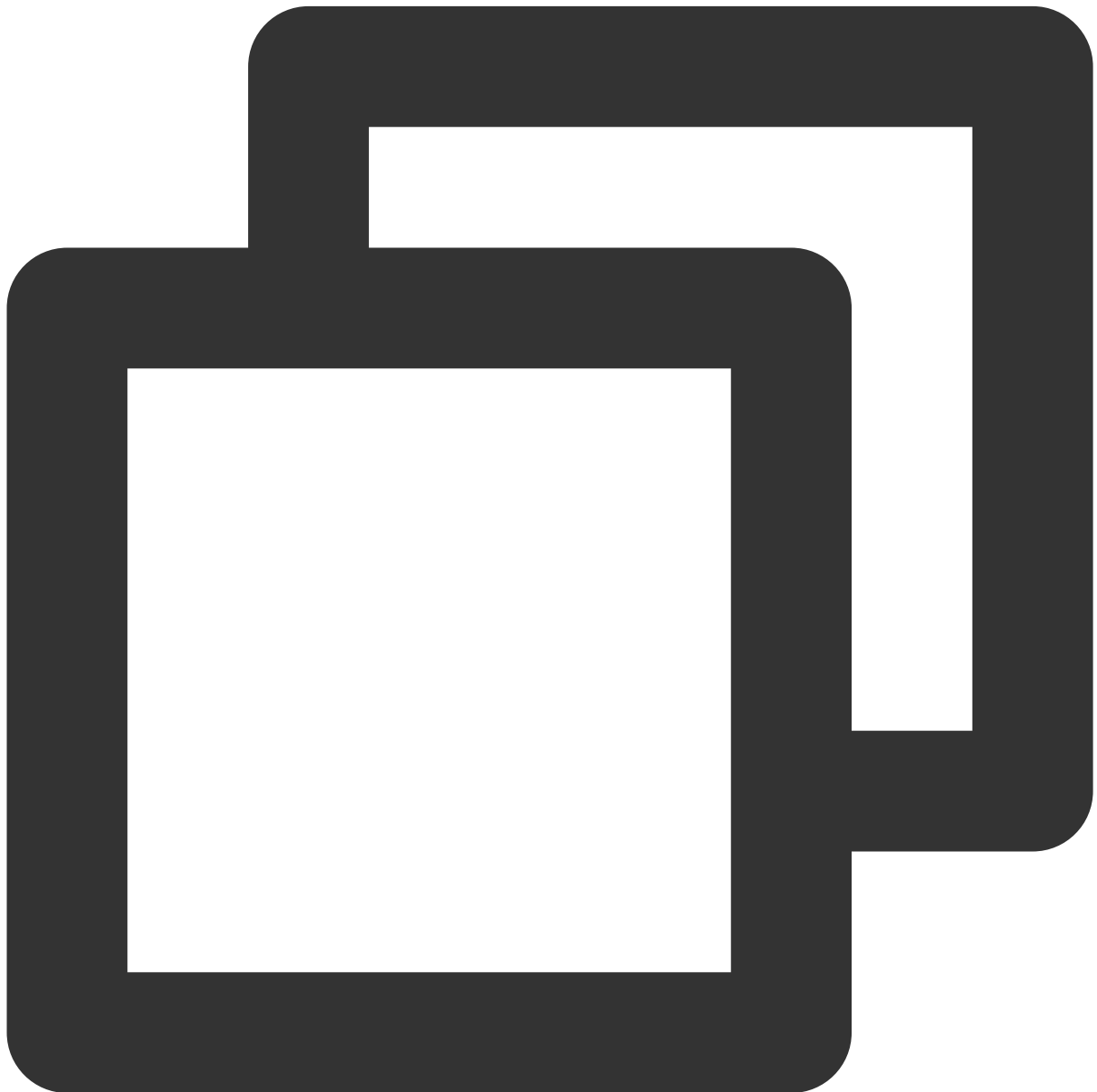
1. 在 pom.xml 中添加以下依赖。



```
<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>
</dependencies>
```

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

2. 创建 JAAS 配置文件 `ckafka_client_jaas.conf`，使用 **ACL 策略管理**下的**用户管理**界面创建的用户进行修改。



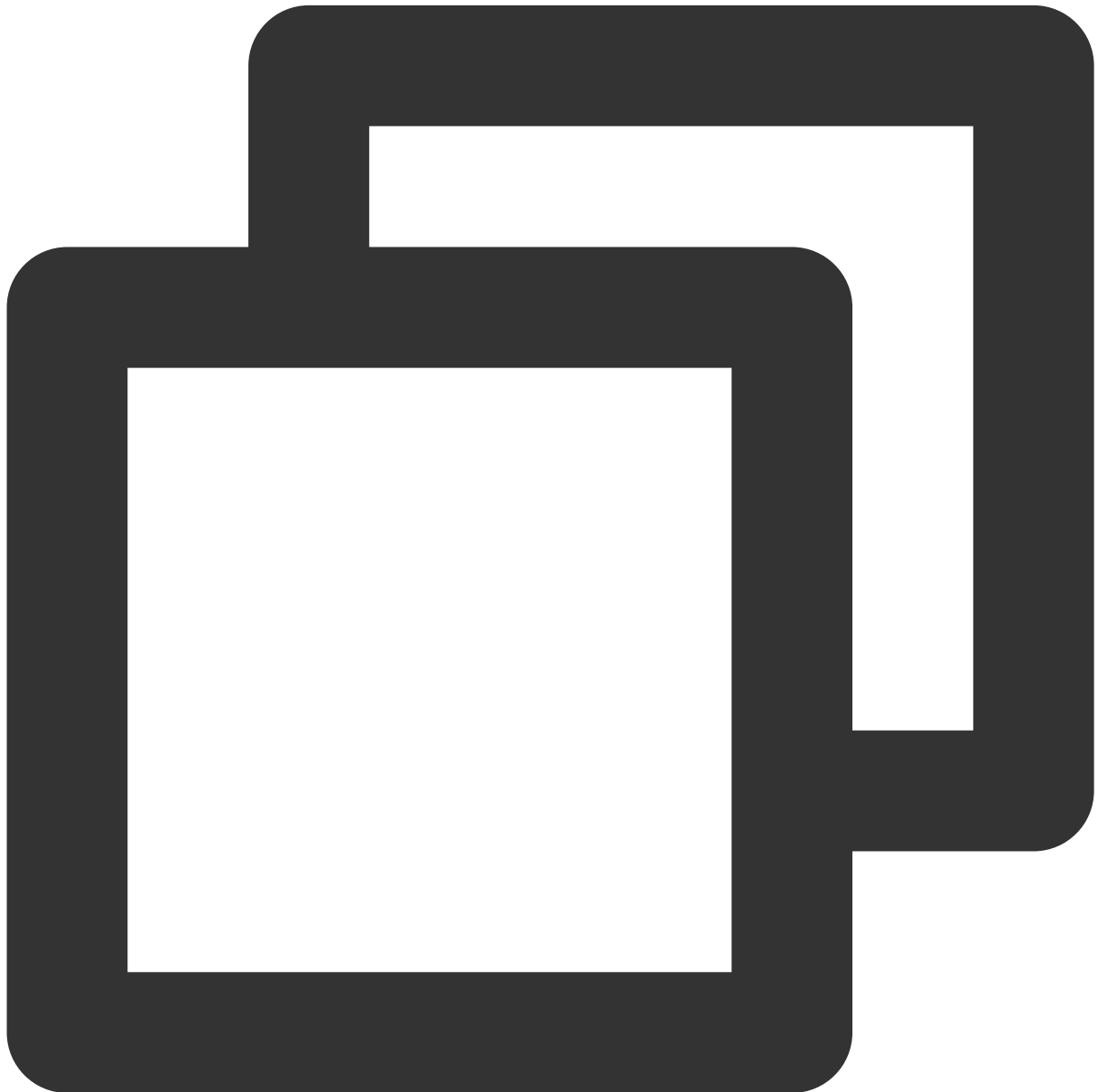
```
KafkaClient {
```

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

说明：

username 是 实例 ID + # + 配置的用户名，password 是配置的用户密码。

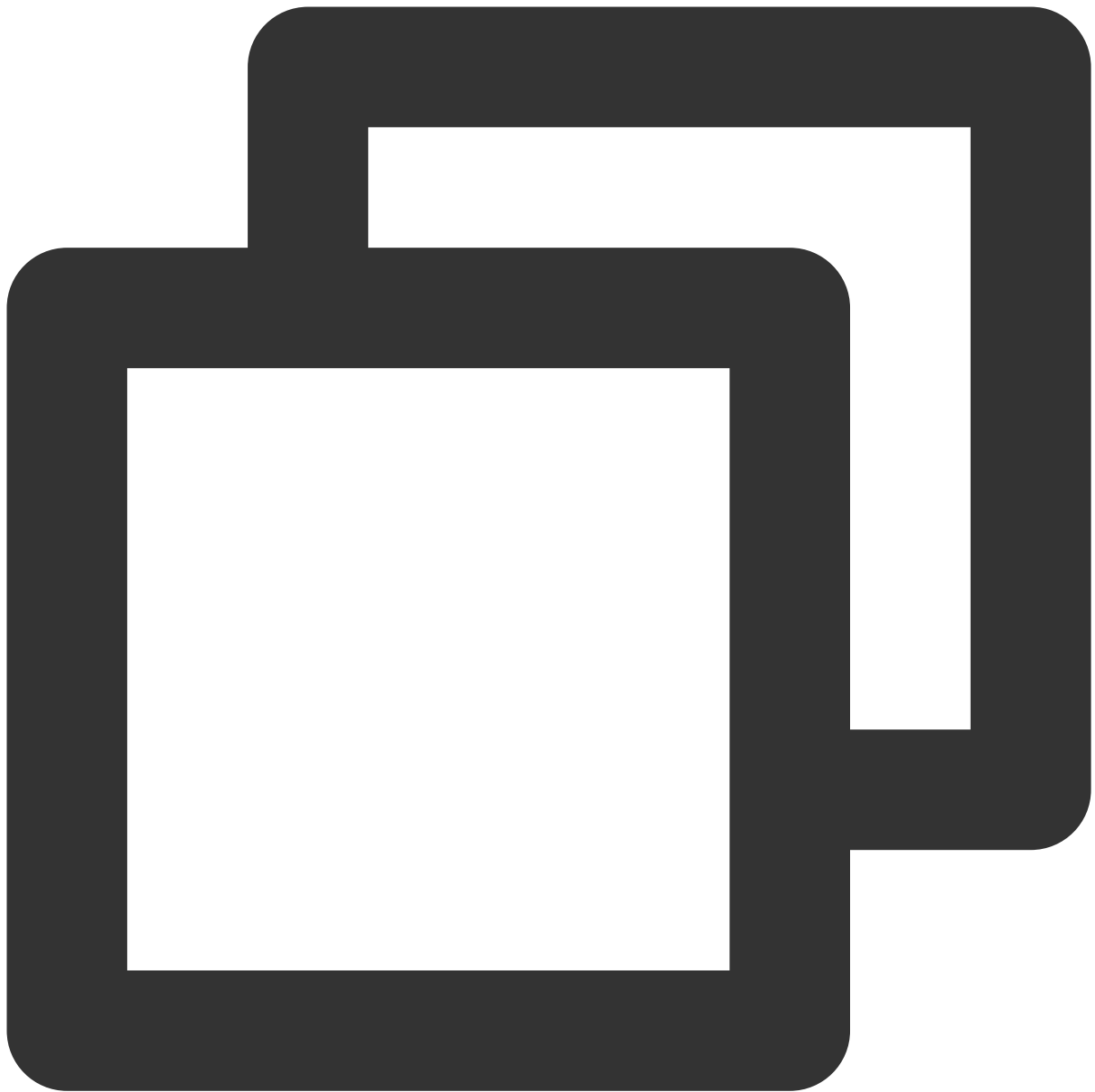
3. 创建消息队列 CKafka 版配置文件 kafka.properties。



```
## 配置接入网络，在控制台的实例详情页面接入方式模块的网络列复制。
bootstrap.servers=xx.xx.xx.xx:xxxx
```

## 配置 Topic，在控制台上 topic 管理页面复制。 topic=XXX ## 配置 consumer group，您可以自定义设置 group.id=XXX ## SASL 配置 java.security.auth.login.config.plain=/xxxx/ckafka_client_jaas.conf																			
参数	说明																		
bootstrap.servers	接入网络，在控制台的实例详情页面接入方式模块的网络列复制。 接入方式? <table><tr><th>接入类型</th><th>接入方式</th><th>网络</th><th>操作</th></tr><tr><td></td><td></td><td></td><td></td></tr><tr><td>VPC网络</td><td>SASL_SCRAM</td><td>vpc-r5sbavzp 10.0.0.50:9092 </td><td>删除</td></tr></table>	接入类型	接入方式	网络	操作					VPC网络	SASL_SCRAM	vpc-r5sbavzp 10.0.0.50:9092 	删除						
接入类型	接入方式	网络	操作																
																			
VPC网络	SASL_SCRAM	vpc-r5sbavzp 10.0.0.50:9092 	删除																
topic	Topic 名称，您可以在控制台上 topic 管理页面复制。 ckafka-12345678901234567890 基本信息topic管理Consumer Group监控事件中心HTTP接入ACL策略管理智能运维专业版集群备份专业版 新建(24/450) 请输入Topic <table><tr><th>ID/名称</th><th>监控</th><th>分区数(个)</th><th>副本数(个)</th><th>标签</th><th>备注</th><th>创建时间</th><th>消息保留时间</th><th>状态</th></tr><tr><td>topic-j cocos </td><td></td><td>3</td><td>2</td><td></td><td></td><td>2023-03-25 23:03:00</td><td>2 天</td><td>正常</td></tr></table>	ID/名称	监控	分区数(个)	副本数(个)	标签	备注	创建时间	消息保留时间	状态	topic-j cocos 		3	2			2023-03-25 23:03:00	2 天	正常
ID/名称	监控	分区数(个)	副本数(个)	标签	备注	创建时间	消息保留时间	状态											
topic-j cocos 		3	2			2023-03-25 23:03:00	2 天	正常											
group.id	您可以自定义设置，Demo 运行成功后可以在 Consumer Group 页面看到																		
java.security.auth.login.config.plain	填写 JAAS 配置文件 <code>ckafka_client_jaas.conf</code> 的路径。																		

4. 创建配置文件加载程序 CKafkaConfigurer.java。



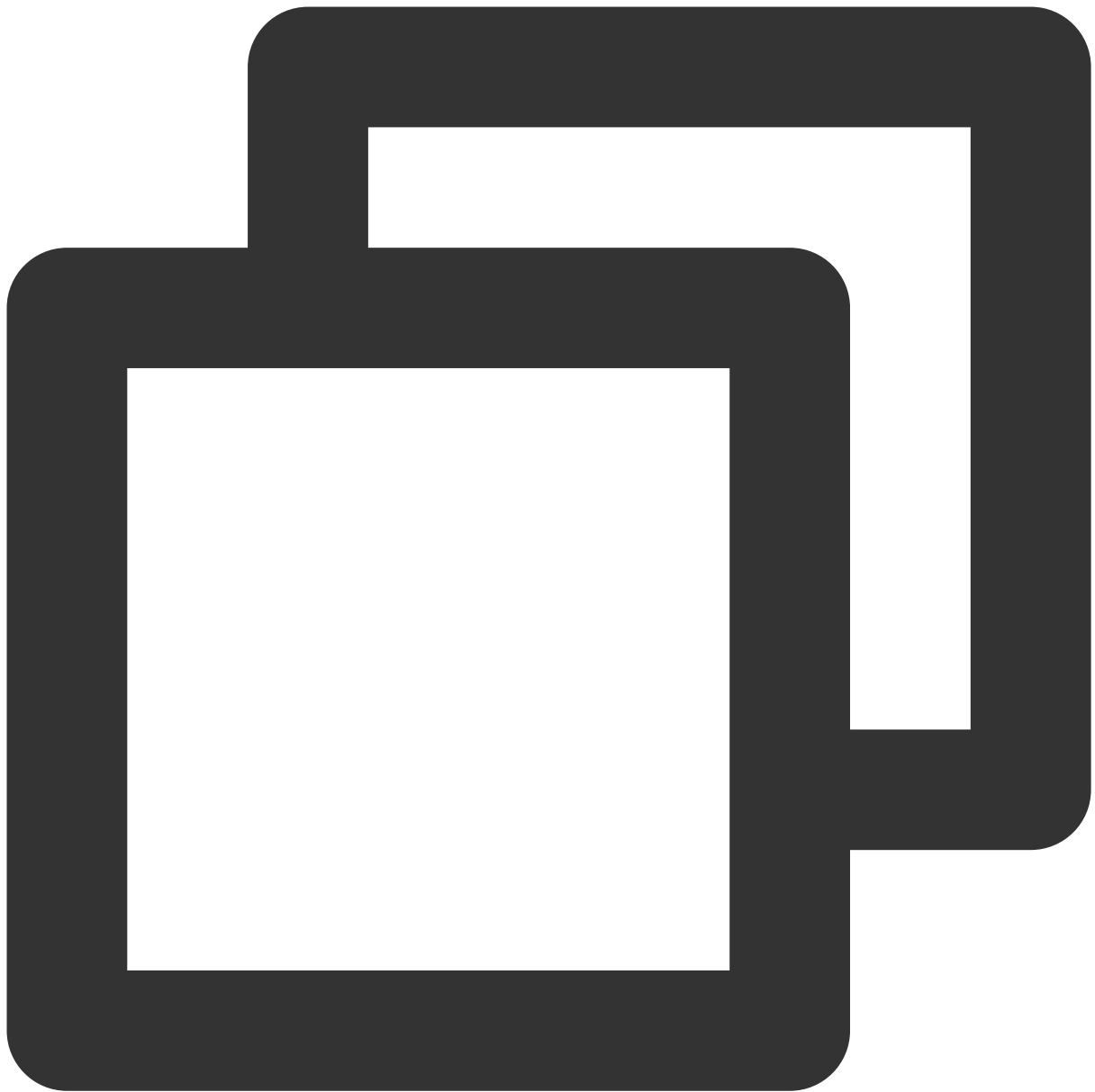
```
public class CKafkaConfigurer {  
  
    private static Properties properties;  
  
    public static void configureSaslPlain() {  
        //如果用 -D 或者其它方式设置过，这里不再设置。  
        if (null == System.getProperty("java.security.auth.login.config")) {  
            //请注意将 xxx 修改为自己的路径。  
            System.setProperty("java.security.auth.login.config",  
                               getCKafkaProperties().getProperty("java.security.auth.login.conf  
        }  
    }  
}
```

```
}

public synchronized static Properties getCKafkaProperties() {
    if (null != properties) {
        return properties;
    }
    //获取配置文件 kafka.properties 的内容。
    Properties kafkaProperties = new Properties();
    try {
        kafkaProperties.load(CKafkaProducerDemo.class.getClassLoader().getResourceAsStream("kafka.properties"));
    } catch (Exception e) {
        System.out.println("getCKafkaProperties error");
    }
    properties = kafkaProperties;
    return kafkaProperties;
}
}
```

步骤3：发送消息

1. 创建发送消息程序 `KafkaSaslProducerDemo.java`。



```
public class KafkaSaslProducerDemo {

    public static void main(String[] args) {
        //设置 JAAS 配置文件的路径。
        CKafkaConfigurer.configureSaslPlain();

        //加载 kafka.properties。
        Properties kafkaProperties = CKafkaConfigurer.getCKafkaProperties();

        Properties props = new Properties();
        //设置接入点，请通过控制台获取对应 Topic 的接入点。
    }
}
```

```
props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
    kafkaProperties.getProperty("bootstrap.servers"));

//
// SASL_SCRAM 接入
//
props.put(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG, "SASL_PLAINTEXT");
// SASL 采用 Plain 方式。
props.put(SaslConfigs.SASL_MECHANISM, "SCRAM-SHA-256");

//消息队列 Kafka 版消息的序列化方式。
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");
//请求的最长等待时间。
props.put(ProducerConfig.MAX_BLOCK_MS_CONFIG, 30 * 1000);
//设置客户端内部重试次数。
props.put(ProducerConfig.RETRIES_CONFIG, 5);
//设置客户端内部重试间隔。
props.put(ProducerConfig.RECONNECT_BACKOFF_MS_CONFIG, 3000);
//ack=0 producer 将不会等待来自 broker 的确认，重试配置不会生效。注意如果被限流了后，就
//ack=1 broker leader 将不会等待所有 broker follower 的确认，就返回 ack。
//ack=all broker leader 将等待所有 broker follower 的确认，才返回 ack。
props.put(ProducerConfig.ACKS_CONFIG, "all");
//构造 Producer 对象，注意，该对象是线程安全的，一般来说，一个进程内一个Producer对象即可。
KafkaProducer<String, String> producer = new KafkaProducer<>(props);

//构造一个消息队列 Kafka 版消息。
String topic = kafkaProperties.getProperty("topic"); //消息所属的Topic，请在控制台E
String value = "this is ckafka msg value"; //消息的内容。

try {
    //批量获取 Future 对象可以加快速度。但注意，批量不要太大。
    List<Future<RecordMetadata>> futures = new ArrayList<>(128);
    for (int i = 0; i < 100; i++) {
        //发送消息，并获得一个Future对象。
        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) {
        //同步获得 Future 对象的结果。
        RecordMetadata recordMetadata = future.get();
    }
}
```

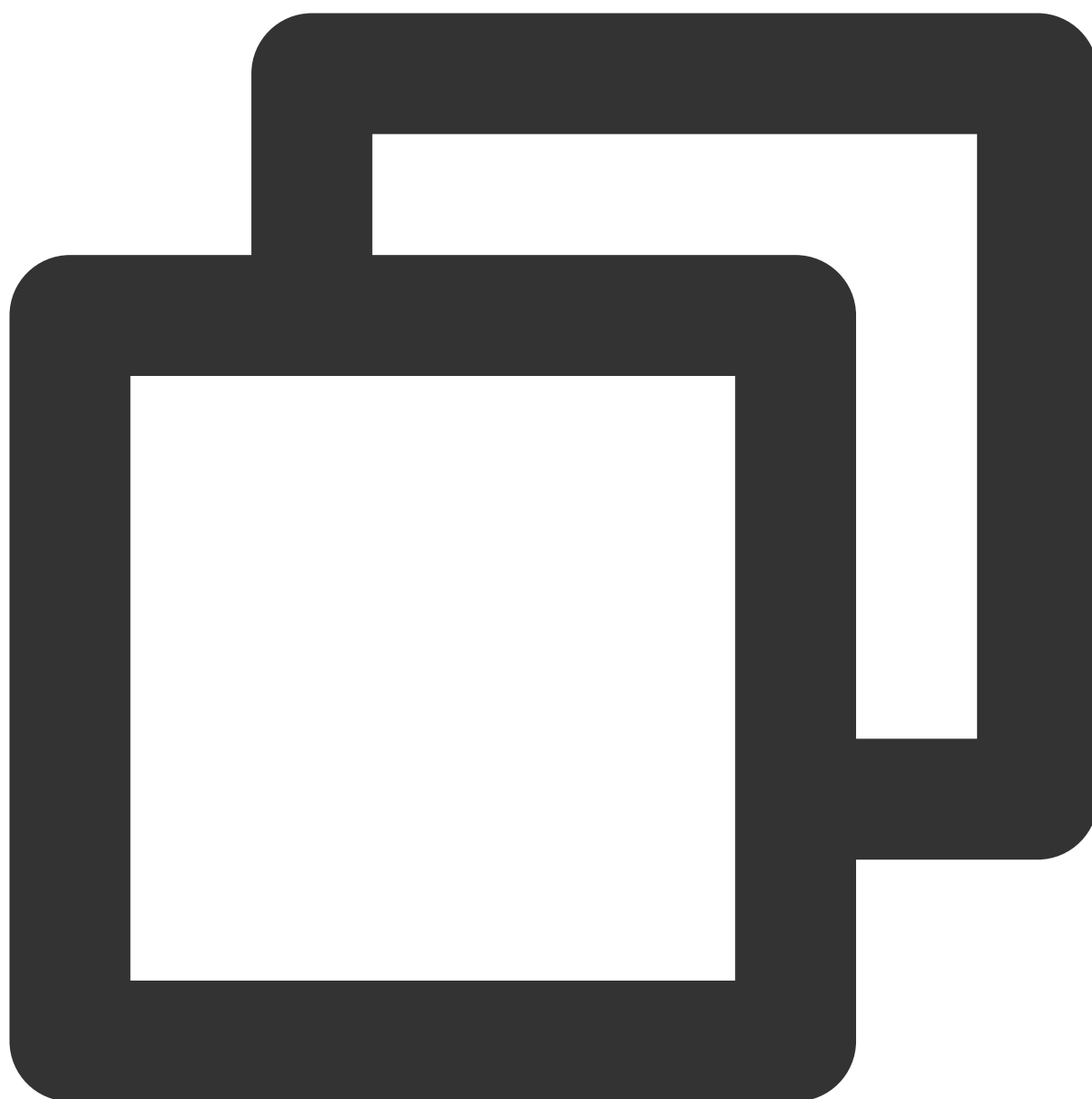


```
        System.out.println("Produce ok:" + recordMetadata.toString());
    }

    } catch (Exception e) {
        //客户端内部重试之后，仍然发送失败，业务要应对此类错误。
        System.out.println("error occurred");
    }
}
}
```

2. 编译并运行 `KafkaSaslProducerDemo.java` 发送消息。

3. 运行结果（输出）。



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

4. 在 CKafka 控制台 **topic 管理** 页面，选择对应的 Topic，单击 **更多 > 消息查询**，查看刚刚发送的消息。

消息查询

消息查询会占用CKafka实例的带宽资源，建议您尽量缩小查询范围查询，不要频繁操作。

消息查询最多展示最近的20条消息。

实例

Topic

test

查询类型

按位点查询

按时间查询

分区ID

0

时间

20

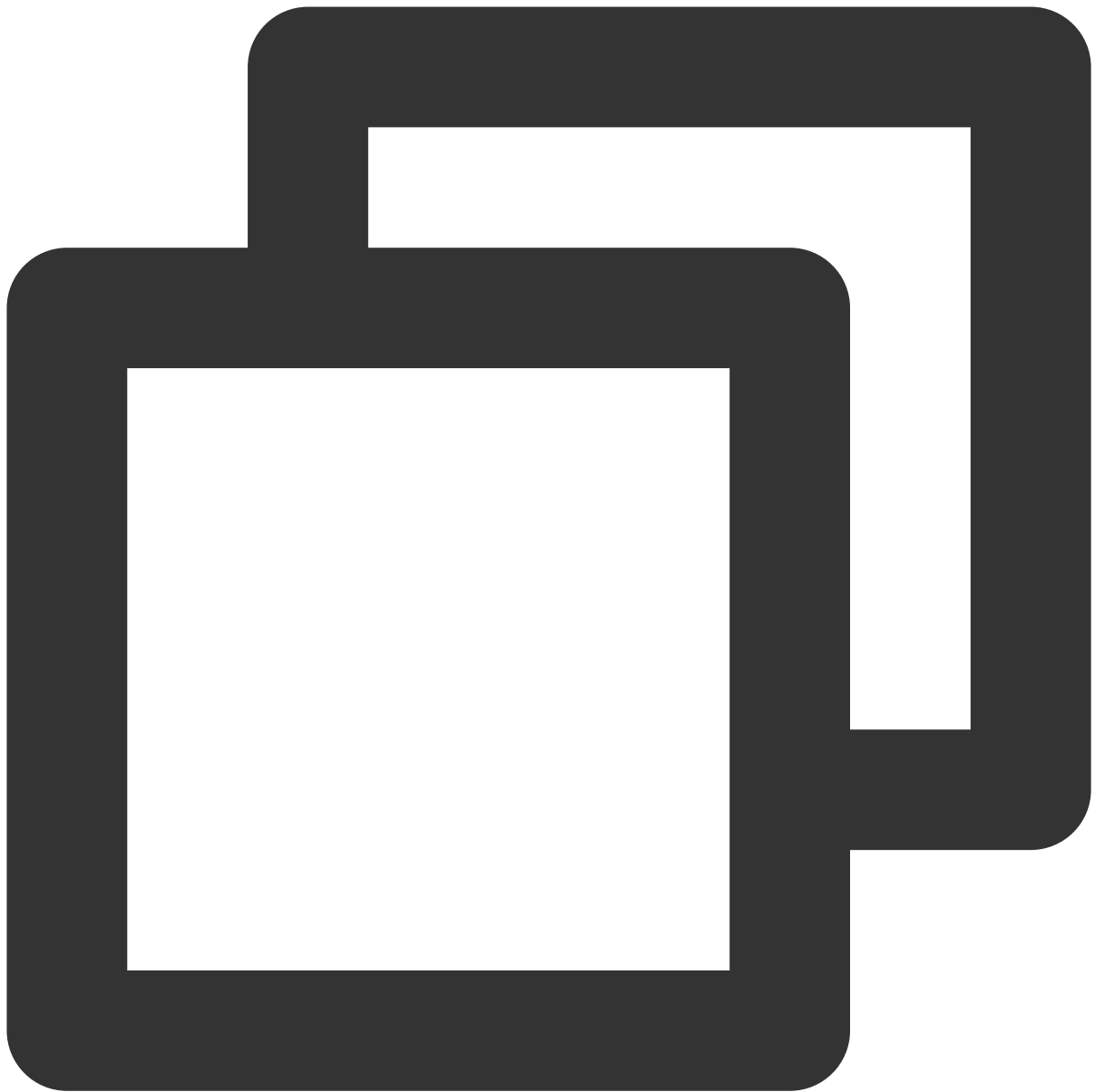
17:22:54

查询

分区ID	位点	时间戳
0	628	17:25:31
0	629	17:25:31

步骤4：消费消息

1. 创建 Consumer 订阅消息程序 `KafkaSaslConsumerDemo.java`。



```
public class KafkaSaslConsumerDemo {  
  
    public static void main(String[] args) {  
        //设置JAAS配置文件的路径。  
        CKafkaConfigurer.configureSaslPlain();  
  
        //加载kafka.properties。  
        Properties kafkaProperties = CKafkaConfigurer.getCKafkaProperties();  
  
        Properties props = new Properties();  
        //设置接入点，请通过控制台获取对应Topic的接入点。  
    }  
}
```

```
props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
    kafkaProperties.getProperty("bootstrap.servers"));

//
// SASL_SCRAM接入
//
props.put(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG, "SASL_PLAINTEXT");
// SASL 采用 Plain 方式。
props.put(SaslConfigs.SASL_MECHANISM, "SCRAM-SHA-256");

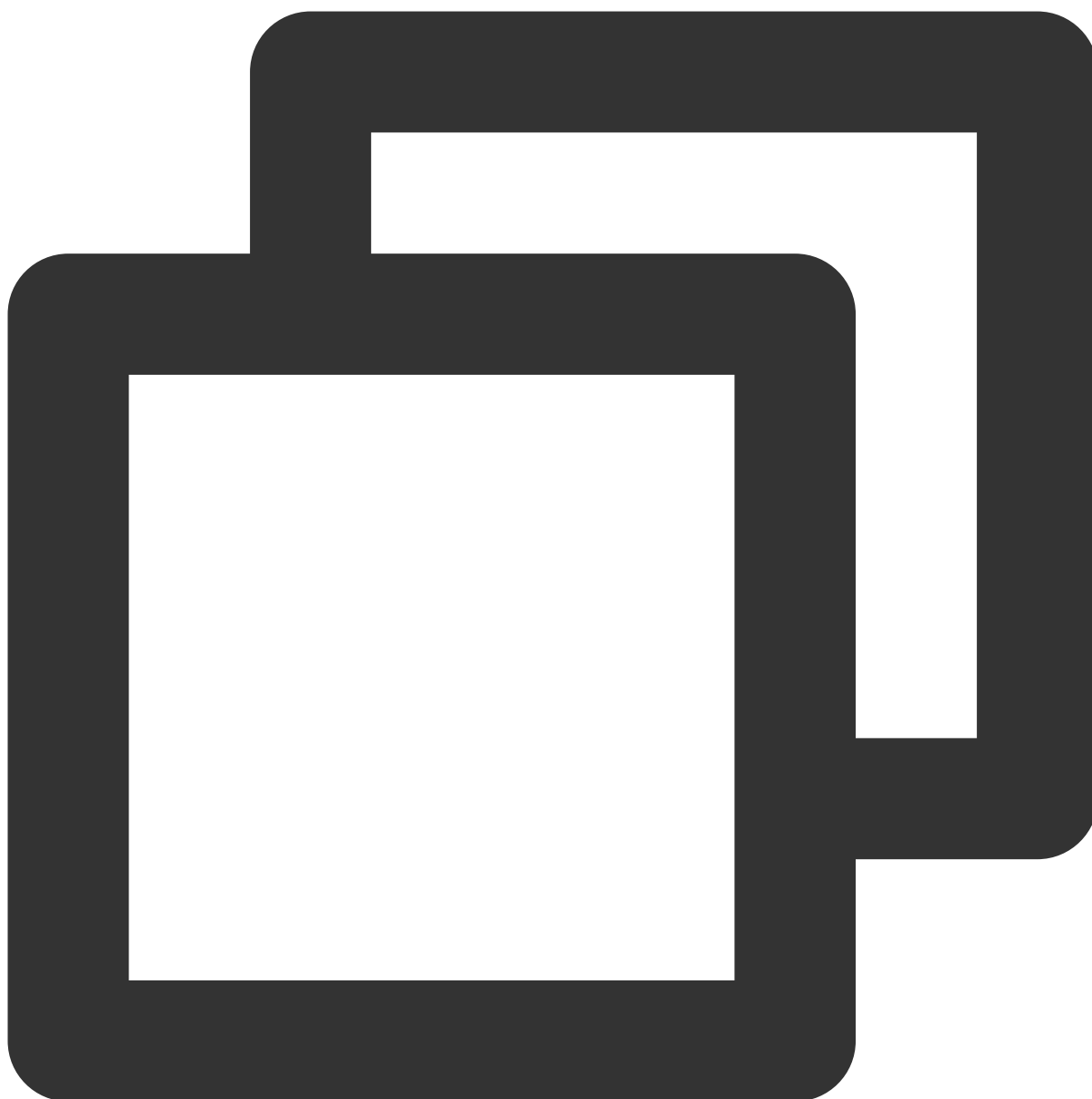
//消费者超时时长
//消费者超过该值没有返回心跳，服务端判断消费者处于非存活状态，服务端将消费者从Consumer Group
props.put(ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG, 30000);
//两次poll的最长时间间隔
//0.10.1.0 版本前这2个概念是混合的，都用session.timeout.ms表示
props.put(ConsumerConfig.MAX_POLL_INTERVAL_MS_CONFIG, 30000);
//每次 Poll 的最大数量。
//注意该值不要改得太大，如果 Poll 太多数据，而不能在下次 Poll 之前消费完，则会触发一次负载均
props.put(ConsumerConfig.MAX_POLL_RECORDS_CONFIG, 30);
//消息的反序列化方式。
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");
//当前消费实例所属的消费组，请在控制台申请之后填写。
//属于同一个组的消费实例，会负载消费消息。
props.put(ConsumerConfig.GROUP_ID_CONFIG, kafkaProperties.getProperty("group.id")
//构造消费对象，也即生成一个消费实例。
KafkaConsumer<String, String> consumer = new KafkaConsumer<String, String>(prop
//设置消费组订阅的 Topic，可以订阅多个。
//如果 GROUP_ID_CONFIG 是一样，则订阅的 Topic 也建议设置成一样。
List<String> subscribedTopics = new ArrayList<String>();
//如果需要订阅多个 Topic，则在这里添加进去即可。
//每个 Topic 需要先在控制台进行创建。
String topicStr = kafkaProperties.getProperty("topic");
String[] topics = topicStr.split(",");
for (String topic : topics) {
    subscribedTopics.add(topic.trim());
}
consumer.subscribe(subscribedTopics);

//循环消费消息。
while (true) {
    try {
        ConsumerRecords<String, String> records = consumer.poll(1000);
        //必须在下次 Poll 之前消费完这些数据，且总耗时不得超过 SESSION_TIMEOUT_MS_CONFIG。
        for (ConsumerRecord<String, String> record : records) {
```

```
        System.out.println(
            String.format("Consume partition:%d offset:%d", record.partition(),
                record.offset()));
    }
} catch (Exception e) {
    System.out.println("consumer error!");
}
}
}
```

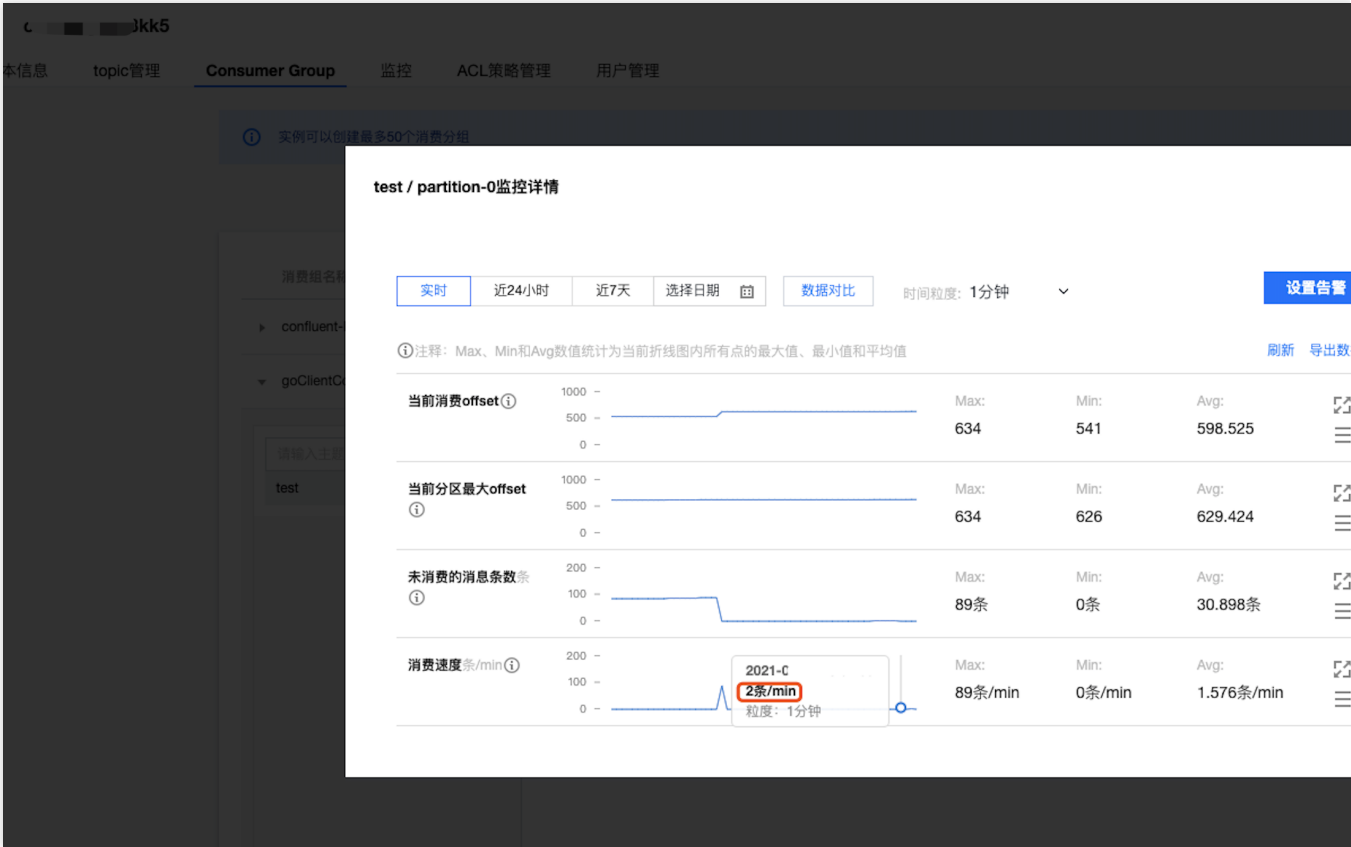
2. 编译并运行 KafkaSaslConsumerDemo.java 消费消息。

3. 运行结果。



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

4. 在 CKafka 控制台 **Consumer Group** 页面，选择对应的消费组名称，在主题名称输入 Topic 名称，单击**查询详情**，查看消费详情。



公网 SASL_PLAINTEXT 方式接入

最近更新时间：2024-05-31 14:40:49

操作背景

该任务以 Java 客户端为例指导您在公网网络环境下，使用 SASL_PLAINTEXT 方式接入消息队列 CKafka 版并收发消息。

前提条件

安装 [1.8 或以上版本 JDK](#)

安装 [2.5 或以上版本 Maven](#)

配置 [ACL 策略](#)

下载 [Demo](#)

操作步骤

步骤1：控制台配置

1. 创建接入点。

1.1 在 [实例列表](#) 页面，单击目标实例 ID，进入实例详情页。

1.2 在 **基本信息 > 接入方式** 中，单击 **添加路由策略**，在打开窗口中选择：**路由类型：公网域名接入**，**接入方式：**
SASL_PLAINTEXT。

添加路由策略

路由类型

公网域名接入

接入方式

SASL_PLAINTEXT

该接入方式提供用户管理和ACL策略配置，以管理用户访问权限

公网带宽

3

198

3

公网带宽收费价格与云服务器公网网络费用 - 按小时带宽 保持一致[公网网络费用](#)

费用

元/小时

提交

关闭

2. 创建角色。

在**用户管理**页面新建角色，设置密码。

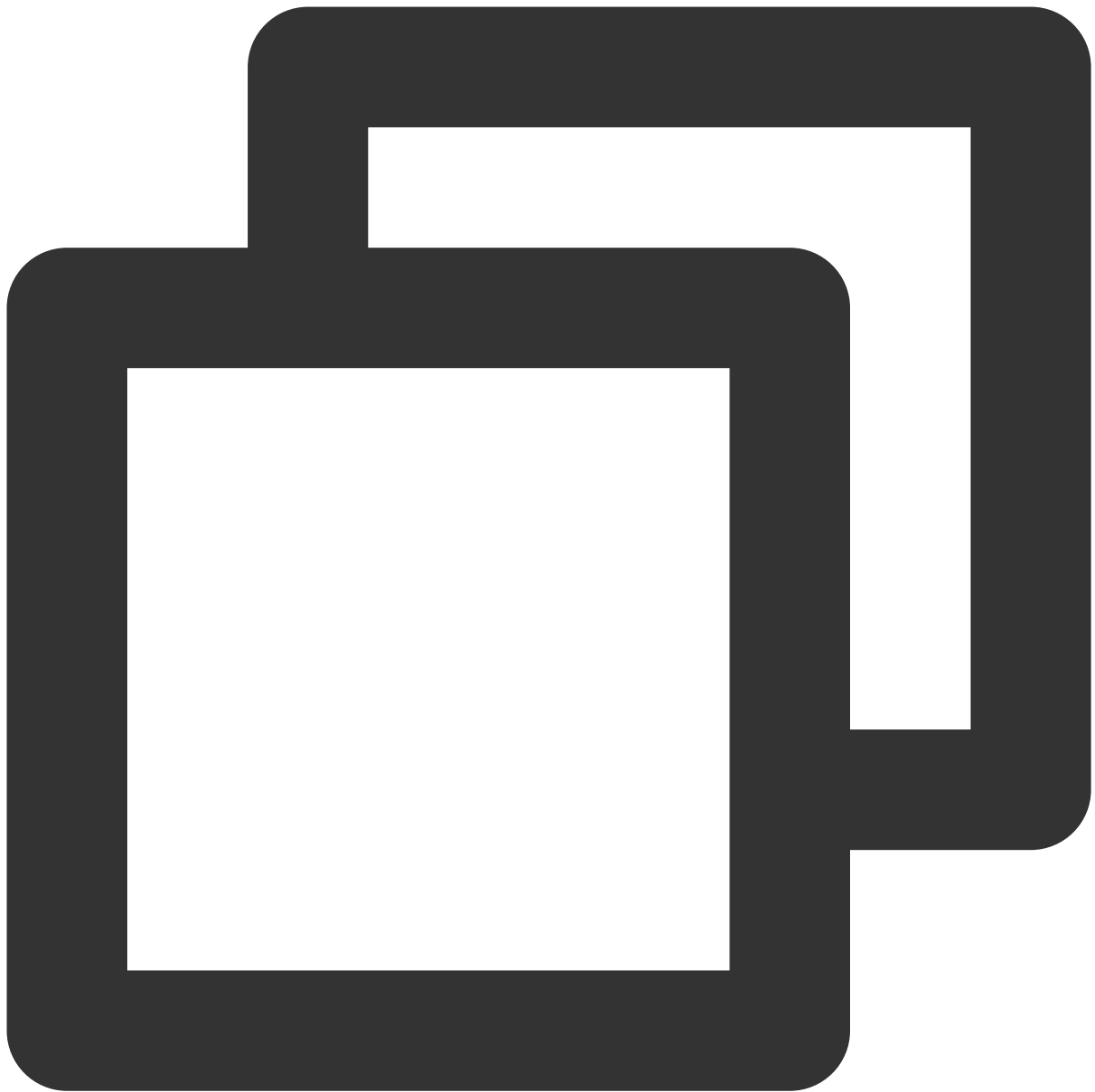
ckafka-			
基本信息topic管理Consumer Group监控事件中心HTTP接入 ACL策略管理 智能运维专业版			
策略列表 用户管理			
新建			
用户名	创建/更新时间		操作
ckafka- user1@	2022-11-03 16:05:47 2022-11-03 16:05:47		修改密码

3. 创建 Topic。

在控制台**Topic 管理**页面新建 Topic（参见 [创建 Topic](#)）。

步骤2：添加配置文件

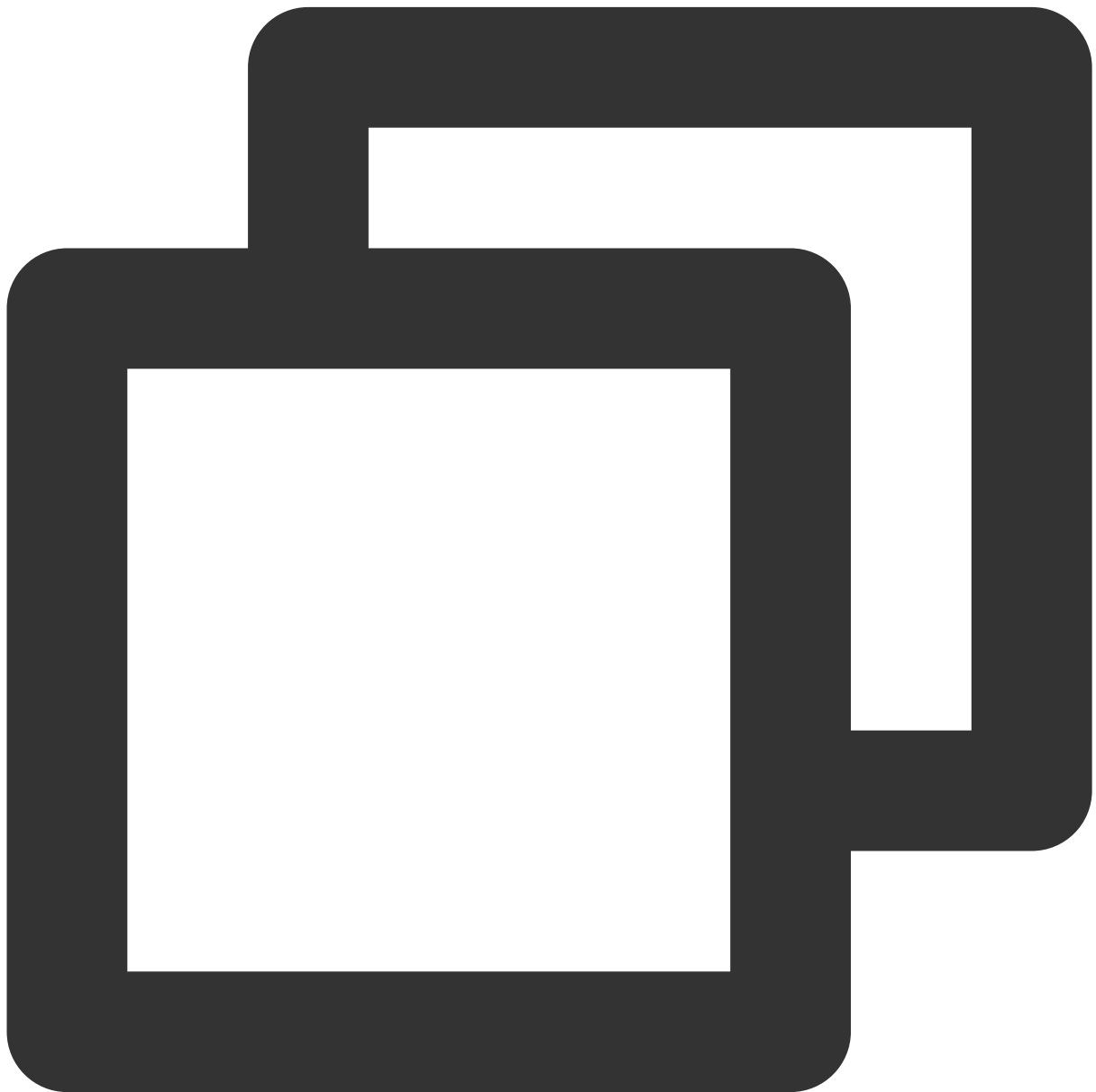
1. 在 pom.xml 中添加以下依赖。



```
<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>
</dependencies>
```

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

2. 创建 JAAS 配置文件 `ckafka_client_jaas.conf`，使用**用户管理**界面创建的用户进行修改。



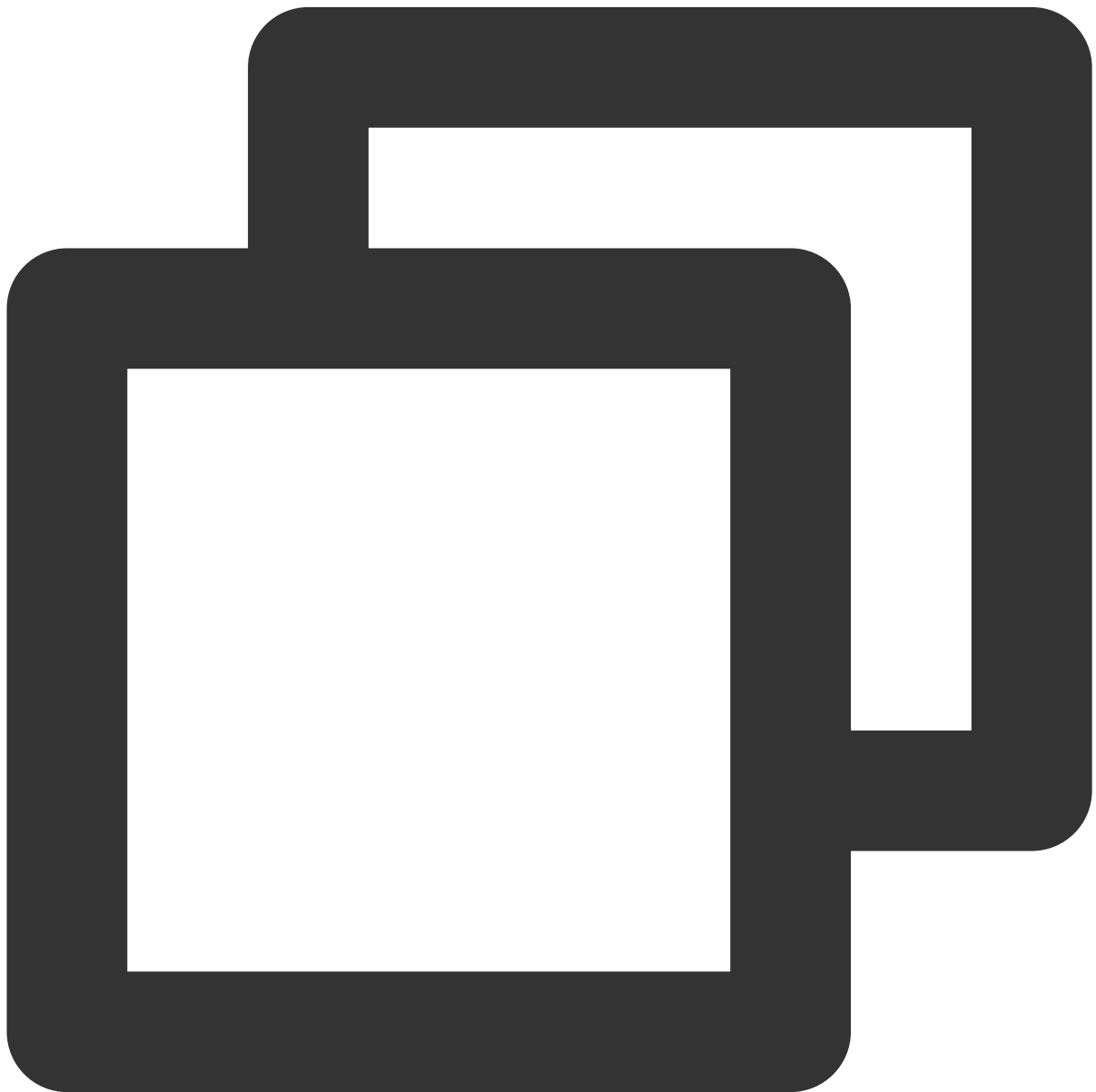
```
KafkaClient {
```

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

说明：

username 是 实例 ID + # + 配置的用户名，password 是配置的用户密码。

3. 创建消息队列 CKafka 版配置文件 kafka.properties。

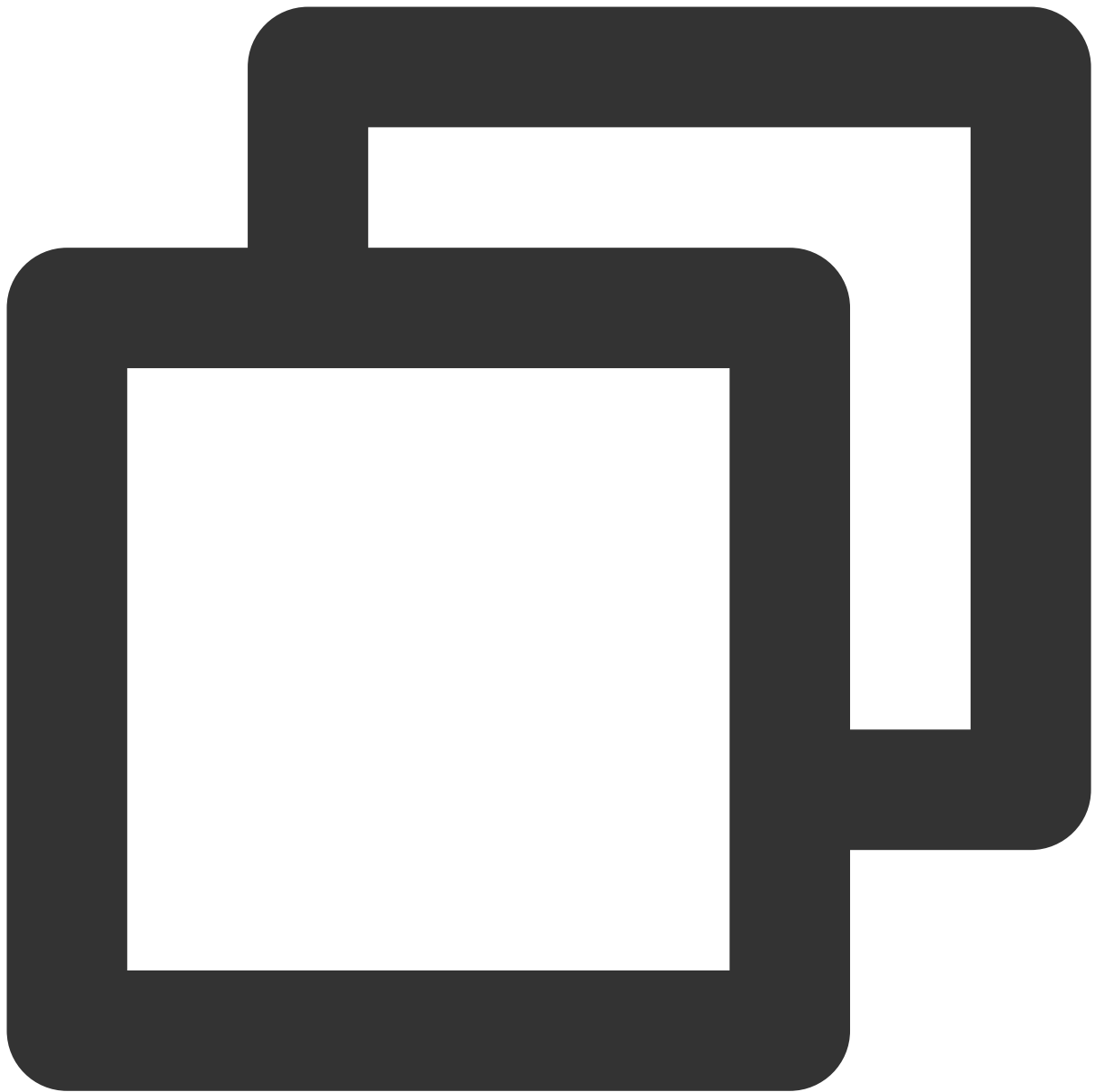


配置接入网络，在控制台的实例详情页面接入方式模块的网络列复制。

```
bootstrap.servers=ckafka-xxxxxxx
## 配置 Topic, 在控制台上 topic 管理页面复制。
topic=XXX
## 配置 consumer group, 您可以自定义设置
group.id=XXX
## SASL 配置
java.security.auth.login.config=/xxxx/ckafka_client_jaas.conf
```

参数	说明																		
bootstrap.servers	接入网络，在控制台的实例详情页面接入方式模块的网络列复制。 接入方式? <table><tr><td>接入类型</td><td>接入方式</td><td>网络</td><td>操作</td></tr><tr><td>公网域名接入</td><td>SASL_PLAINTEXT</td><td>ckafka-l4xppqrz...</td><td>删除</td></tr></table>	接入类型	接入方式	网络	操作	公网域名接入	SASL_PLAINTEXT	ckafka-l4xppqrz...	删除										
接入类型	接入方式	网络	操作																
公网域名接入	SASL_PLAINTEXT	ckafka-l4xppqrz...	删除																
topic	Topic 名称，您可以在控制台上 topic管理页面复制。 ckafka-pl... topic管理 Consumer Group 监控 事件中心 HTTP接入 ACL策略管理 智能运维 集群备份 新建(24/450) 请输入 <table><tr><td>ID/名称</td><td>监控</td><td>分区数(个)</td><td>副本数(个)</td><td>标签</td><td>备注</td><td>创建时间</td><td>消息保留时间</td><td>状态</td></tr><tr><td>topic-j-coos ID</td><td></td><td>3</td><td>2</td><td></td><td></td><td>2023-03-25 23:03:00</td><td>2 天</td><td>正常</td></tr></table>	ID/名称	监控	分区数(个)	副本数(个)	标签	备注	创建时间	消息保留时间	状态	topic-j-coos ID		3	2			2023-03-25 23:03:00	2 天	正常
ID/名称	监控	分区数(个)	副本数(个)	标签	备注	创建时间	消息保留时间	状态											
topic-j-coos ID		3	2			2023-03-25 23:03:00	2 天	正常											
group.id	您可以自定义设置， Demo 运行成功后可以在 Consumer Group 页面看到该																		
java.security.auth.login.config.plain	填写 JAAS 配置文件 kafka_client_jaas.conf 的路径。																		

4. 创建配置文件加载程序 CKafkaConfigurer.java。



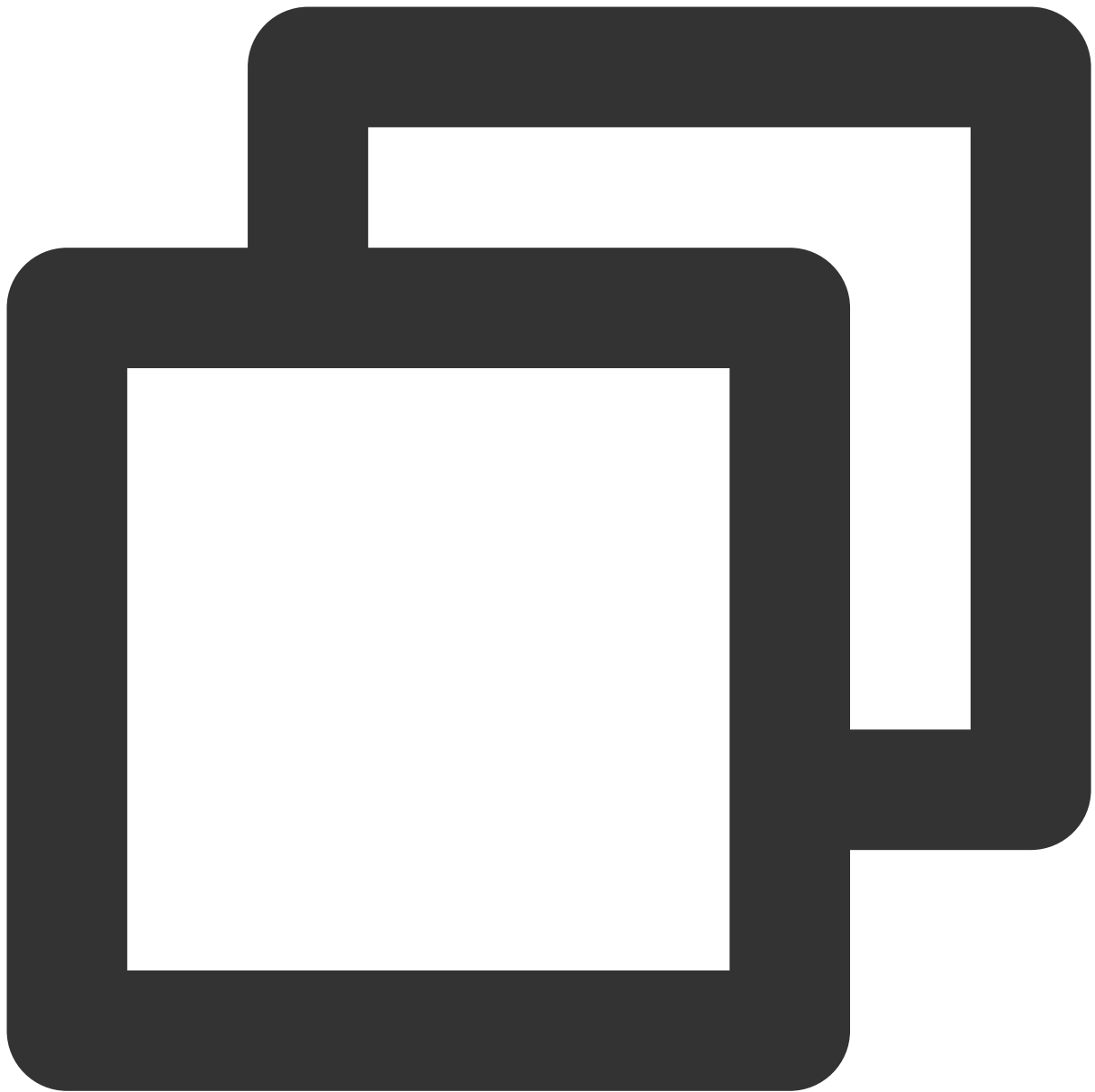
```
public class CKafkaConfigurer {  
  
    private static Properties properties;  
  
    public static void configureSaslPlain() {  
        //如果用 -D 或者其它方式设置过，这里不再设置。  
        if (null == System.getProperty("java.security.auth.login.config")) {  
            //请注意将 xxx 修改为自己的路径。  
            System.setProperty("java.security.auth.login.config",  
                               getCKafkaProperties().getProperty("java.security.auth.login.con  
        }  
    }  
}
```

```
}

public synchronized static Properties getCKafkaProperties() {
    if (null != properties) {
        return properties;
    }
    //获取配置文件 kafka.properties 的内容。
    Properties kafkaProperties = new Properties();
    try {
        kafkaProperties.load(CKafkaProducerDemo.class.getClassLoader().getResou
    } catch (Exception e) {
        System.out.println("getCKafkaProperties error");
    }
    properties = kafkaProperties;
    return kafkaProperties;
}
}
```

步骤3：发送消息

1. 创建发送消息程序 `KafkaSaslProducerDemo.java`。



```
public class KafkaSaslProducerDemo {  
  
    public static void main(String[] args) {  
        //设置 JAAS 配置文件的路径。  
        CKafkaConfigurer.configureSaslPlain();  
  
        //加载 kafka.properties。  
        Properties kafkaProperties = CKafkaConfigurer.getCKafkaProperties();  
  
        Properties props = new Properties();  
        //设置接入点，请通过控制台获取对应 Topic 的接入点。  
    }  
}
```

```
props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
          kafkaProperties.getProperty("bootstrap.servers"));

//
// SASL_PLAINTEXT 公网接入
//
props.put(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG, "SASL_PLAINTEXT");
// SASL 采用 Plain 方式。
props.put(SaslConfigs.SASL_MECHANISM, "PLAIN");

//消息队列 Kafka 版消息的序列化方式。
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");
//请求的最长等待时间。
props.put(ProducerConfig.MAX_BLOCK_MS_CONFIG, 30 * 1000);
//设置客户端内部重试次数。
props.put(ProducerConfig.RETRIES_CONFIG, 5);
//设置客户端内部重试间隔。
props.put(ProducerConfig.RECONNECT_BACKOFF_MS_CONFIG, 3000);
//ack=0 producer 将不会等待来自 broker 的确认，重试配置不会生效。注意如果被限流了后，
//ack=1 broker leader 将不会等待所有 broker follower 的确认，就返回 ack。
//ack=all broker leader 将等待所有 broker follower 的确认，才返回 ack。
props.put(ProducerConfig.ACKS_CONFIG, "all");
//构造 Producer 对象，注意，该对象是线程安全的，一般来说，一个进程内一个Producer对象即可。
KafkaProducer<String, String> producer = new KafkaProducer<>(props);

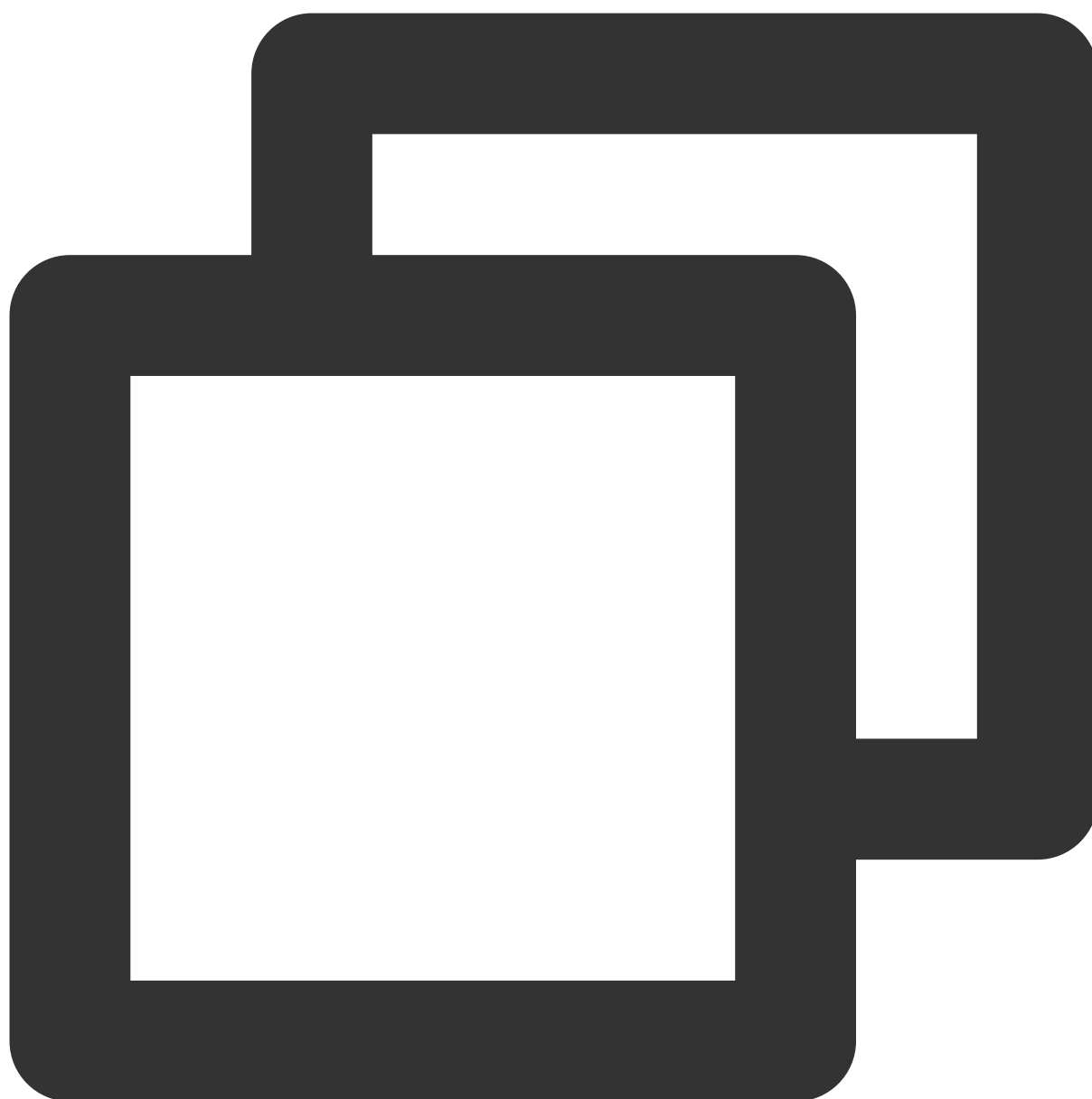
//构造一个消息队列 Kafka 版消息。
String topic = kafkaProperties.getProperty("topic"); //消息所属的Topic，请在控制
String value = "this is ckafka msg value"; //消息的内容。

try {
    //批量获取 Future 对象可以加快速度。但注意，批量不要太大。
    List<Future<RecordMetadata>> futures = new ArrayList<>(128);
    for (int i = 0; i < 100; i++) {
        //发送消息，并获得一个Future对象。
        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) {
        //同步获得 Future 对象的结果。
        RecordMetadata recordMetadata = future.get();
    }
}
```

```
        System.out.println("Produce ok:" + recordMetadata.toString());
    }
} catch (Exception e) {
    //客户端内部重试之后，仍然发送失败，业务要应对此类错误。
    System.out.println("error occurred");
}
}
}
```

2. 编译并运行 `KafkaSaslProducerDemo.java` 发送消息。

3. 运行结果（输出）。



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

4. 在 CKafka 控制台 **topic 管理** 页面，选择对应的 Topic，单击 **更多 > 消息查询**，查看刚刚发送的消息。

消息查询

消息查询会占用CKafka实例的带宽资源，建议您尽量缩小查询范围查询，不要频繁操作。

消息查询最多展示最近的20条消息。

实例

Topic

test

查询类型

按位点查询

按时间查询

分区ID

0

时间

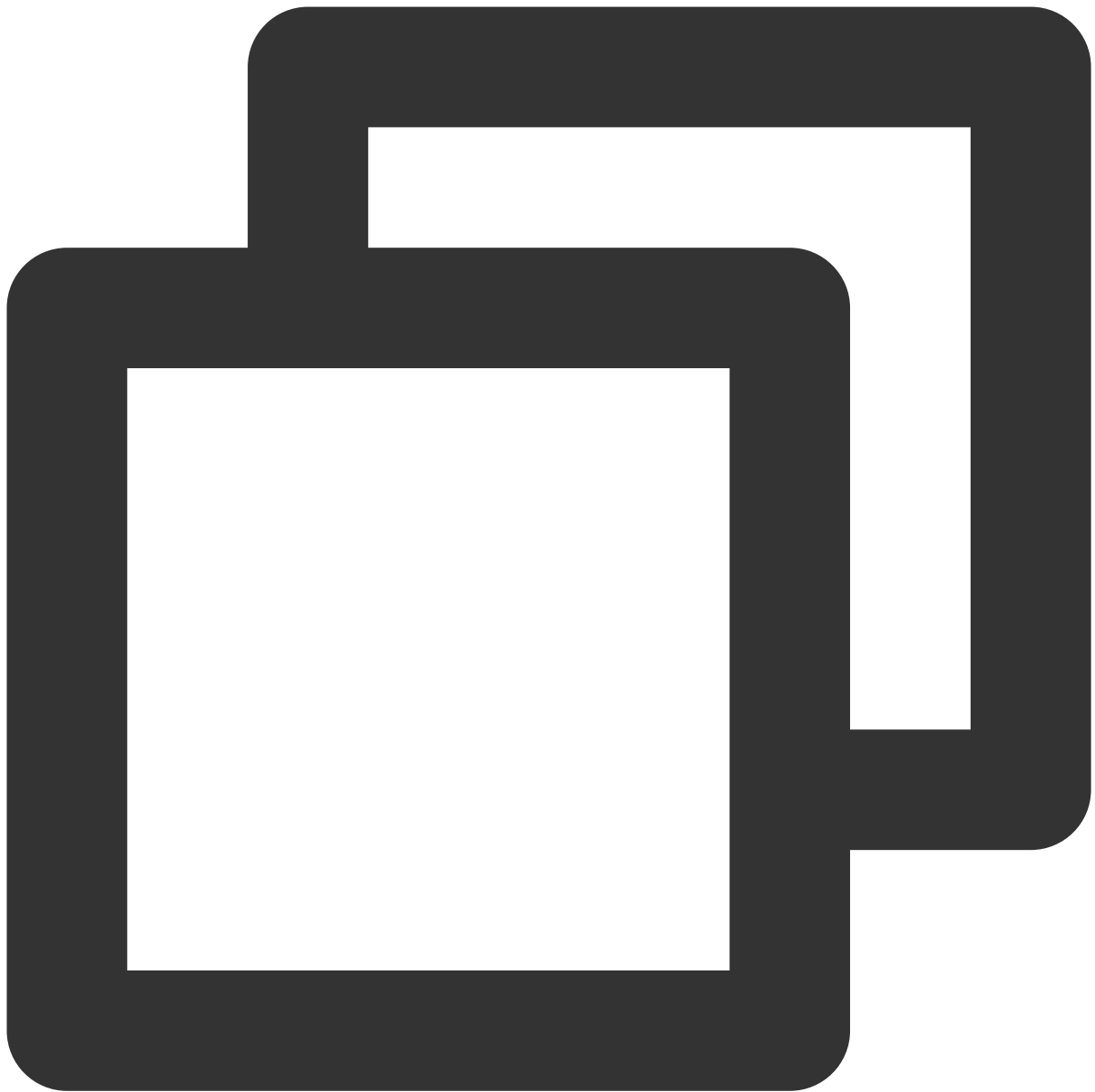
2023-07-26 17:22:54

查询

分区ID	位点	时间戳
0	628	2023-07-26 17:25:31
0	629	2023-07-26 17:25:31

步骤4：消费消息

1. 创建 Consumer 订阅消息程序 `KafkaSaslConsumerDemo.java`。



```
public class KafkaSaslConsumerDemo {  
  
    public static void main(String[] args) {  
        //设置JAAS配置文件的路径。  
        CKafkaConfigurer.configureSaslPlain();  
  
        //加载kafka.properties。  
        Properties kafkaProperties = CKafkaConfigurer.getCKafkaProperties();  
  
        Properties props = new Properties();  
        //设置接入点，请通过控制台获取对应Topic的接入点。  
    }  
}
```

```
props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
    kafkaProperties.getProperty("bootstrap.servers"));

//
// SASL_PLAINTEXT 公网接入
//
props.put(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG, "SASL_PLAINTEXT");
// SASL 采用 Plain 方式。
props.put(SaslConfigs.SASL_MECHANISM, "PLAIN");

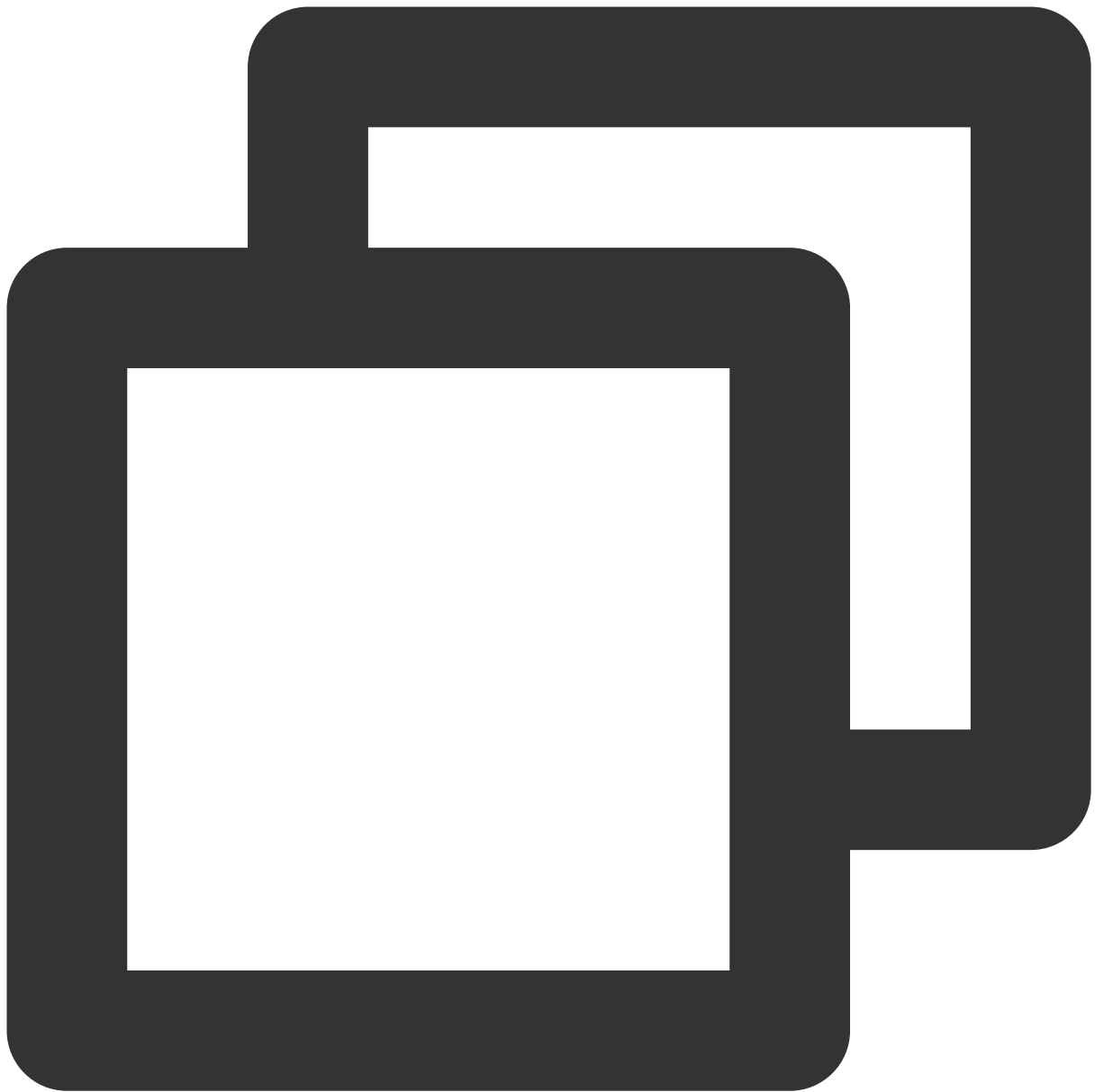
//消费者超时时长
//消费者超过该值没有返回心跳，服务端判断消费者处于非存活状态，服务端将消费者从Consumer Group
props.put(ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG, 30000);
//两次poll的最长时间间隔
//0.10.1.0 版本前这2个概念是混合的，都用session.timeout.ms表示
props.put(ConsumerConfig.MAX_POLL_INTERVAL_MS_CONFIG, 30000);
//每次 Poll 的最大数量。
//注意该值不要改得太大，如果 Poll 太多数据，而不能在下次 Poll 之前消费完，则会触发一次负载均衡
props.put(ConsumerConfig.MAX_POLL_RECORDS_CONFIG, 30);
//消息的反序列化方式。
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");
//当前消费实例所属的消费组，请在控制台申请之后填写。
//属于同一个组的消费实例，会负载消费消息。
props.put(ConsumerConfig.GROUP_ID_CONFIG, kafkaProperties.getProperty("group.id"));
//构造消费对象，也即生成一个消费实例。
KafkaConsumer<String, String> consumer = new KafkaConsumer<String, String>(props);
//设置消费组订阅的 Topic，可以订阅多个。
//如果 GROUP_ID_CONFIG 是一样，则订阅的 Topic 也建议设置成一样。
List<String> subscribedTopics = new ArrayList<String>();
//如果需要订阅多个 Topic，则在这里添加进去即可。
//每个 Topic 需要先在控制台进行创建。
String topicStr = kafkaProperties.getProperty("topic");
String[] topics = topicStr.split(",");
for (String topic : topics) {
    subscribedTopics.add(topic.trim());
}
consumer.subscribe(subscribedTopics);

//循环消费消息。
while (true) {
    try {
        ConsumerRecords<String, String> records = consumer.poll(1000);
        //必须在下次 Poll 之前消费完这些数据，且总耗时不得超过 SESSION_TIMEOUT_MS_CONFIG
        for (ConsumerRecord<String, String> record : records) {
```

```
        System.out.println(
            String.format("Consume partition:%d offset:%d", record.partition(),
                record.offset()));
    }
    } catch (Exception e) {
        System.out.println("consumer error!");
    }
    }
    }
}
```

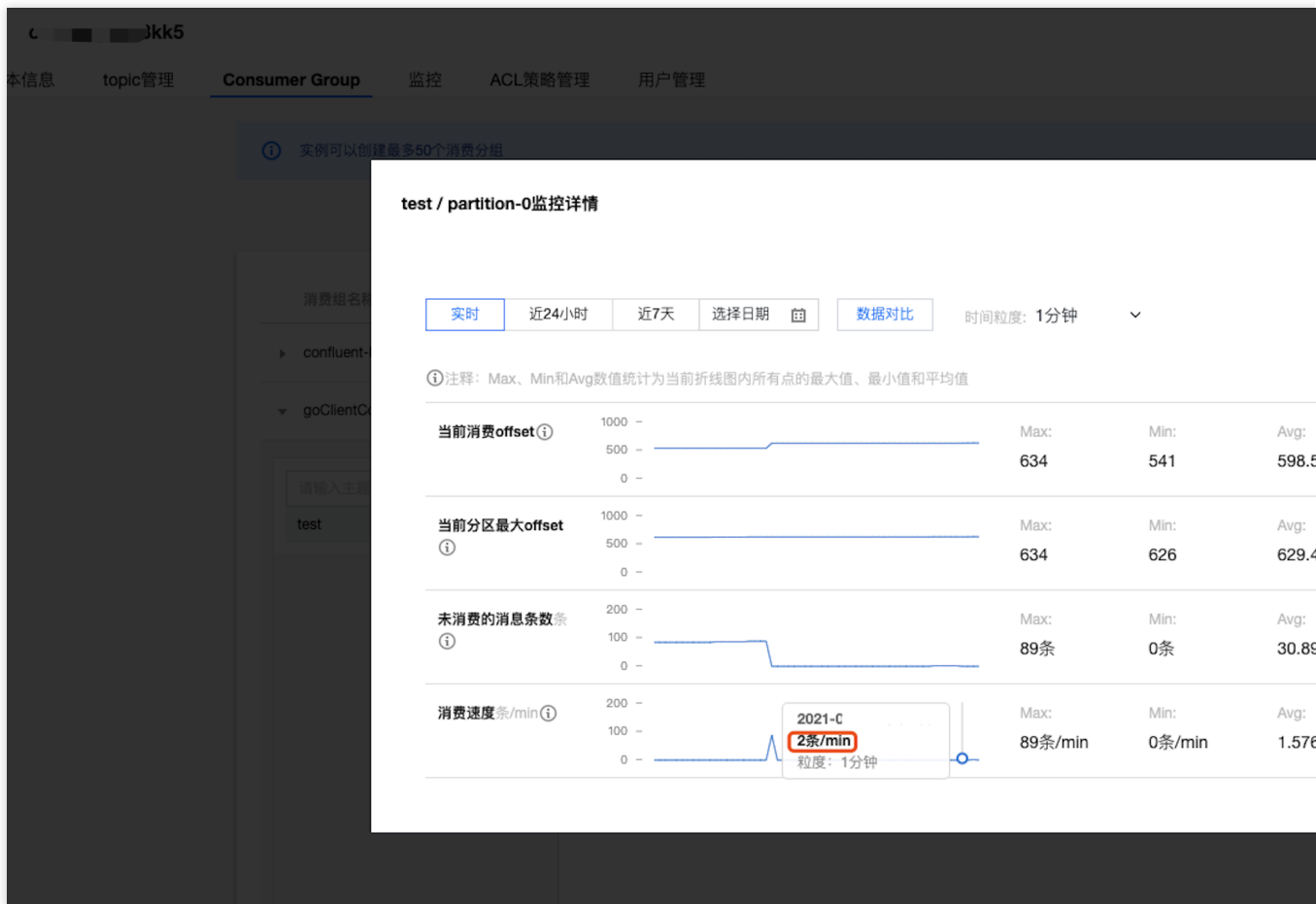
2. 编译并运行 KafkaSaslConsumerDemo.java 消费消息。

3. 运行结果。



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

4. 在 CKafka 控制台 **Consumer Group** 页面，选择对应的消费组名称，在主题名称输入 Topic 名称，单击[查询详情](#)，查看消费详情。



公网 SASL_SSL 方式接入

最近更新时间：2024-05-31 14:40:49

操作背景

该任务以 Java 客户端为例指导您在公网网络环境下，使用 SASL_SSL 方式接入消息队列 CKafka 版并收发消息。SSL 证书的核心功能是保护服务器-客户端通信。数据通过 SSL 证书加密，其他人无法拥有解锁它的私钥，只能由预期的服务端解锁。

前提条件

- [安装 1.8 或以上版本 JDK](#)
- [安装 2.5 或以上版本 Maven](#)
- [配置 ACL 策略](#)
- [下载 Demo](#)
- [下载 SASL_SSL 证书](#)

操作步骤

步骤1：控制台配置

- 创建接入点。
 - 在 [实例列表](#) 页面，单击目标实例 ID，进入实例详情页。
 - 在 **基本信息** > **接入方式** 中，单击 **添加路由策略**，在打开窗口中选择：**路由类型**：公网域名接入，**接入方式**：SASL_SSL。

添加路由策略

路由类型

公网域名接入

接入方式

SASL_SSL

公网带宽

3

198

公网带宽收费价格与云服务器公网网络费用 - 按小时带宽 保持一致[公网网络费用](#)

费用

0.00元/小时

提交

关闭

2. 创建角色。

在**用户管理**页面新建角色，设置密码。

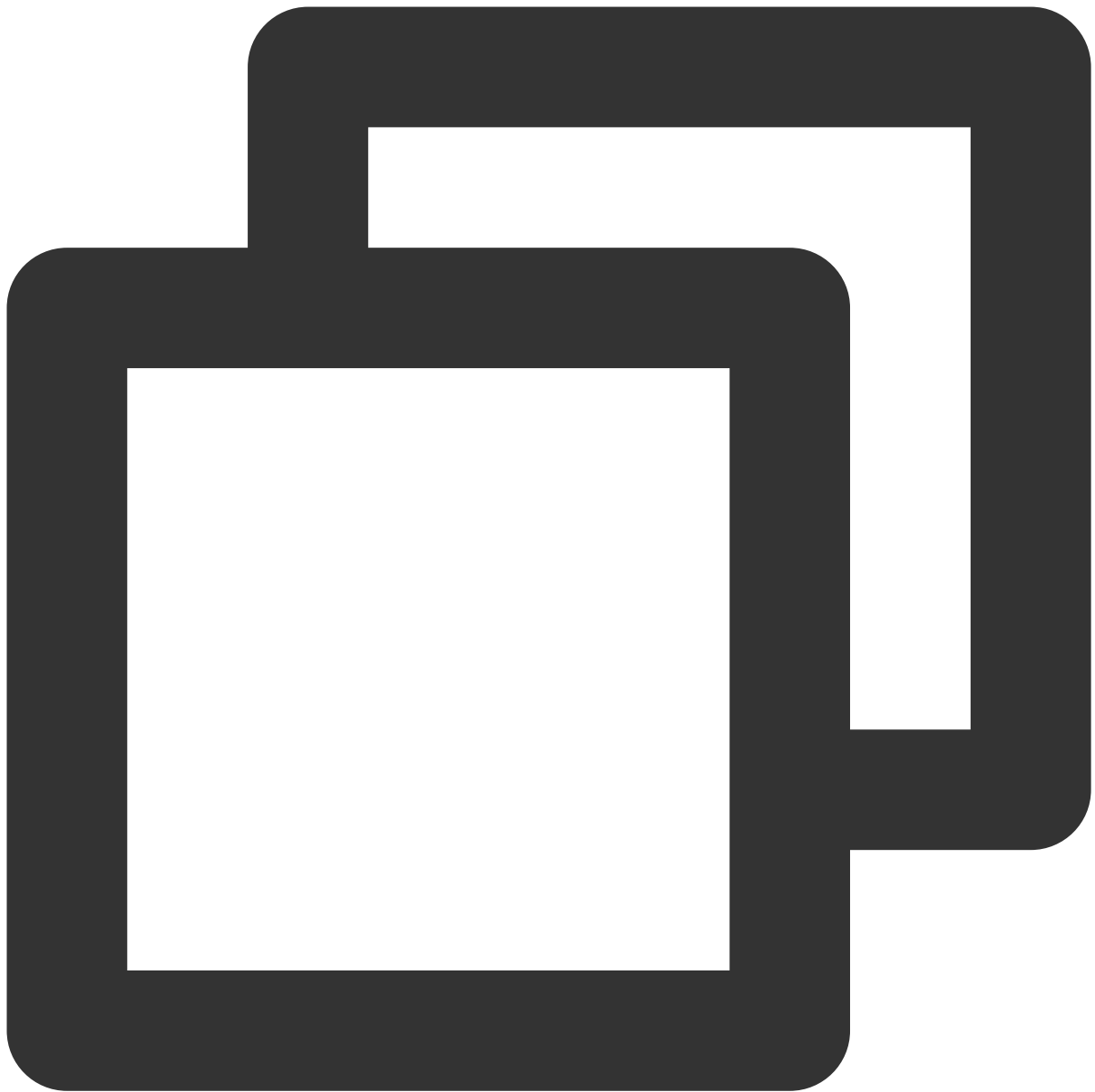
ckafka-			
基本信息topic管理Consumer Group监控事件中心HTTP接入ACL策略管理智能运维			
策略列表用户管理			
新建			
用户名	创建/更新时间	操作	
ckafka- user1	2022-11-03 18:05:47 2022-11-03 18:05:47	修改策略	

3. 创建 Topic。

在控制台 **topic 管理** 页面新建 Topic（参见 [创建 Topic](#)）。

步骤2：添加配置文件

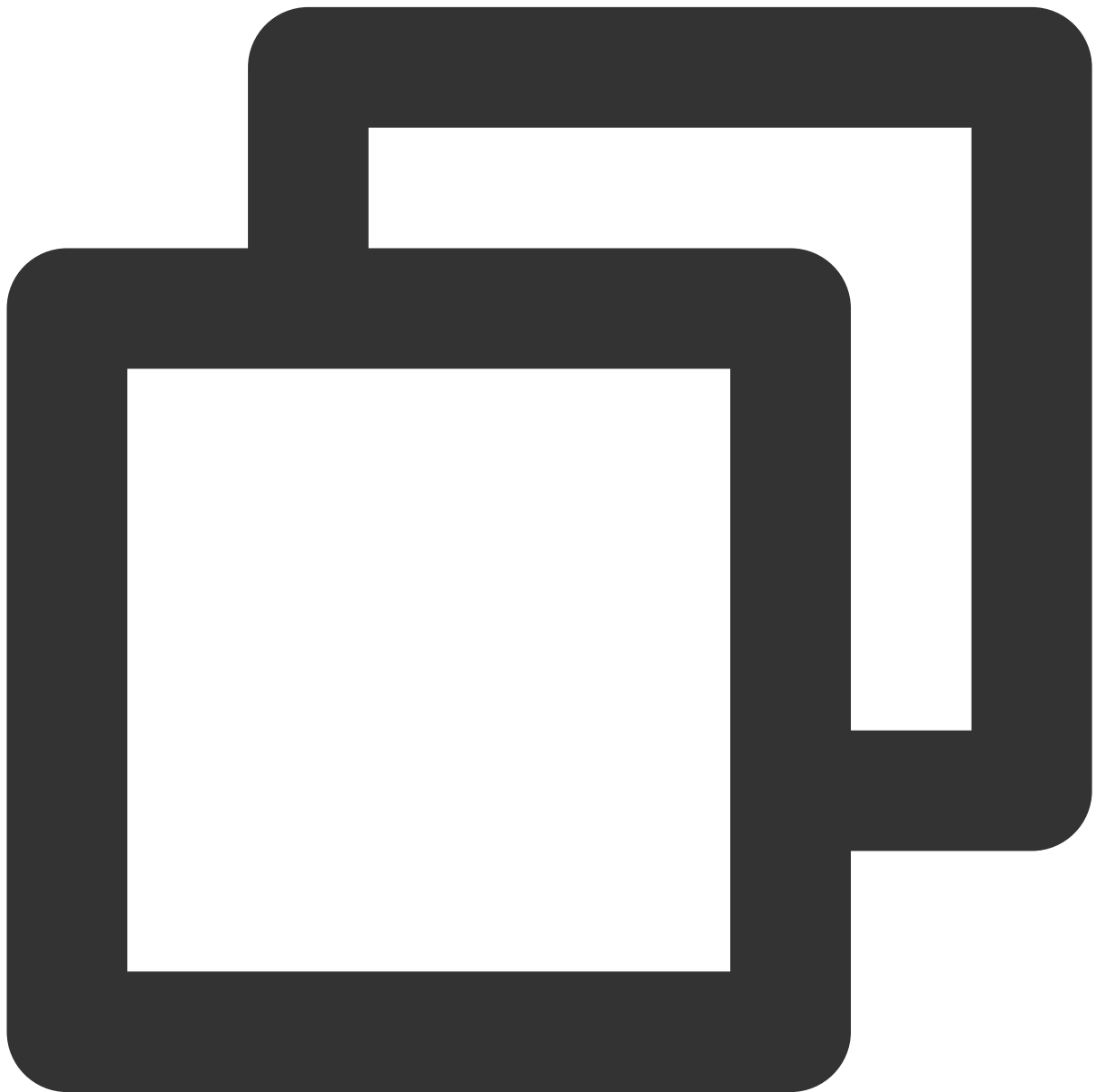
1. 在 pom.xml 中添加以下依赖。



```
<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>
</dependencies>
```

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

2. 创建 JAAS 配置文件 `ckafka_client_jaas.conf`，使用**用户管理**界面创建的用户进行修改。



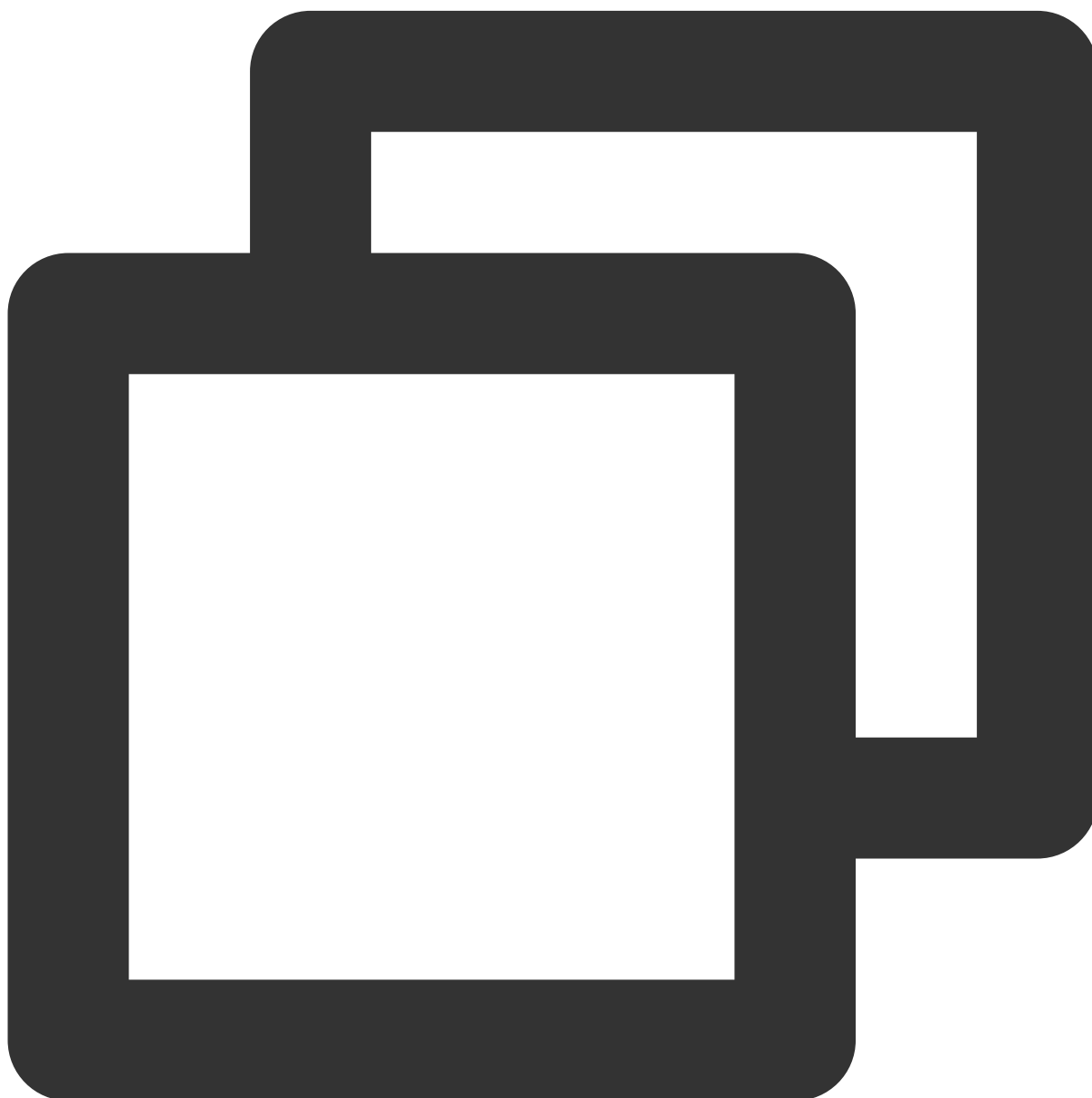
```
KafkaClient {
```

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

说明：

username 是 实例 ID + # + 配置的用户名，password 是配置的用户密码。

3. 创建消息队列 CKafka 版配置文件 kafka.properties。

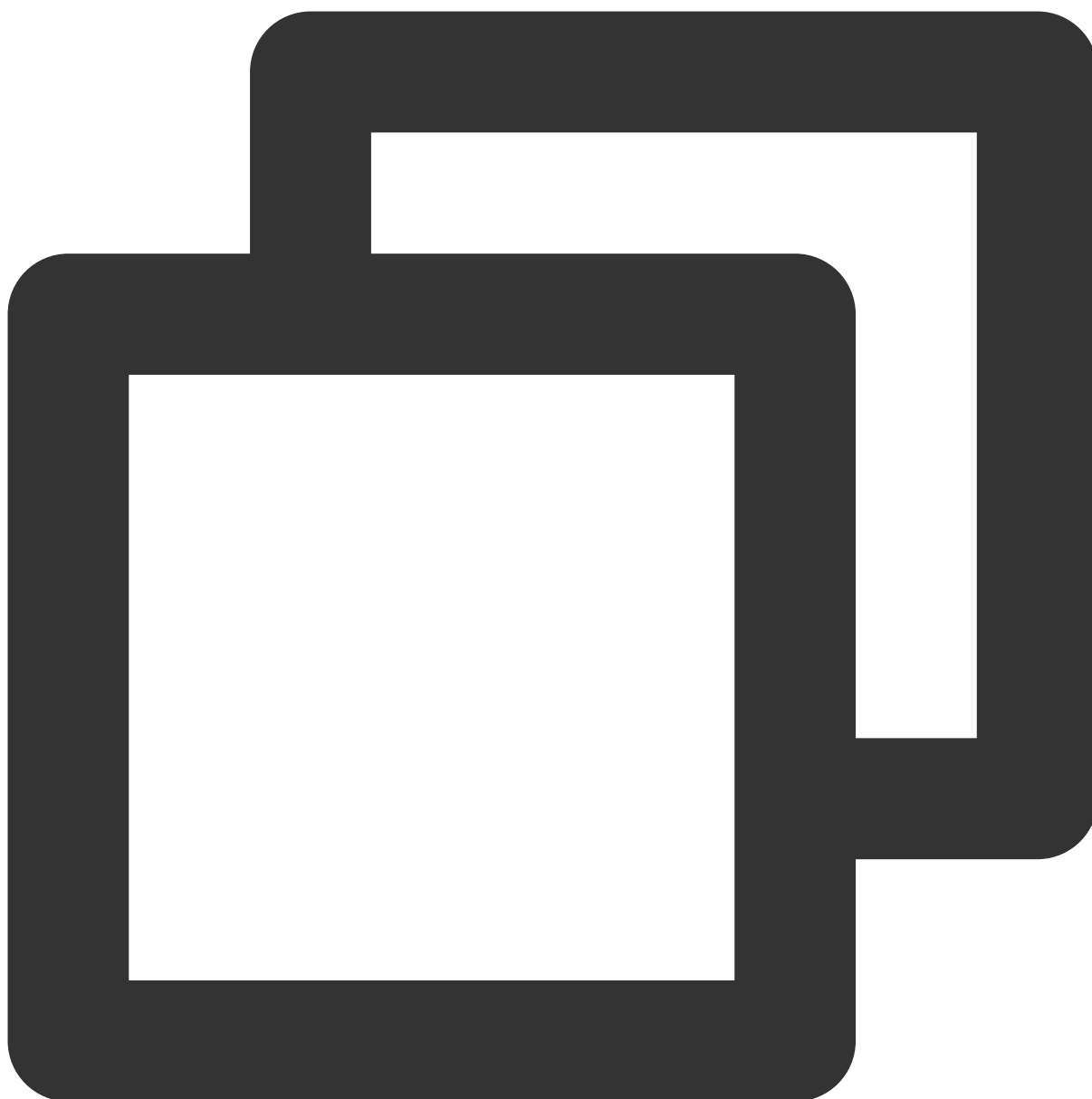


配置接入网络，在控制台的实例详情页面接入方式模块的网络列复制。

```
bootstrap.servers=ckafka-xxxxxxx
## 配置 Topic, 在控制台上 topic 管理页面复制。
topic=XXX
## 配置 consumer group, 您可以自定义设置
group.id=XXX
## SASL 配置
java.security.auth.login.config=/xxxx/ckafka_client_jaas.conf
## SSL 证书配置, 接入方式选择为 SASL_SSL 时生效
ssl.truststore.location=/xxxx/client.truststore.jks
ssl.truststore.password=5fi6R!M
ssl.endpoint.identification.algorithm=
```

参数	说明																		
bootstrap.servers	接入网络，在控制台的实例详情页面接入方式模块的网络列复制。 接入方式 ② <table><tr><td>接入类型</td><td>接入方式</td><td>网络</td><td>操作</td></tr><tr><td>公网域名接入</td><td>SASL_SSL</td><td>ckafka-gnzmqj... 159.75.176.66:50001</td><td>删除 查看所有IP和</td></tr></table>	接入类型	接入方式	网络	操作	公网域名接入	SASL_SSL	ckafka-gnzmqj... 159.75.176.66:50001	删除 查看所有IP和										
接入类型	接入方式	网络	操作																
公网域名接入	SASL_SSL	ckafka-gnzmqj... 159.75.176.66:50001	删除 查看所有IP和																
topic	Topic 名称，您可以在控制台上 topic管理页面复制。 ckafka-p... ← 基本信息 topic管理 Consumer Group 监控 事件中心 HTTP接入 ACL策略管理 智能运维 专业版 集群备份 专业版 新建(24/480) <table><tr><th>ID/名称</th><th>监控</th><th>分区数(个)</th><th>副本数(个)</th><th>标签</th><th>备注</th><th>创建时间</th><th>消息保留时间</th><th>状态</th></tr><tr><td>topic-j cocs</td><td></td><td>3</td><td>2</td><td></td><td></td><td>2023-03-25 23:03:00</td><td>2 天</td><td>正常</td></tr></table>	ID/名称	监控	分区数(个)	副本数(个)	标签	备注	创建时间	消息保留时间	状态	topic-j cocs		3	2			2023-03-25 23:03:00	2 天	正常
ID/名称	监控	分区数(个)	副本数(个)	标签	备注	创建时间	消息保留时间	状态											
topic-j cocs		3	2			2023-03-25 23:03:00	2 天	正常											
group.id	您可以自定义设置，Demo 运行成功后可以在 Consumer Group 页面看到该																		
java.security.auth.login.config.plain	填写 JAAS 配置文件 ckafka_client_jaas.conf 的路径。																		
client.truststore.jks	采用 SASL_SSL 方式接入时，所需的证书路径。																		

4. 创建配置文件加载程序 CKafkaConfigurer.java。

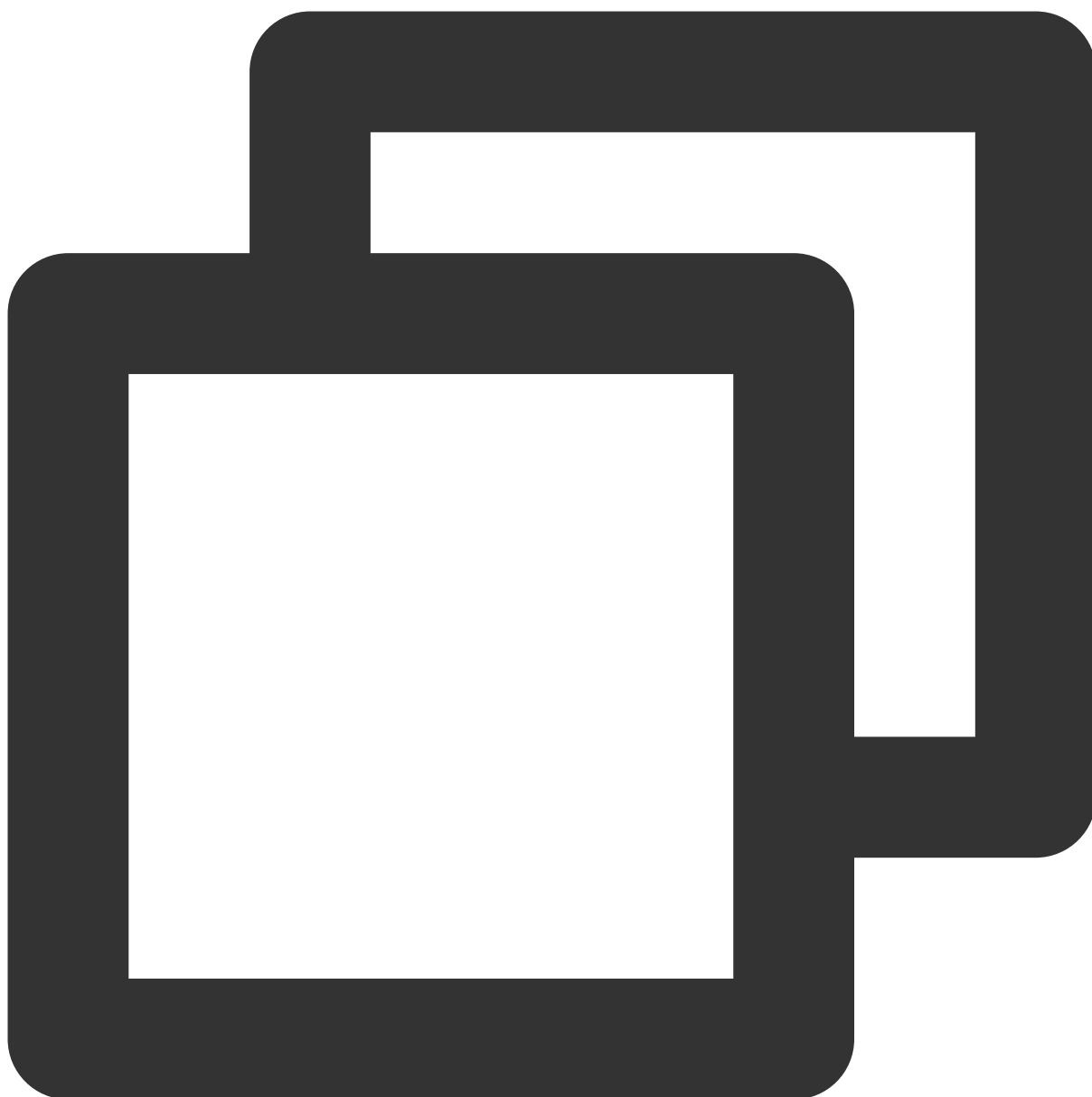


```
public class CKafkaConfigurer {
    private static Properties properties;
    public static void configureSaslPlain() {
        //如果用 -D 或者其它方式设置过，这里不再设置。
        if (null == System.getProperty("java.security.auth.login.config")) {
            //请注意将 xxx 修改为自己的路径。
            System.setProperty("java.security.auth.login.config",
                getCKafkaProperties().getProperty("java.security.auth.login.config"));
        }
    }
    public synchronized static Properties getCKafkaProperties() {
```

```
        if (null != properties) {
            return properties;
        }
        //获取配置文件 kafka.properties 的内容。
        Properties kafkaProperties = new Properties();
        try {
            kafkaProperties.load(CKafkaProducerDemo.class.getClassLoader().getResourceAsStream("kafka.properties"));
        } catch (Exception e) {
            System.out.println("getCKafkaProperties error");
        }
        properties = kafkaProperties;
        return kafkaProperties;
    }
}
```

步骤3：发送消息

1. 创建发送消息程序 `KafkaSaslProducerDemo.java`。

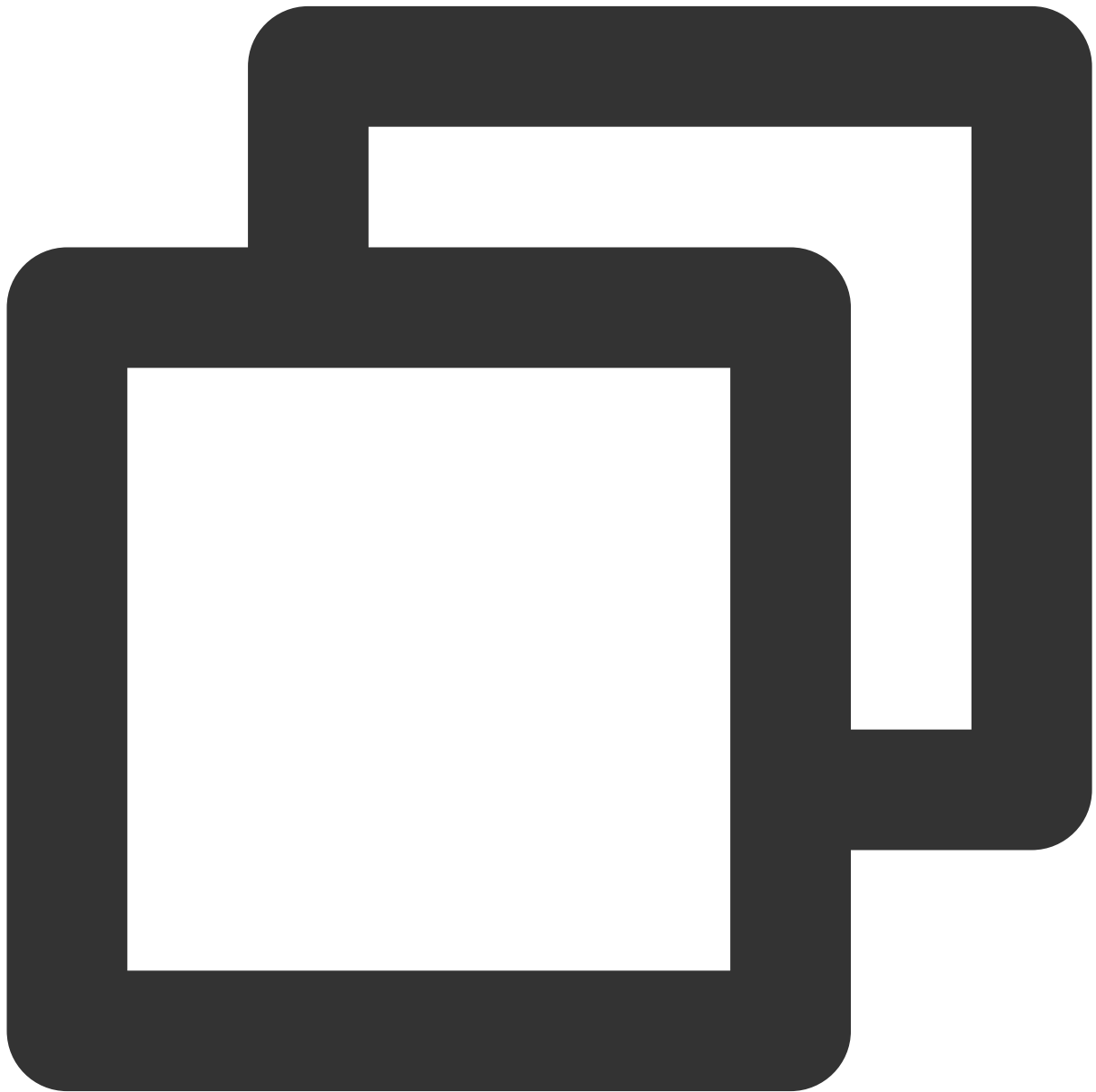


```
public class KafkaSaslProducerDemo {  
    public static void main(String[] args) {  
        //设置 JAAS 配置文件的路径。  
        CKafkaConfigurer.configureSaslPlain();  
        //加载 kafka.properties。  
        Properties kafkaProperties = CKafkaConfigurer.getCKafkaProperties();  
        Properties props = new Properties();  
        //设置接入点，请通过控制台获取对应 Topic 的接入点。  
        props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,  
            kafkaProperties.getProperty("bootstrap.servers"));  
        //  
    }  
}
```

```
// SASL_SSL 公网接入
//
// 接入协议。
props.put(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG, "SASL_SSL");
// SASL 采用 Plain 方式。
props.put(SaslConfigs.SASL_MECHANISM, "PLAIN");
// SSL 加密。
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
//消息队列 Kafka 版消息的序列化方式。
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");
//请求的最长等待时间。
props.put(ProducerConfig.MAX_BLOCK_MS_CONFIG, 30 * 1000);
//设置客户端内部重试次数。
props.put(ProducerConfig.RETRIES_CONFIG, 5);
//设置客户端内部重试间隔。
props.put(ProducerConfig.RECONNECT_BACKOFF_MS_CONFIG, 3000);
//ack=0 producer 将不会等待来自 broker 的确认, 重试配置不会生效。注意如果被限流了后,
//ack=1 broker leader 将不会等待所有 broker follower 的确认, 就返回 ack。
//ack=all broker leader 将等待所有 broker follower 的确认, 才返回 ack。
props.put(ProducerConfig.ACKS_CONFIG, "all");
//构造 Producer 对象, 注意, 该对象是线程安全的, 一般来说, 一个进程内一个Producer对象即可。
KafkaProducer<String, String> producer = new KafkaProducer<>(props);
//构造一个消息队列 Kafka 版消息。
String topic = kafkaProperties.getProperty("topic"); //消息所属的Topic, 请在控制
String value = "this is ckafka msg value"; //消息的内容。
try {
    //批量获取 Future 对象可以加快速度。但注意, 批量不要太大。
    List<Future<RecordMetadata>> futures = new ArrayList<>(128);
    for (int i = 0; i < 100; i++) {
        //发送消息, 并获得一个Future对象。
        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) {
        //同步获得 Future 对象的结果。
        RecordMetadata recordMetadata = future.get();
        System.out.println("Produce ok:" + recordMetadata.toString());
    }
} catch (Exception e) {
```

```
//客户端内部重试之后，仍然发送失败，业务要应对此类错误。  
System.out.println("error occurred");  
    }  
}  
}
```

2. 编译并运行 `KafkaSaslProducerDemo.java` 发送消息。
3. 运行结果（输出）。



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

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

4. 在 CKafka 控制台 **topic 管理** 页面，选择对应的 Topic，单击**更多 > 消息查询**，查看刚刚发送的消息。

消息查询

消息查询会占用CKafka实例的带宽资源，建议您尽量缩小查询范围查询，不要频繁操作。
消息查询最多展示最近的20条消息。

实例

Topic

查询类型

分区ID

时间

查询

按位点查询

按时间查询

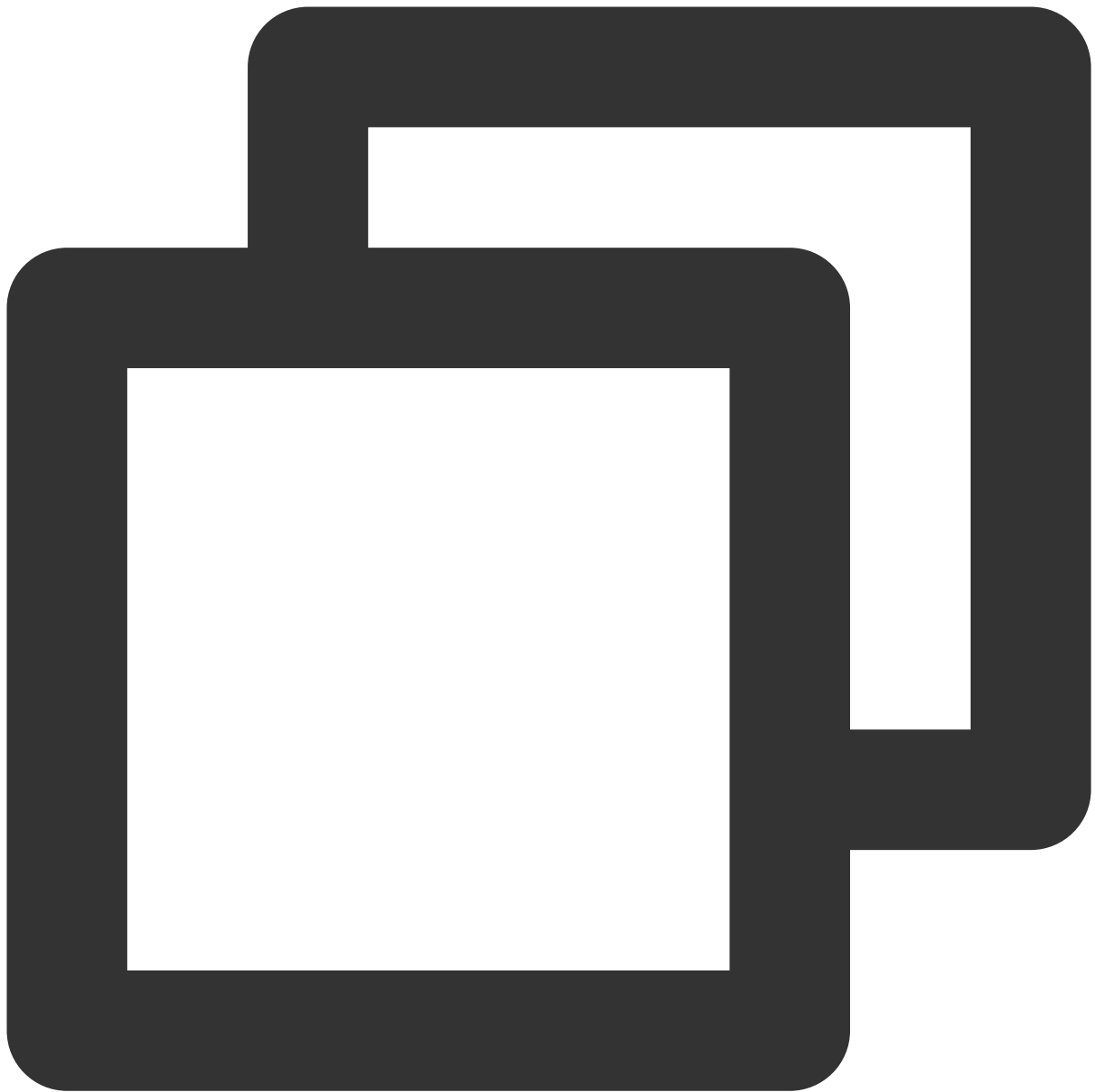
0

2017-22:54

分区ID	位点	时间戳
0	628	2017-25:31
0	629	2017-25:31

步骤4：消费消息

1. 创建 Consumer 订阅消息程序 `KafkaSaslConsumerDemo.java`。



```
public class KafkaSaslConsumerDemo {  
    public static void main(String[] args) {  
        //设置JAAS配置文件的路径。  
        CKafkaConfigurer.configureSaslPlain();  
        //加载kafka.properties。  
        Properties kafkaProperties = CKafkaConfigurer.getCKafkaProperties();  
        Properties props = new Properties();  
        //设置接入点， 请通过控制台获取对应Topic的接入点。  
        props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,  
            kafkaProperties.getProperty("bootstrap.servers"));  
        //  
    }  
}
```

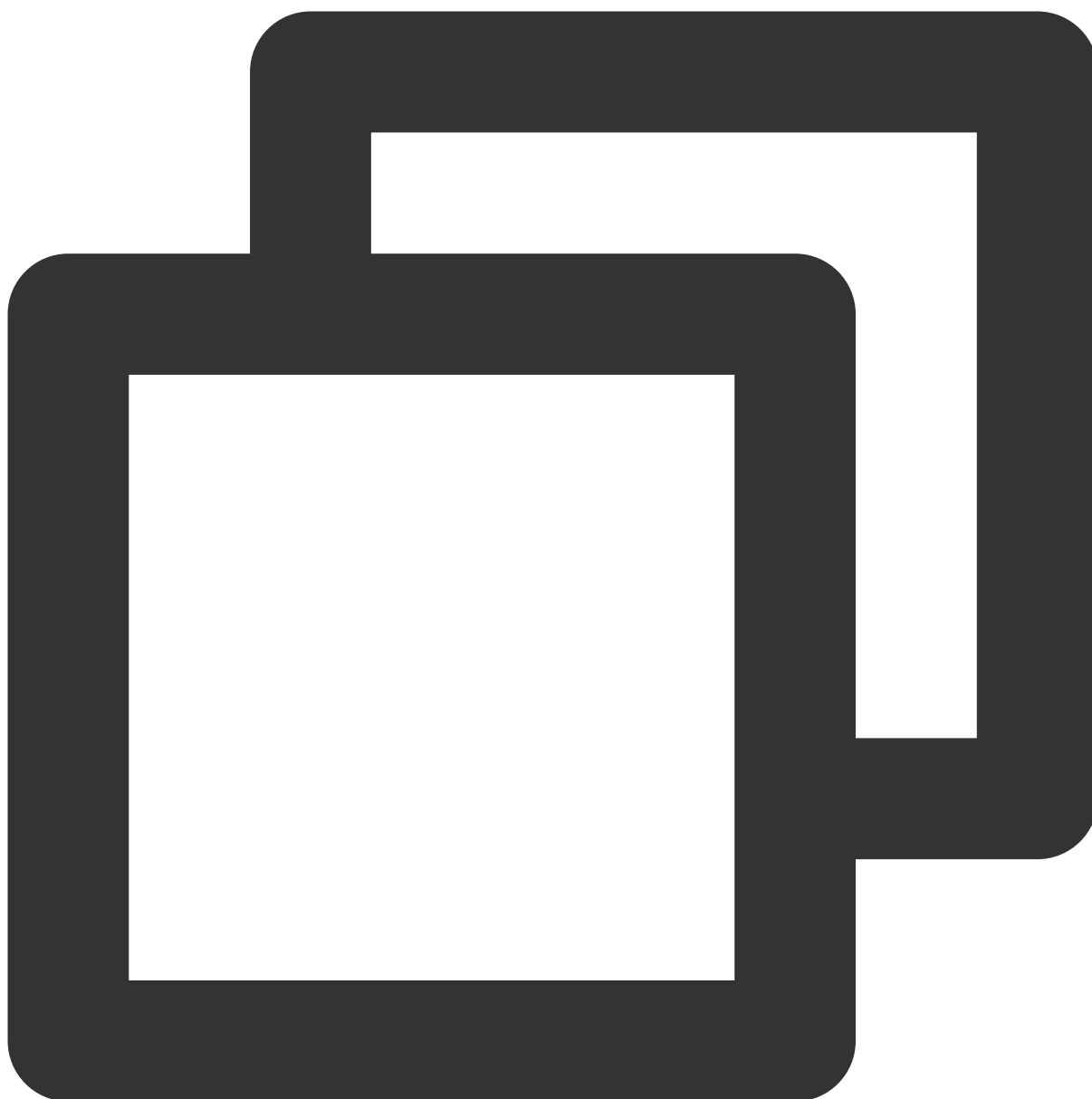
```
// SASL_SSL 公网接入
//
// 接入协议。
props.put(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG, "SASL_SSL");
// SASL 采用 Plain 方式。
props.put(SaslConfigs.SASL_MECHANISM, "PLAIN");
// SSL 加密。
props.put(SslConfigs.SSL_TRUSTSTORE_LOCATION_CONFIG, kafkaProperties.getPrope
props.put(SslConfigs.SSL_TRUSTSTORE_PASSWORD_CONFIG, kafkaProperties.getPrope
props.put(SslConfigs.SSL_ENDPOINT_IDENTIFICATION_ALGORITHM_CONFIG, kafkaProper
//消费者超时时长
//消费者超过该值没有返回心跳，服务端判断消费者处于非存活状态，服务端将消费者从 Consumer Gr
props.put(ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG, 30000);
//两次poll的最长时间间隔
//0.10.1.0 版本前这2个概念是混合的，都用session.timeout.ms表示
props.put(ConsumerConfig.MAX_POLL_INTERVAL_MS_CONFIG, 30000);
//每次 Poll 的最大数量。
//注意该值不要改得太大，如果 Poll 太多数据，而不能在下次 Poll 之前消费完，则会触发一次负载
props.put(ConsumerConfig.MAX_POLL_RECORDS_CONFIG, 30);
//消息的反序列化方式。
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");
//当前消费实例所属的消费组，请在控制台申请之后填写。
//属于同一个组的消费实例，会负载消费消息。
props.put(ConsumerConfig.GROUP_ID_CONFIG, kafkaProperties.getProperty("group.
//构造消费对象，也即生成一个消费实例。
KafkaConsumer<String, String> consumer = new KafkaConsumer<String, String>(pr
//设置消费组订阅的 Topic，可以订阅多个。
//如果 GROUP_ID_CONFIG 是一样，则订阅的 Topic 也建议设置成一样。
List<String> subscribedTopics = new ArrayList<String>();
//如果需要订阅多个 Topic，则在这里添加进去即可。
//每个 Topic 需要先在控制台进行创建。
String topicStr = kafkaProperties.getProperty("topic");
String[] topics = topicStr.split(",");
for (String topic : topics) {
    subscribedTopics.add(topic.trim());
}
consumer.subscribe(subscribedTopics);
//循环消费消息。
while (true) {
    try {
        ConsumerRecords<String, String> records = consumer.poll(1000);
        //必须在下次 Poll 之前消费完这些数据，且总耗时不得超过 SESSION_TIMEOUT_MS_CONF
        for (ConsumerRecord<String, String> record : records) {
            System.out.println(
```



```
                String.format("Consume partition:%d offset:%d", record.partition(),
                                record.offset()));
            }
        } catch (Exception e) {
            System.out.println("consumer error!");
        }
    }
}
```

2. 编译并运行 KafkaSaslConsumerDemo.java 消费消息。

3. 运行结果。



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

4. 在 CKafka 控制台 **Consumer Group** 页面，选择对应的消费组名称，在主题名称输入 Topic 名称，单击**查询详情**，查看消费详情。



Python SDK

VPC 网络接入

最近更新时间：2024-05-31 14:40:49

操作场景

该任务以 Python 客户端为例指导您使用 VPC 网络接入消息队列 CKafka 版并收发消息。

前提条件

[安装 Python](#)

[安装 pip](#)

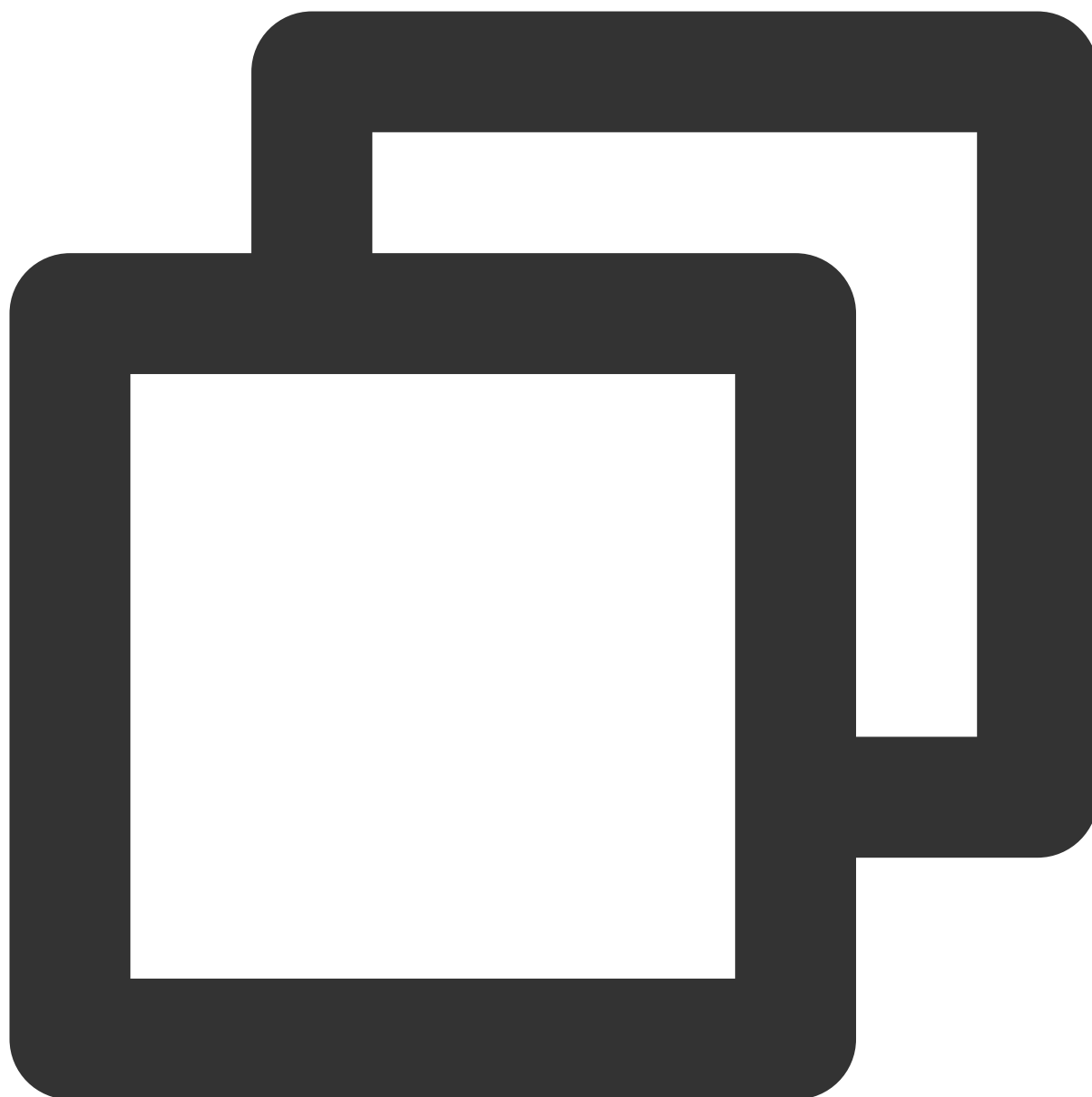
[下载 Demo](#)

操作步骤

将下载的 Demo 中的 pythonkafkademo 上传至 Linux 服务器，登录 Linux 服务器，进入 pythonkafkademo 目录。

步骤1：添加 Python 依赖库

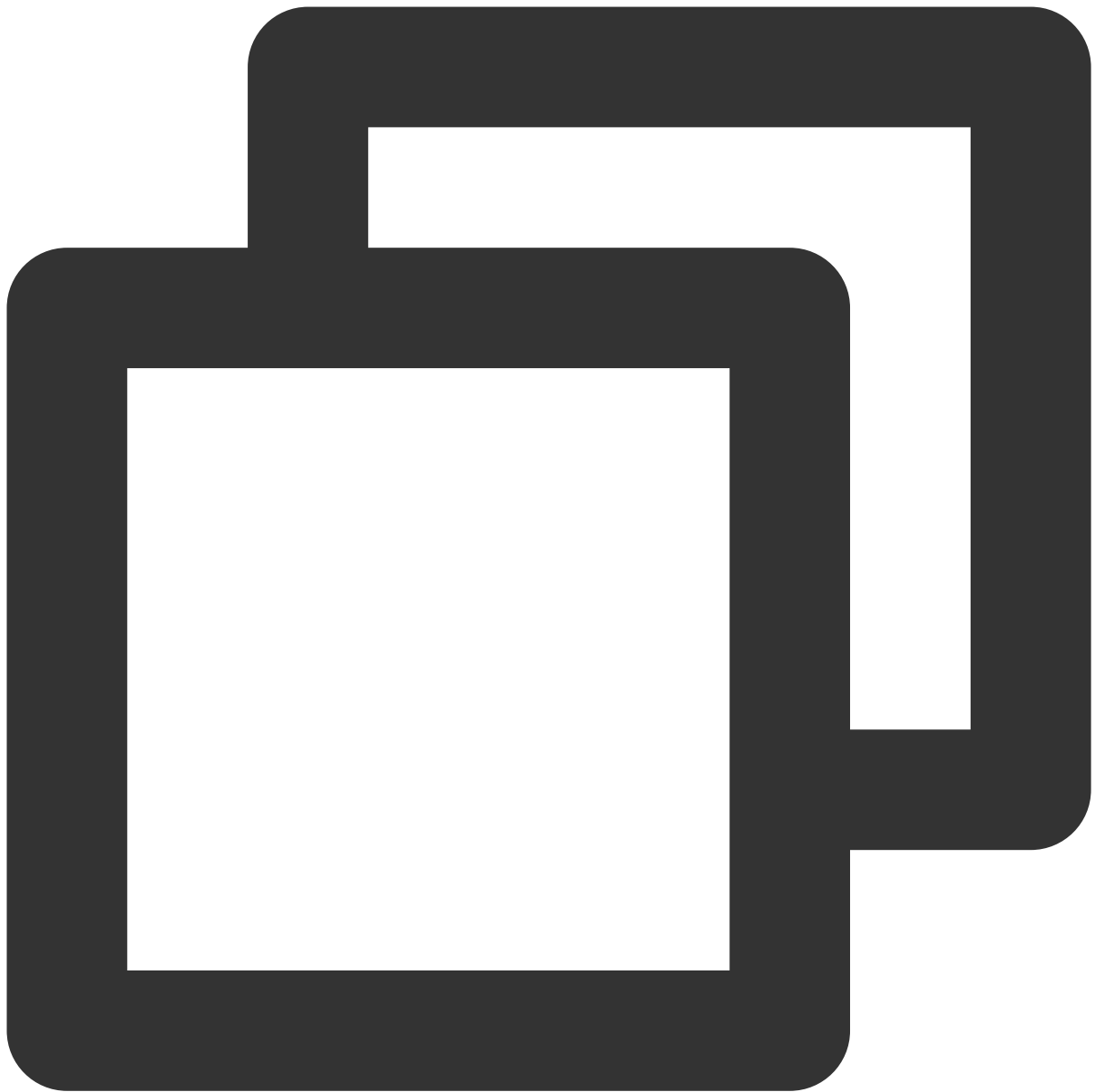
执行以下命令安装：



```
pip install kafka-python
```

步骤2：生产消息

1. 修改生产消息程序 `producer.py` 中配置参数。



```
#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.")
```

<pre>producer.close()</pre>																					
参数	描述																				
bootstrap_servers	接入网络，在控制台的实例详情页面接入方式模块的网络列复制。 接入方式? 添加路由策略 <table><tr><th>接入类型</th><th>接入方式</th><th>网络</th><th>操作</th></tr><tr><td>VPC网络</td><td>PLAINTEXT</td><td> vpc-l subnet-5gl6yj86 1.....0092 删除 查看所有IP和端口 </td></tr></table>	接入类型	接入方式	网络	操作	VPC网络	PLAINTEXT	vpc-l subnet-5gl6yj86 1.....0092 删除 查看所有IP和端口													
接入类型	接入方式	网络	操作																		
VPC网络	PLAINTEXT	vpc-l subnet-5gl6yj86 1.....0092 删除 查看所有IP和端口																			
topic_name	Topic 名称，您可以在控制台上 topic 管理页面复制。 ckafka-plain 基本信息topic管理Consumer Group监控事件中心HTTP接入ACL策略管理智能运维专业版集群备份专业版 新建24/450 请输入TopicId或名称 <table><tr><th>ID/名称</th><th>监控</th><th>分区数(个)</th><th>副本数(个)</th><th>标签</th><th>备注</th><th>创建时间</th><th>消息保留时间</th><th>状态</th><th>操作</th></tr><tr><td> topic-j cocs </td><td></td><td>3</td><td>2</td><td></td><td></td><td>2023-03-25 23:03:00</td><td>2 天</td><td>正常</td><td> 编辑删除 更多 </td></tr></table>	ID/名称	监控	分区数(个)	副本数(个)	标签	备注	创建时间	消息保留时间	状态	操作	topic-j cocs		3	2			2023-03-25 23:03:00	2 天	正常	编辑删除 更多
ID/名称	监控	分区数(个)	副本数(个)	标签	备注	创建时间	消息保留时间	状态	操作												
topic-j cocs		3	2			2023-03-25 23:03:00	2 天	正常	编辑删除 更多												

2. 编译并运行 `producer.py` 。
3. 查看运行结果。

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

4. 在 [CKafka 控制台](#) 的 **topic 管理**页面，选择对应的 Topic，单击**更多**>**消息查询**，查看刚刚发送的消息。

消息查询

广州

消息查询会占用CKafka实例的带宽资源，建议您尽量缩小查询范围，不要频繁操作。

消息查询最多展示指定位点或时间点后的20条数据。

实例

ckafka-【勿删-不要#

Topic

cocs

查询类型

按位点查询

按起始时间查询

分区ID

所有分区

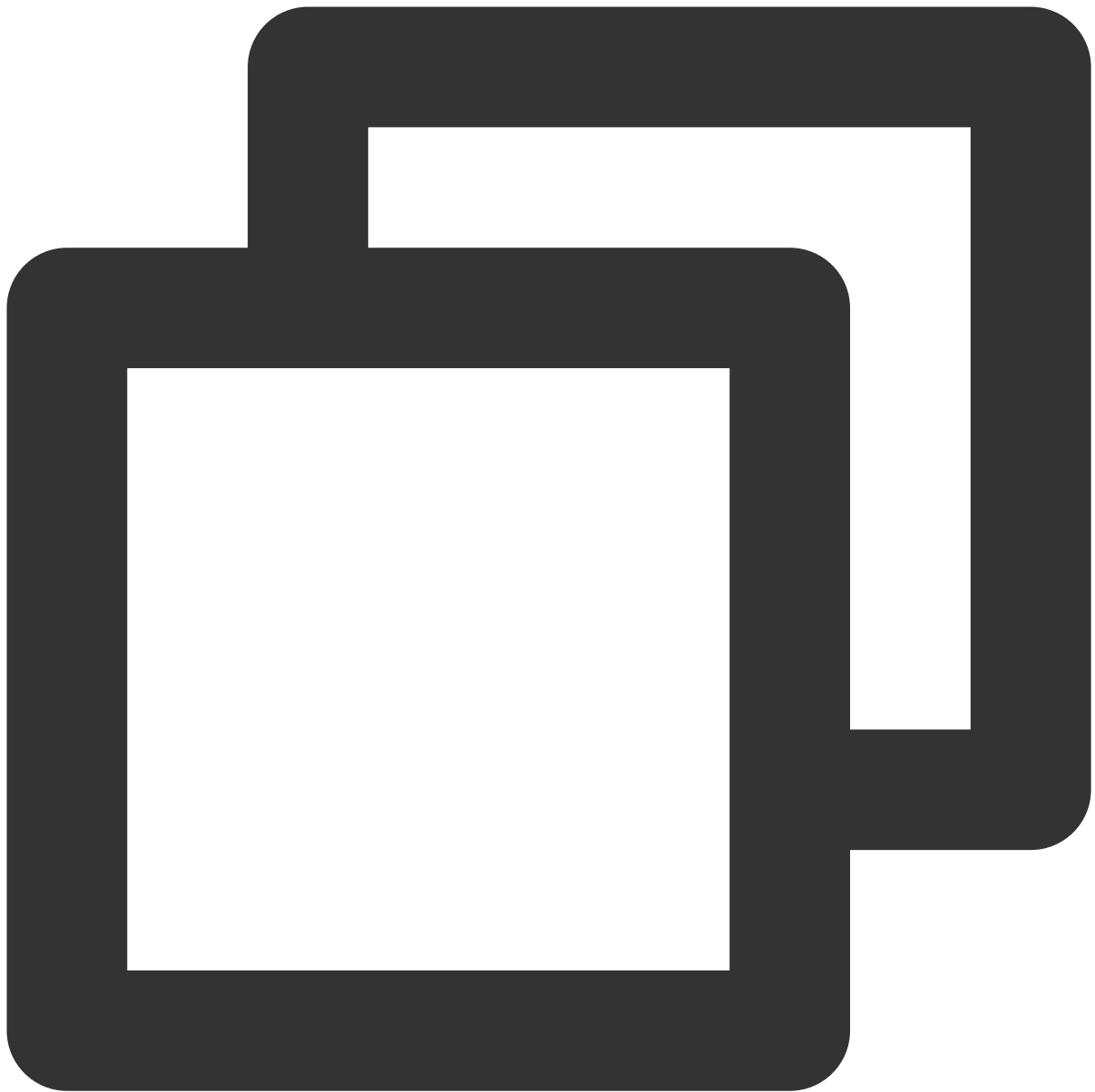
时间

2023-03-25 11:17:05

查询

步骤3：消费消息

1. 修改消费消息程序 consumer.py 中配置参数。



```
#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
```

参数	描述																				
bootstrap_servers	接入网络，在控制台的实例详情页面接入方式模块的网络列复制。 接入方式? 添加路由策略 <table><tr><th>接入类型</th><th>接入方式</th><th>网络</th><th>操作</th></tr><tr><td>VPC网络</td><td>PLAINTEXT</td><td>vpc-l subnet-5gl6yj86 10.0.0.0/24 10.0.0.0/24</td><td>删除 查看所有IP和端口</td></tr></table>	接入类型	接入方式	网络	操作	VPC网络	PLAINTEXT	vpc-l subnet-5gl6yj86 10.0.0.0/24 10.0.0.0/24	删除 查看所有IP和端口												
接入类型	接入方式	网络	操作																		
VPC网络	PLAINTEXT	vpc-l subnet-5gl6yj86 10.0.0.0/24 10.0.0.0/24	删除 查看所有IP和端口																		
group_id	消费者的组 ID，根据业务需求自定义																				
topic_name	Topic 名称，您可以在控制台上 topic 管理页面复制。 ckafka-1 topic管理Consumer Group监控事件中心HTTP接入ACL策略管理智能运维专业版集群备份专业版 新建(24/450) 请输入TopicId或名称 <table><tr><th>ID/名称</th><th>监控</th><th>分区数(个)</th><th>副本数(个)</th><th>标签</th><th>备注</th><th>创建时间</th><th>消息保留时间</th><th>状态</th><th>操作</th></tr><tr><td>topic-l coocs</td><td></td><td>3</td><td>2</td><td></td><td></td><td>2023-03-25 23:03:00</td><td>2 天</td><td>正常</td><td>编辑删除更多</td></tr></table>	ID/名称	监控	分区数(个)	副本数(个)	标签	备注	创建时间	消息保留时间	状态	操作	topic-l coocs		3	2			2023-03-25 23:03:00	2 天	正常	编辑删除更多
ID/名称	监控	分区数(个)	副本数(个)	标签	备注	创建时间	消息保留时间	状态	操作												
topic-l coocs		3	2			2023-03-25 23:03:00	2 天	正常	编辑删除更多												

2. 编译并运行 `consumer.py` 。
3. 查看运行结果。

```
[root@VM-8-16-centos sas1]# python3 consumer.py
Topic:[test] Partition:[0] Offset:[2011] Value:[b'"Hello World! Hello C
```

4. 在 **CKafka 控制台** 的 **Consumer Group** 页面，选择对应的消费组名称，在主题名称输入 Topic 名称，单击**查询详情**，查看消费详情。

消息查询

广州

消息查询会占用CKafka实例的带宽资源，建议您尽量缩小查询范围，不要频繁操作。

消息查询最多展示指定位点或时间点后的20条数据。

实例

ckafka-【勿删-不要开】

Topic

cocs

查询类型

按位点查询

按起始时间查询

分区ID

所有分区

时间

2023-03-25 11:17:05

查询

公网 SASL_PLAINTEXT 方式接入

最近更新时间：2024-05-31 14:40:49

操作场景

该任务以 Python 客户端为例，指导您使用公网 SASL_PLAINTEXT 方式接入消息队列 CKafka 版并收发消息。

前提条件

[安装 Python](#)

[安装 pip](#)

[配置 ACL 策略](#)

[下载 Demo](#)

操作步骤

步骤1：准备工作

1. 创建接入点。

1.1 在 [实例列表](#) 页面，单击目标实例 ID，进入实例详情页。

1.2 在 **基本信息 > 接入方式** 中，单击**添加路由策略**，在打开窗口中选择：**路由类型：公网域名接入**，**接入方式：**
SASL_PLAINTEXT。

添加路由策略

① 当前broker版本不支持SASL_SCRAM认证方式，请升级实例内核小版本。

路由类型 公网域名接入

接入方式 SASL_PLAINTEXT

该接入方式提供用户管理和ACL策略配置，以管理用户访问权限

公网计费模式 按小时计费

公网带宽 3Mbps

CKafka默认赠送3Mbps公网带宽

提交

关闭

2. 创建角色。

在 **ACL 策略管理**下的**用户管理**页面新建角色，设置密码。

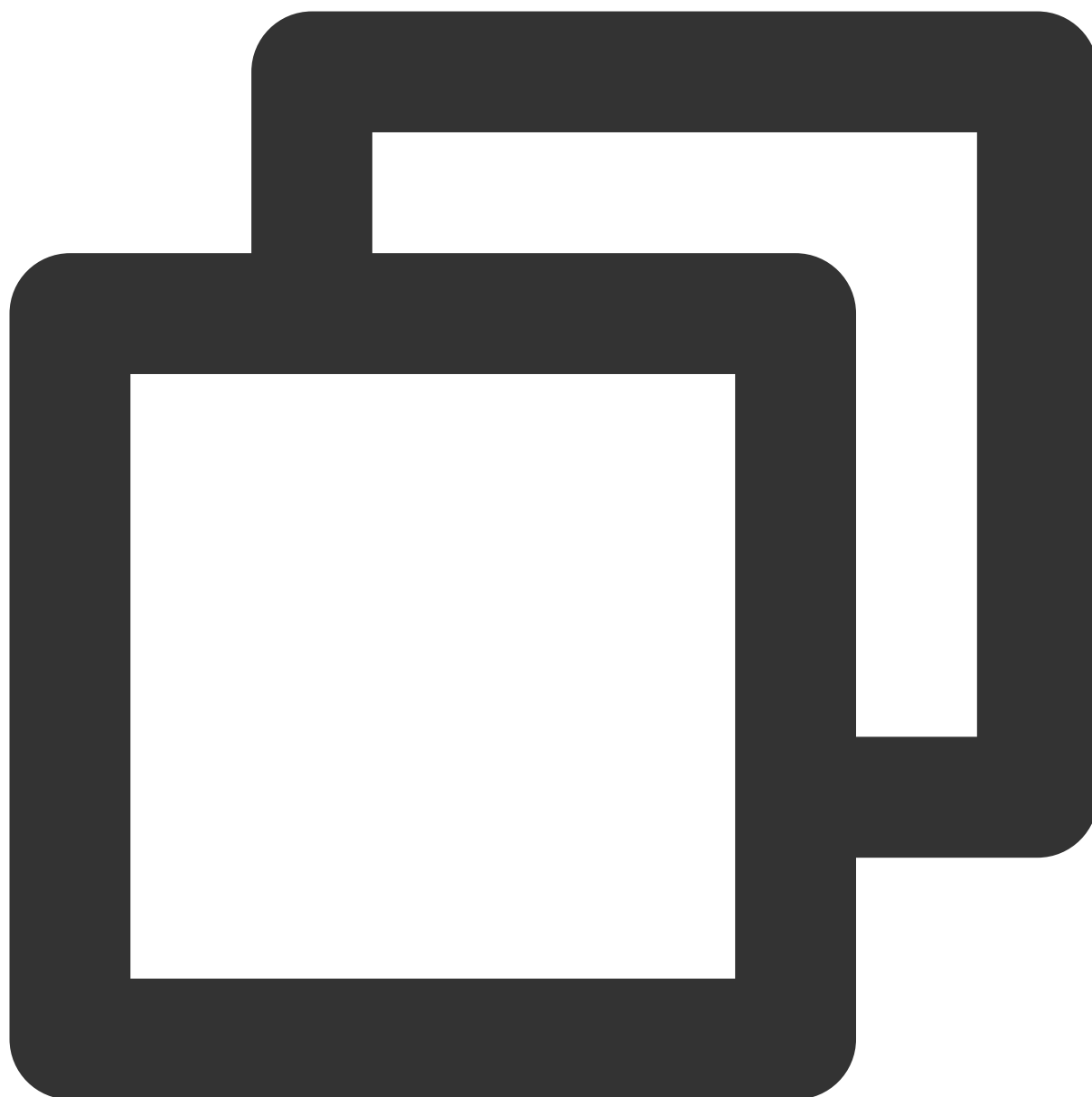


3. 创建 Topic。

在控制台 **topic 管理**页面新建 Topic（参见 [创建 Topic](#)）。

4. 添加 Python 依赖库。

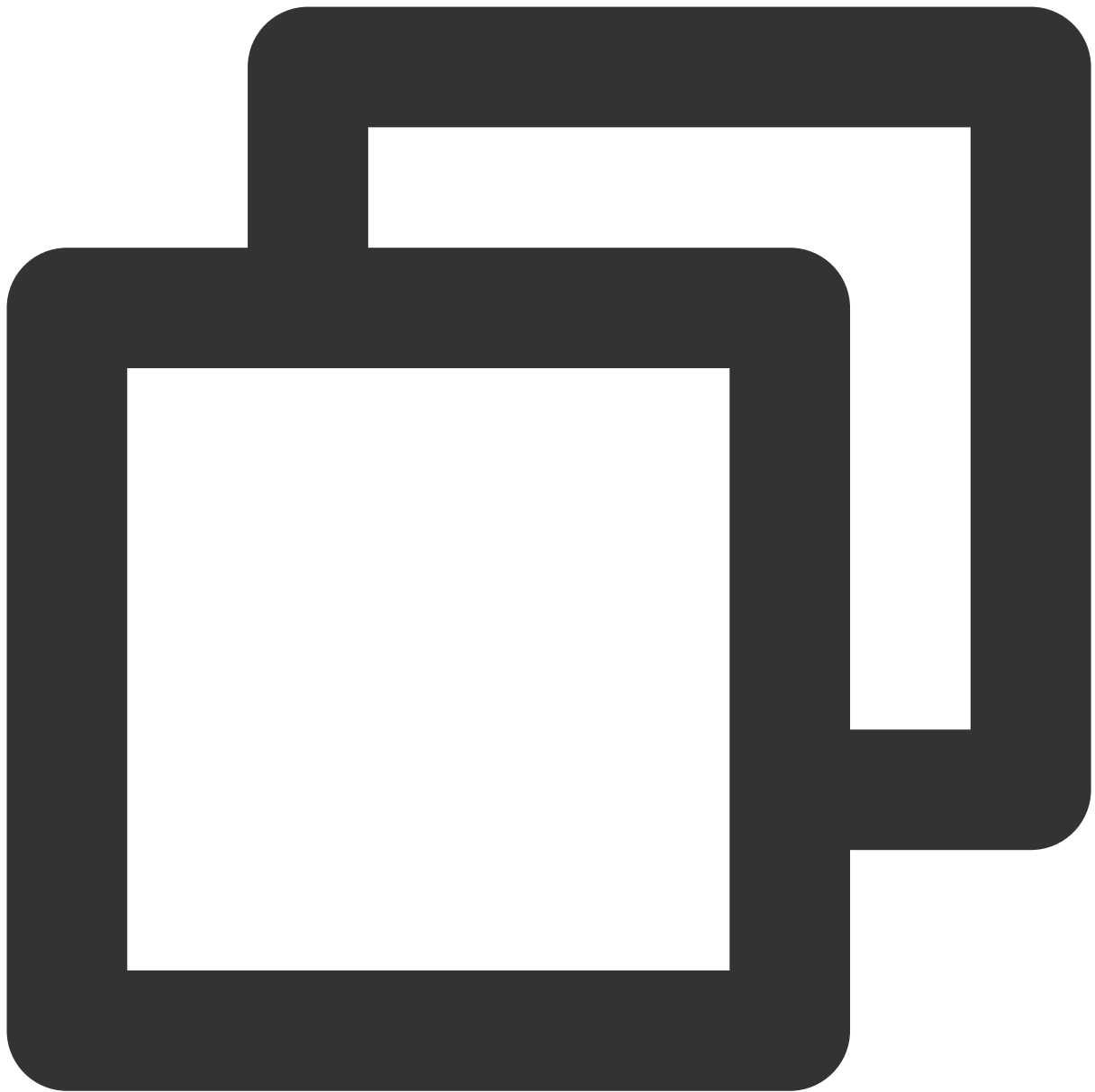
执行以下命令安装：



```
pip install kafka-python
```

步骤2：生产消息

1. 修改生产消息程序 `producer.py` 中配置参数。



```
producer = KafkaProducer(  
    bootstrap_servers = ['xx.xx.xx.xx:port'],  
    api_version = (1, 1),  
  
    #  
    # SASL_PLAINTEXT 公网接入  
    #  
    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()
```

参数	描述																		
<code>bootstrap_servers</code>	接入网络，在控制台的实例详情页面接入方式模块的网络列复制。 接入方式⑦ <table><tr><th>接入类型</th><th>接入方式</th><th>网络</th><th>操作</th></tr><tr><td>公网域名接入</td><td>SASL_SSL</td><td>ckafka-gnzmqj... 159.75.176.66:50001 内</td><td>删除 查看所有</td></tr></table>	接入类型	接入方式	网络	操作	公网域名接入	SASL_SSL	ckafka-gnzmqj... 159.75.176.66:50001 内	删除 查看所有										
接入类型	接入方式	网络	操作																
公网域名接入	SASL_SSL	ckafka-gnzmqj... 159.75.176.66:50001 内	删除 查看所有																
<code>sasl_plain_username</code>	用户名，格式为 <code>实例 ID + # + 用户名</code> 。实例 ID 在 CKafka 控制台 的实例获取，用户在 ACL策略管理 下的 用户管理 创建用户时设置。																		
<code>sasl_plain_password</code>	用户密码，在 CKafka 控制台实例详情页面 ACL策略管理 下的 用户管理 创建用户时																		
<code>topic_name</code>	Topic 名称，您可以在控制台上topic管理页面复制。 ckafka-plain... topic管理 Consumer Group 监控 事件中心 HTTP接入 ACL策略管理 智能运维 专业版 集群备份 专业版 新建(24/450) <table><tr><th>ID/名称</th><th>监控</th><th>分区数(个)</th><th>副本数(个)</th><th>标签</th><th>备注</th><th>创建时间</th><th>消息保留时间</th><th>状态</th></tr><tr><td>topic-j cocs 内</td><td></td><td>3</td><td>2</td><td></td><td></td><td>2023-03-25 23:03:00</td><td>2 天</td><td>正常</td></tr></table>	ID/名称	监控	分区数(个)	副本数(个)	标签	备注	创建时间	消息保留时间	状态	topic-j cocs 内		3	2			2023-03-25 23:03:00	2 天	正常
ID/名称	监控	分区数(个)	副本数(个)	标签	备注	创建时间	消息保留时间	状态											
topic-j cocs 内		3	2			2023-03-25 23:03:00	2 天	正常											

- 2. 编译并运行 `producer.py`。
- 3. 查看运行结果。

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

- 4. 在 [CKafka 控制台](#) 的 **topic管理** 页面，选择对应的 Topic，单击**更多**>**消息查询**，查看刚刚发送的消息。

消息查询

广州

消息查询会占用CKafka实例的带宽资源，建议您尽量缩小查询范围，不要频繁操作。
消息查询最多展示指定位点或时间点后的20条数据。

实例

ckafka-1-janhuai

Topic

test1

查询类型

按位点查询

按起始时间查询

分区ID

0

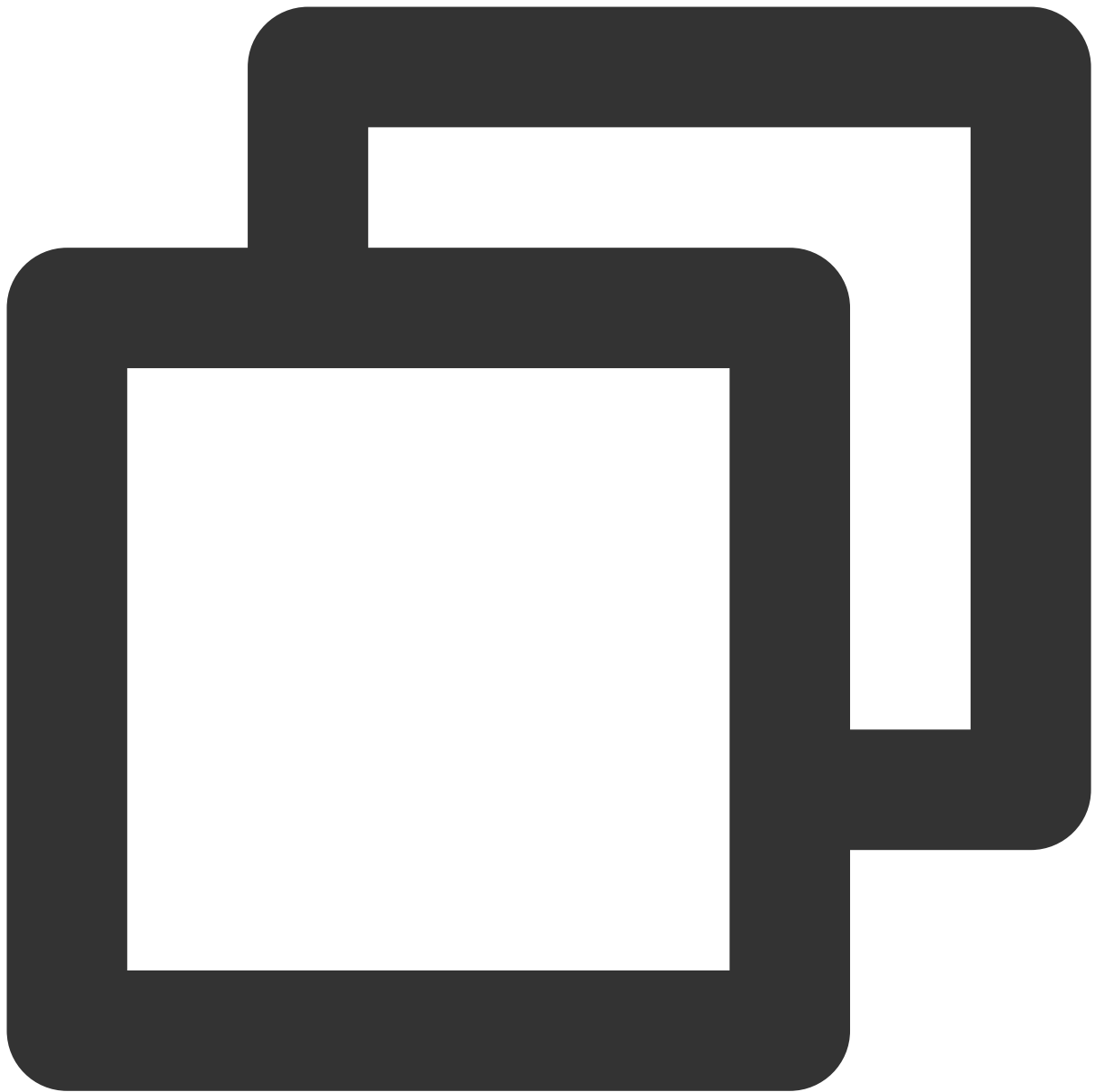
起始位点

请输入起始位点

查询

步骤3：消费消息

1. 修改消费消息程序 consumer.py 中配置参数。



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

参数	描述																		
<code>bootstrap_servers</code>	接入网络，在控制台的实例详情页面接入方式模块的网络列复制。 接入方式⑦ <table><tr><th>接入类型</th><th>接入方式</th><th>网络</th><th>操作</th></tr><tr><td>公网域名接入</td><td>SASL_SSL</td><td>ckafka-gnzmqj... 159.75.176.66:50001</td><td>删除 查看所有</td></tr></table>	接入类型	接入方式	网络	操作	公网域名接入	SASL_SSL	ckafka-gnzmqj... 159.75.176.66:50001	删除 查看所有										
接入类型	接入方式	网络	操作																
公网域名接入	SASL_SSL	ckafka-gnzmqj... 159.75.176.66:50001	删除 查看所有																
<code>group_id</code>	消费者的组 ID，根据业务需求自定义。																		
<code>sasl_plain_username</code>	用户名，格式为 <code>实例 ID + # + 用户名</code> 。实例 ID 在CKafka 控制台的实例获取，用户在 ACL 策略管理 下的 用户管理 创建用户时设置。																		
<code>sasl_plain_password</code>	用户名密码，在 CKafka 控制台实例详情页面 ACL 策略管理 下的 用户管理 创建用																		
<code>topic_name</code>	Topic 名称，您可以在控制台上 topic 管理页面复制。 ckafka-pl... topic管理 Consumer Group 监控 事件中心 HTTP接入 ACL策略管理 智能运维 专业版 集群备份 专业版 新建(24/450) <table><tr><th>ID/名称</th><th>监控</th><th>分区数(个)</th><th>副本数(个)</th><th>标签</th><th>备注</th><th>创建时间</th><th>消息保留时间</th><th>状态</th></tr><tr><td>topic-j cocs</td><td></td><td>3</td><td>2</td><td></td><td></td><td>2023-03-25 23:03:00</td><td>2 天</td><td>正常</td></tr></table>	ID/名称	监控	分区数(个)	副本数(个)	标签	备注	创建时间	消息保留时间	状态	topic-j cocs		3	2			2023-03-25 23:03:00	2 天	正常
ID/名称	监控	分区数(个)	副本数(个)	标签	备注	创建时间	消息保留时间	状态											
topic-j cocs		3	2			2023-03-25 23:03:00	2 天	正常											

- 2. 编译并运行 `consumer.py` 。
- 3. 查看运行结果。

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

- 4. 在 **CKafka 控制台** 的 **Consumer Group** 页面，选择对应的消费组名称，在主题名称输入 Topic 名称，单击**查看详情**，查看消费详情。

datahub-task-y92dovob详情

请输入关键字

memberID	Client ID	Client Host	topic名称	分
cross 0/11 463b-947d-6947c658ff8b	en- 35-867d-	cross-Region#s our-um er-yr- y9m/quen-0	/ 95 jas 313	pa

共 1 条

请选择条 / 页

我知道了

公网 SASL_SSL 方式接入

最近更新时间：2024-05-31 14:40:49

操作场景

该任务以 Python 客户端为例，指导您使用公网 SASL_SSL 方式接入消息队列 CKafka 版并收发消息。

前提条件

[安装 Python](#)

[安装 pip](#)

[配置 ACL 策略](#)

[下载 Demo](#)

[下载 SASL_SSL 证书](#)

操作步骤

步骤一：准备工作

1. 创建接入点。

1.1 在 [实例列表](#) 页面，单击目标实例 ID，进入实例详情页。

1.2 在 **基本信息** > **接入方式** 中，单击**添加路由策略**，在打开窗口中选择：**路由类型：公网域名接入**，**接入方式：**
SASL_SSL。

添加路由策略

① 当前broker版本不支持SASL_SCRAM认证方式，请升级实例内核小版本。

路由类型 公网域名接入

接入方式

SASL_SSL

[下载证书](#)

公网计费模式 按小时计费

公网带宽

3Mbps

CKafka默认赠送3Mbps公网带宽

提交

关闭

2. 创建角色。

在**用户管理**页面新建角色，设置密码。

基本信息 topic管理 Consumer Group 监控 事件中心 HTTP接入 **ACL策略管理** 智能运维 专业版 集群备份 专业版

策略列表 **用户管理**

新建

请输入名称

用户名	创建/更新时间	操作
ckafka-  -hhj	2023-04-13 18:01:11 2023-04-13 18:01:11	修改密码 删除

共 1 条

20 条 / 页

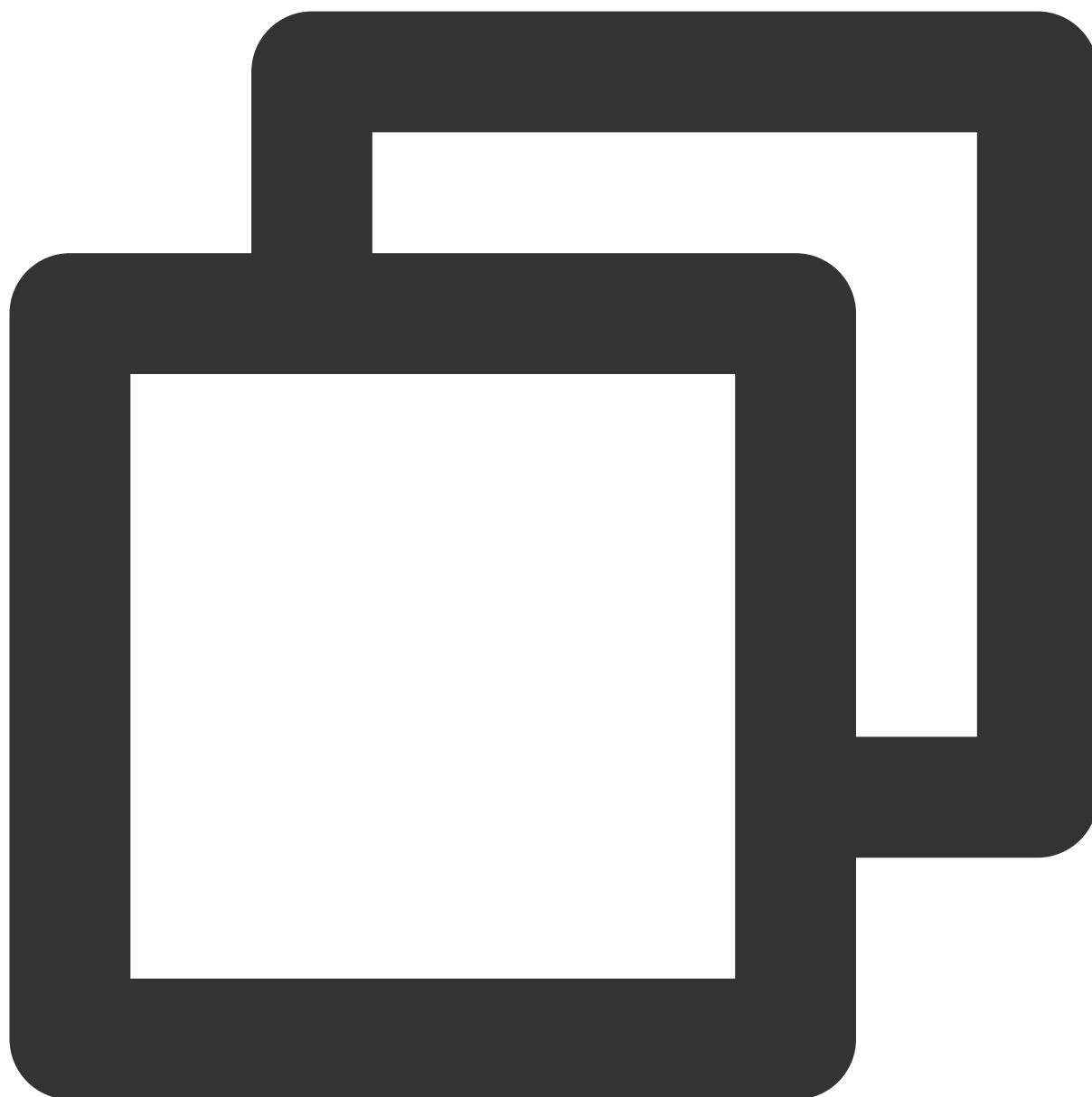
1 / 1 页

3. 创建 Topic。

在控制台 **topic 管理** 页面新建 Topic（参考 [创建 Topic](#)）。

4. 添加 Python 依赖库。

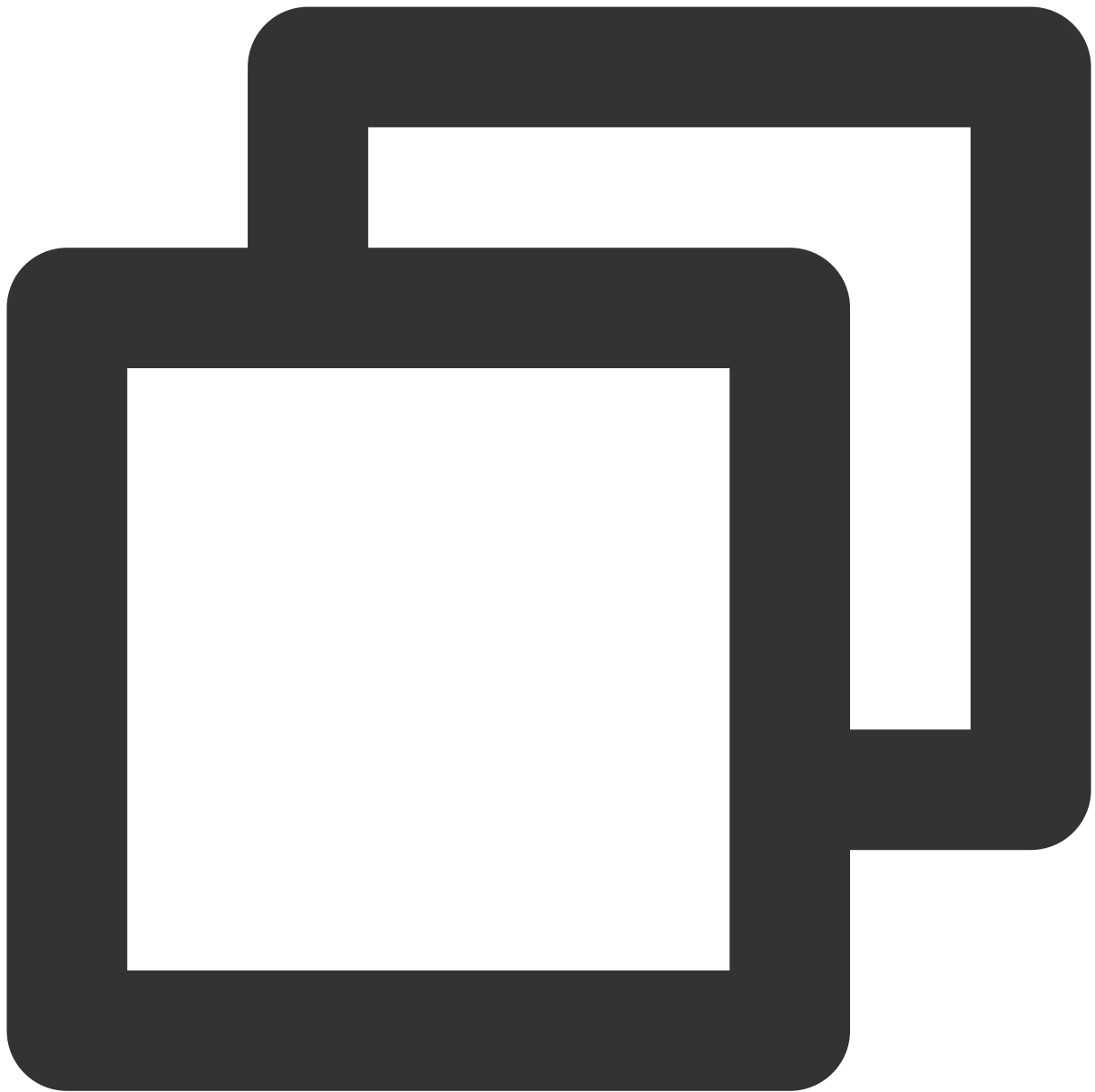
执行以下命令安装：



```
pip install kafka-python
```

步骤二：生产消息

1. 修改生产消息程序 `producer.py` 中配置参数。



```
producer = KafkaProducer(  
    bootstrap_servers = ['xx.xx.xx.xx:port'],  
    api_version = (1, 1),  
  
    #  
    # SASL_SSL 公网接入  
    #  
    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()
```

参数	描述																				
<code>bootstrap_servers</code>	接入网络，在控制台的实例详情页面接入方式模块的网络列复制。 接入方式? 添加路由策略 <table><tr><th>接入类型</th><th>接入方式</th><th>网络</th><th>操作</th></tr><tr><td>公网域名接入</td><td>SASL_PLAINTEXT</td><td>ckafka-l4xppqrz...</td><td>删除</td></tr></table>	接入类型	接入方式	网络	操作	公网域名接入	SASL_PLAINTEXT	ckafka-l4xppqrz...	删除												
接入类型	接入方式	网络	操作																		
公网域名接入	SASL_PLAINTEXT	ckafka-l4xppqrz...	删除																		
<code>sasl_plain_username</code>	用户名，格式为 <code>实例 ID + # + 用户名</code> 。实例 ID 在 CKafka 控制台 的实例基本信息获取，用户在 用户管理 创建用户时设置。																				
<code>sasl_plain_password</code>	用户密码，在 CKafka 控制台 实例详情页面的 用户管理 创建用户时设置。																				
<code>topic_name</code>	Topic 名称，您可以在控制台上 topic 管理页面复制。 ckafka-pl... topic管理Consumer Group监控事件中心HTTP接入ACL策略管理智能运维专业版集群备份专业版 新建(24/450) 请输入TopicId或名称 <table><tr><th>ID/名称</th><th>监控</th><th>分区数(个)</th><th>副本数(个)</th><th>标签</th><th>备注</th><th>创建时间</th><th>消息保留时间</th><th>状态</th><th>操作</th></tr><tr><td>topic-jcoos</td><td>山</td><td>3</td><td>2</td><td></td><td></td><td>2023-03-25 23:03:00</td><td>2 天</td><td>正常</td><td>编辑更多</td></tr></table>	ID/名称	监控	分区数(个)	副本数(个)	标签	备注	创建时间	消息保留时间	状态	操作	topic-jcoos	山	3	2			2023-03-25 23:03:00	2 天	正常	编辑更多
ID/名称	监控	分区数(个)	副本数(个)	标签	备注	创建时间	消息保留时间	状态	操作												
topic-jcoos	山	3	2			2023-03-25 23:03:00	2 天	正常	编辑更多												
<code>CARoot.pem</code>	采用 <code>SASL_SSL</code> 方式接入时，所需的证书路径。																				

- 2. 编译并运行 `producer.py`。
- 3. 查看运行结果。

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

- 4. 在 [CKafka 控制台](#) 的 **topic管理**页面，选择对应的 Topic，单击**更多 > 消息查询**，查看刚刚发送的消息。

实例

ckafka-xxxxxxhuanhua

Topic

test1

查询类型

按位点查询

按起始时间查询

分区ID

0

时间

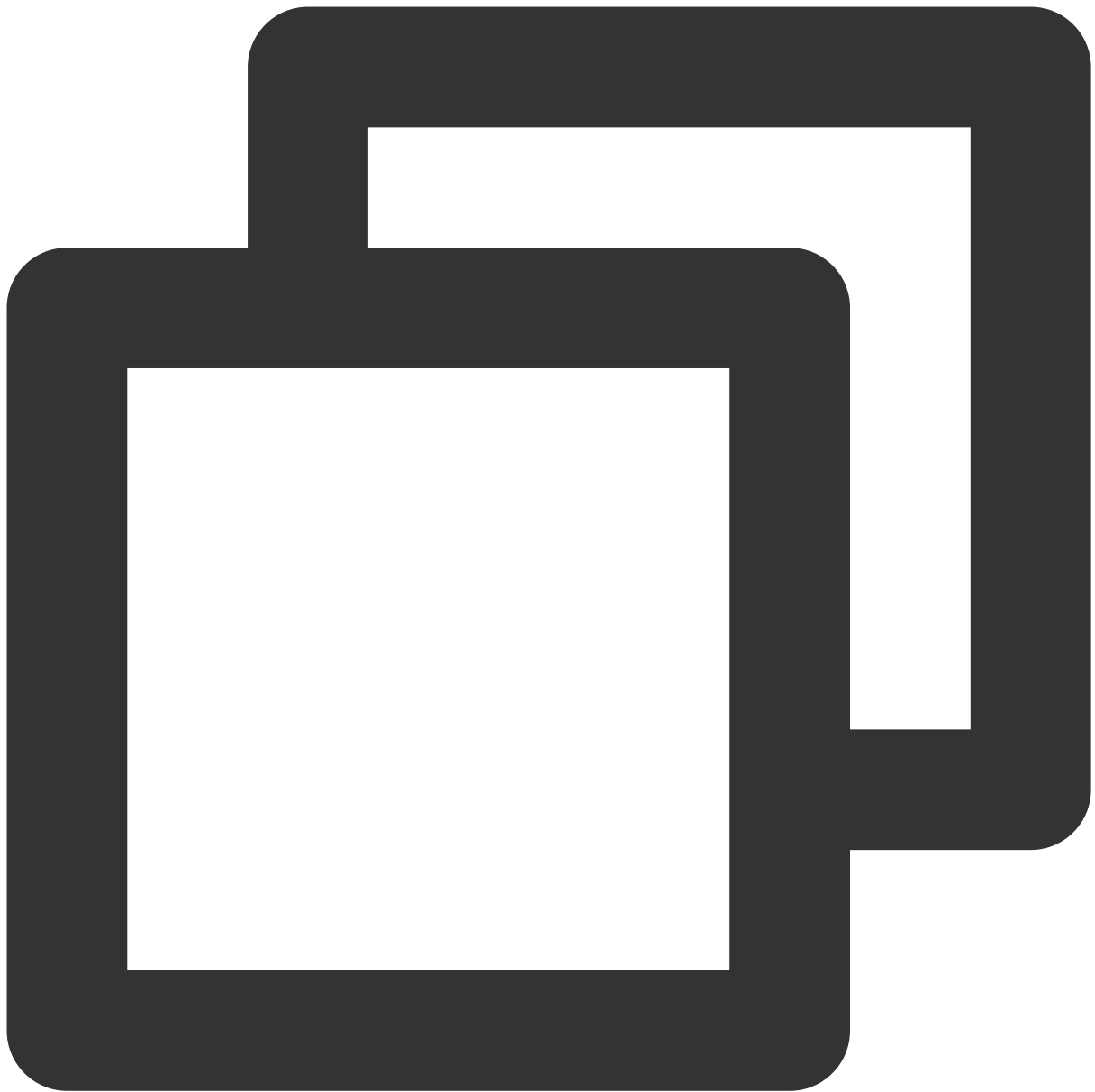
2023-04-13 18:27:17

查询

分区ID	位点	时间
暂无数据		

步骤三：消费消息

1. 修改消费消息程序 consumer.py 中配置参数。



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

参数	描述																		
<code>bootstrap_servers</code>	接入网络，在控制台的实例详情页面接入方式模块的网络列复制。 接入方式① <table><tr><th>接入类型</th><th>接入方式</th><th>网络</th><th>操作</th></tr><tr><td>公网域名接入</td><td>SASL_SSL</td><td>ckafka-gnzmqj... 159.75.176.66:50001</td><td>删除 查看所有IP和</td></tr></table>	接入类型	接入方式	网络	操作	公网域名接入	SASL_SSL	ckafka-gnzmqj... 159.75.176.66:50001	删除 查看所有IP和										
接入类型	接入方式	网络	操作																
公网域名接入	SASL_SSL	ckafka-gnzmqj... 159.75.176.66:50001	删除 查看所有IP和																
<code>group_id</code>	消费者的组 ID，根据业务需求自定义。																		
<code>sasl_plain_username</code>	用户名，格式为 <code>实例 ID + # + 用户名</code> 。实例 ID 在CKafka 控制台的实例息获取，用户在 用户管理 创建用户时设置。																		
<code>sasl_plain_password</code>	用户名密码，在 CKafka 控制台实例详情页面的 用户管理 创建用户时设置																		
<code>topic_name</code>	Topic 名称，您可以在控制台上 topic 管理页面复制。 ckafka-pi... 基本信息 topic管理 Consumer Group 监控 事件中心 HTTP接入 ACL策略管理 智能运维 专业版 集群备份 专业版 新建(24/450) 请输入Topic <table><tr><th>ID/名称</th><th>监控</th><th>分区数(个)</th><th>副本数(个)</th><th>标签</th><th>备注</th><th>创建时间</th><th>消息保留时间</th><th>状态</th></tr><tr><td>topic-1 coos</td><td></td><td>3</td><td>2</td><td></td><td></td><td>2023-03-25 23:03:00</td><td>2 天</td><td>正常</td></tr></table>	ID/名称	监控	分区数(个)	副本数(个)	标签	备注	创建时间	消息保留时间	状态	topic-1 coos		3	2			2023-03-25 23:03:00	2 天	正常
ID/名称	监控	分区数(个)	副本数(个)	标签	备注	创建时间	消息保留时间	状态											
topic-1 coos		3	2			2023-03-25 23:03:00	2 天	正常											
<code>CARoot.pem</code>	采用 <code>SASL_SSL</code> 方式接入时，所需的证书路径。																		

- 2. 编译并运行 consumer.py。
- 3. 查看运行结果。

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

4. 在 [CKafka 控制台](#) 的 **Consumer Group** 页面，选择对应的消费组名称，在主题名称输入 Topic 名称，单击**查询详情**，查看消费详情。

datahub-task-y92dobdn详情

请输入关键字

memberID

Client ID

Client Host

topic名称

分区

/11.168.180.95

jasontest0313

partition-0

共 1 条

请选择 条 / 页

1 / 1 页

我知道了

Go SDK

VPC 网络接入

最近更新时间：2024-05-31 14:40:49

操作场景

该任务以 Go 客户端为例指导您使用 VPC 网络接入消息队列 CKafka 版并收发消息。

前提条件

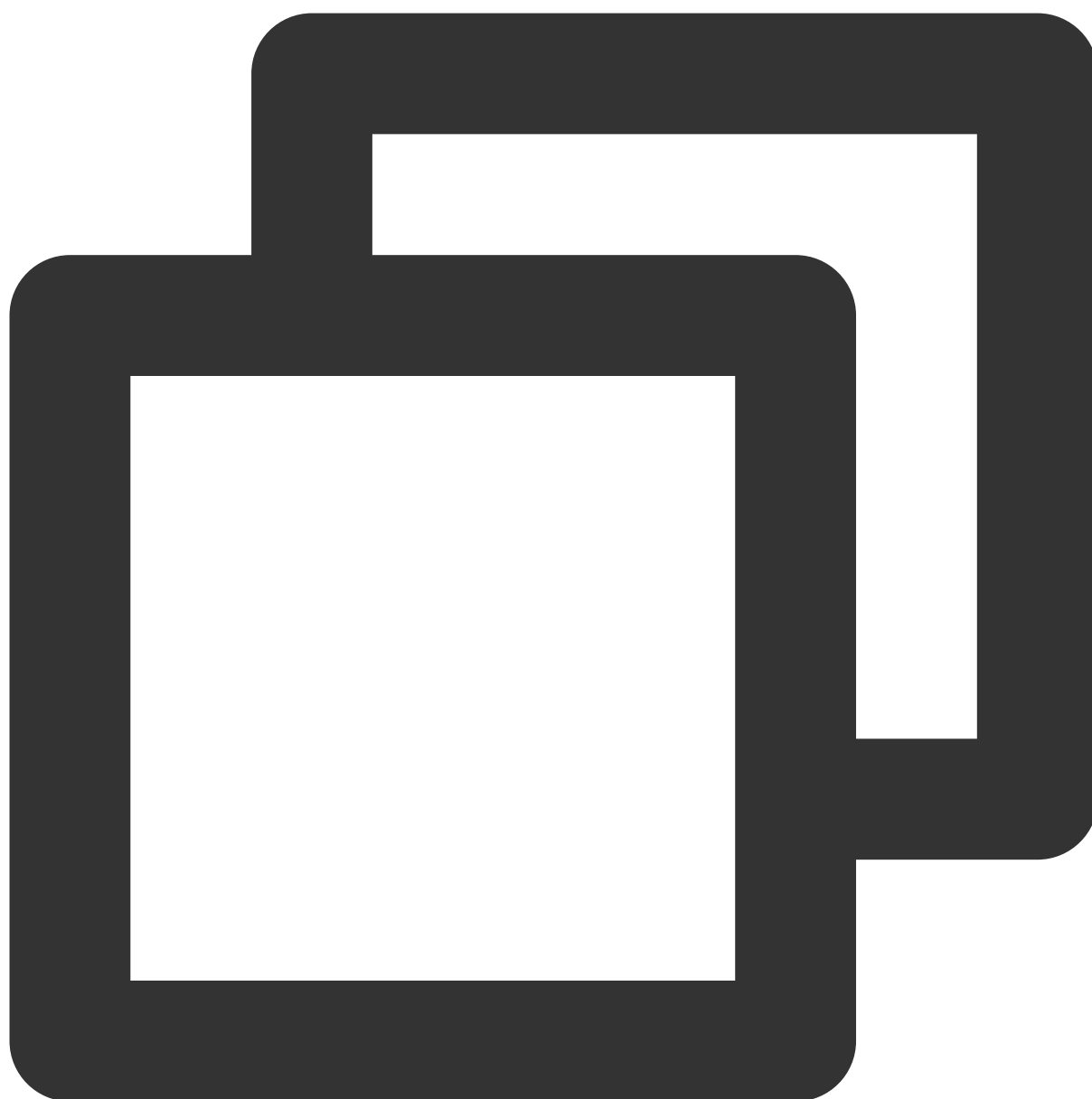
[安装 Go](#)

[下载 Demo](#)

操作步骤

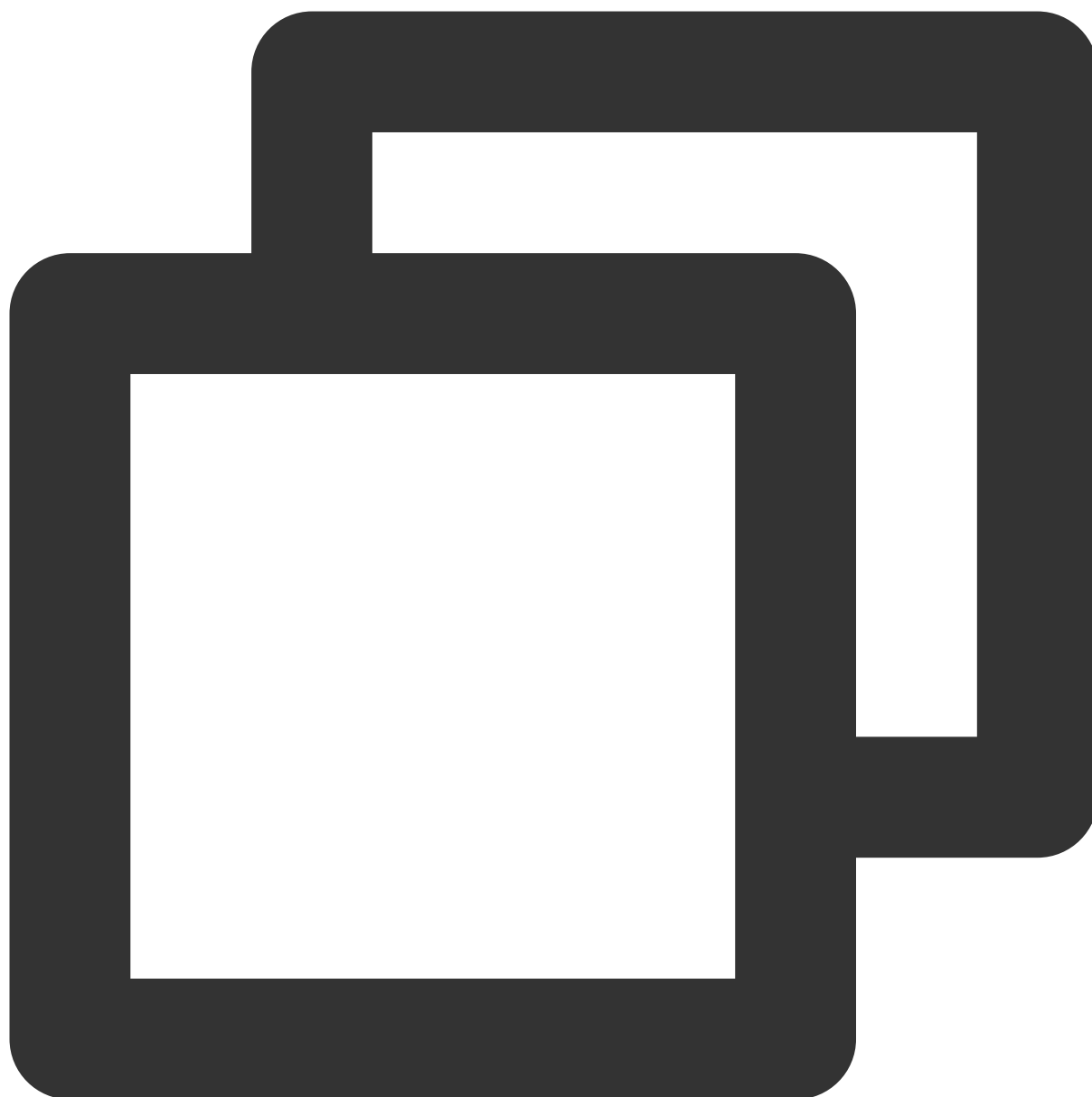
步骤1：准备配置

1. 将下载的 Demo 中的 gokafkdemo 上传至 Linux 服务器。
2. 登录 Linux 服务器，进入 gokafkdemo 目录，执行以下命令添加依赖库。



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

3. 修改配置文件 kafka.json。

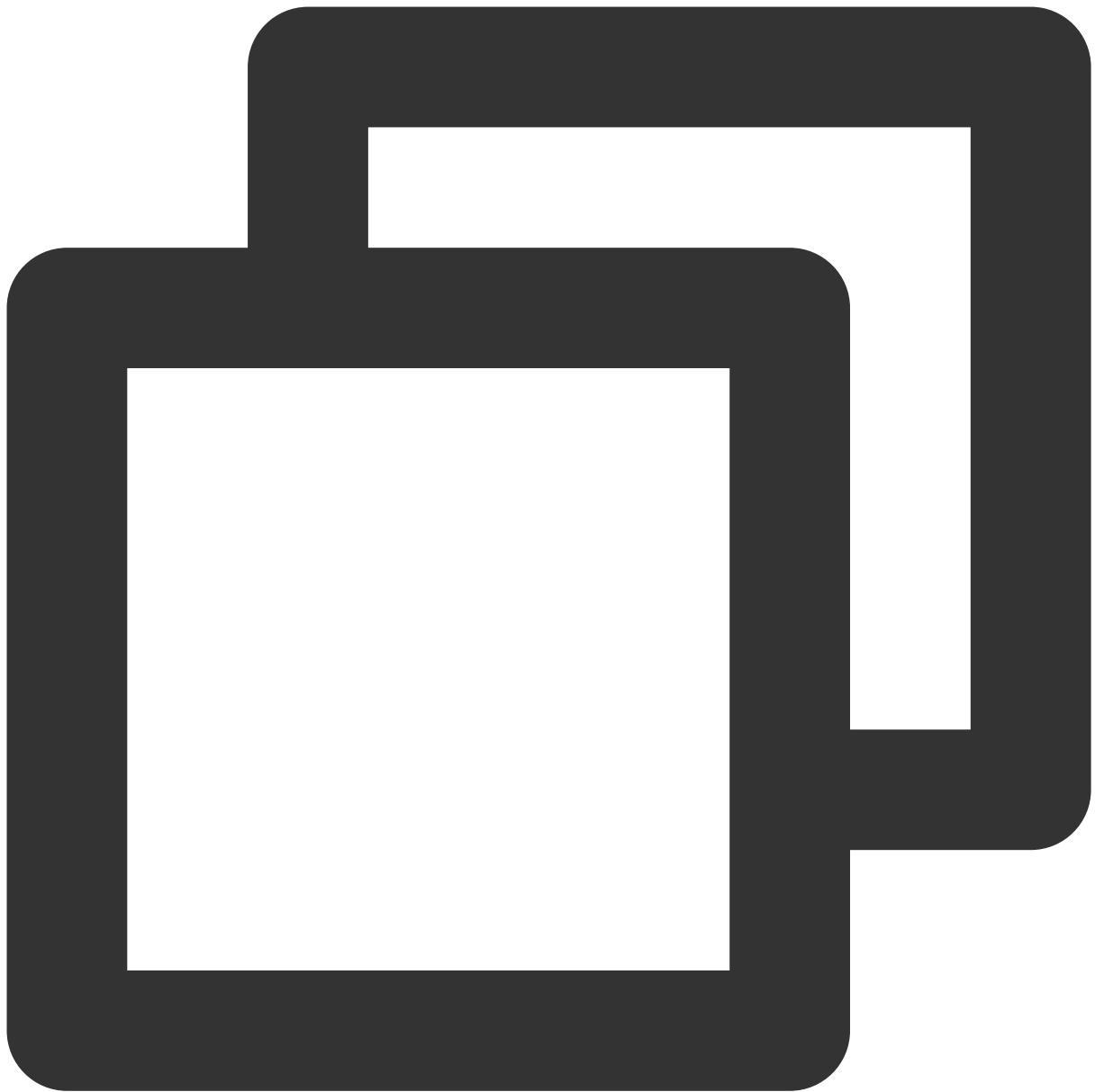


```
{
  "topic": [
    "test"
  ],
  "bootstrapServers": [
    "xx.xx.xx.xx:xxxx"
  ],
  "consumerGroupId": "yourConsumerId"
}
```


参数	描述
topic	Topic 名称，您可以在控制台上 topic 管理页面复制。 
bootstrapServers	接入网络，在控制台的实例详情页面接入方式模块的网络列复制。 
consumerGroupId	您可以自定义设置， Demo 运行成功后可以在 Consumer Group 页面看到该消费者。

步骤2：发送消息

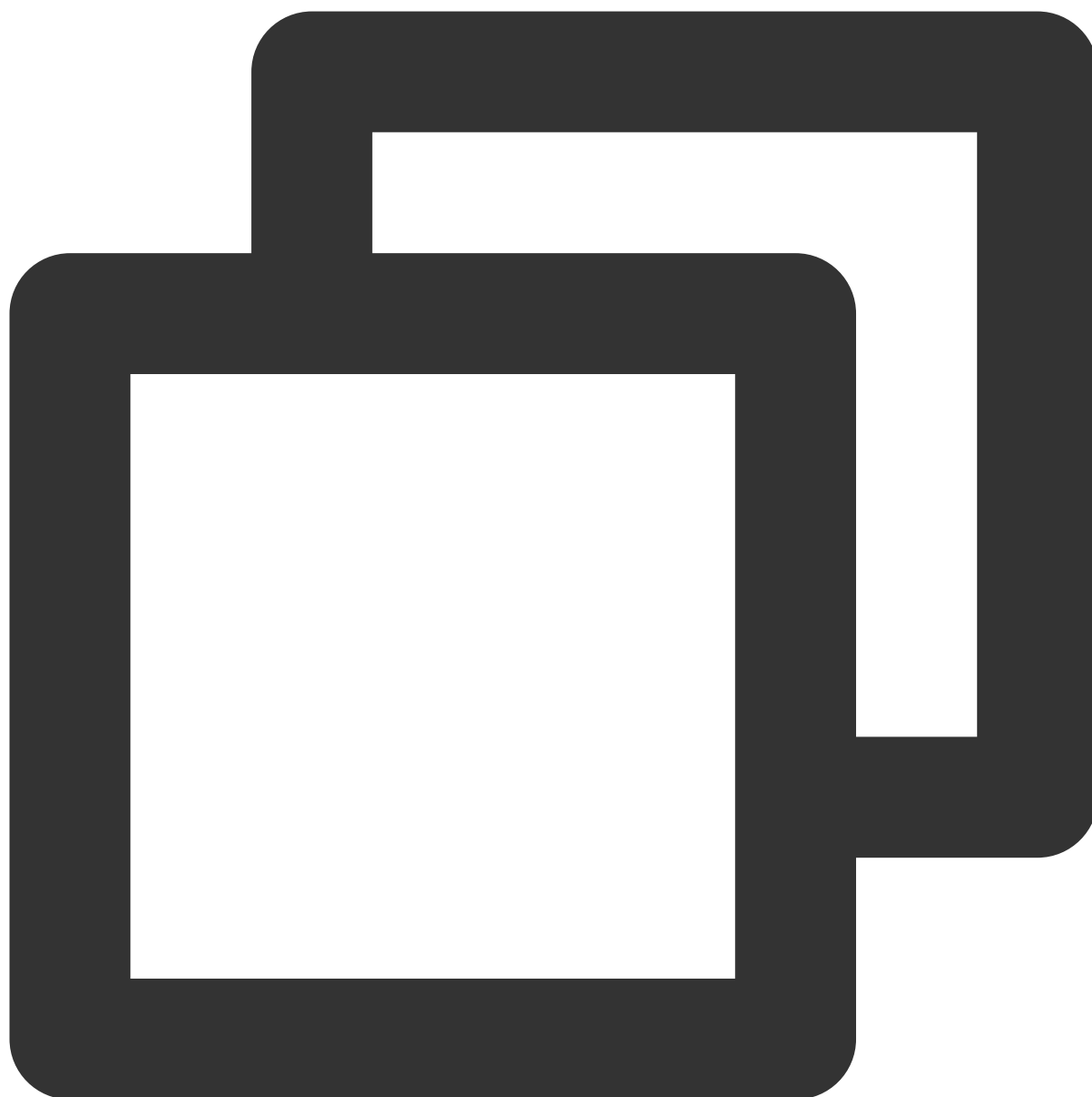
1. 编写生产消息程序。



```
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 {
```

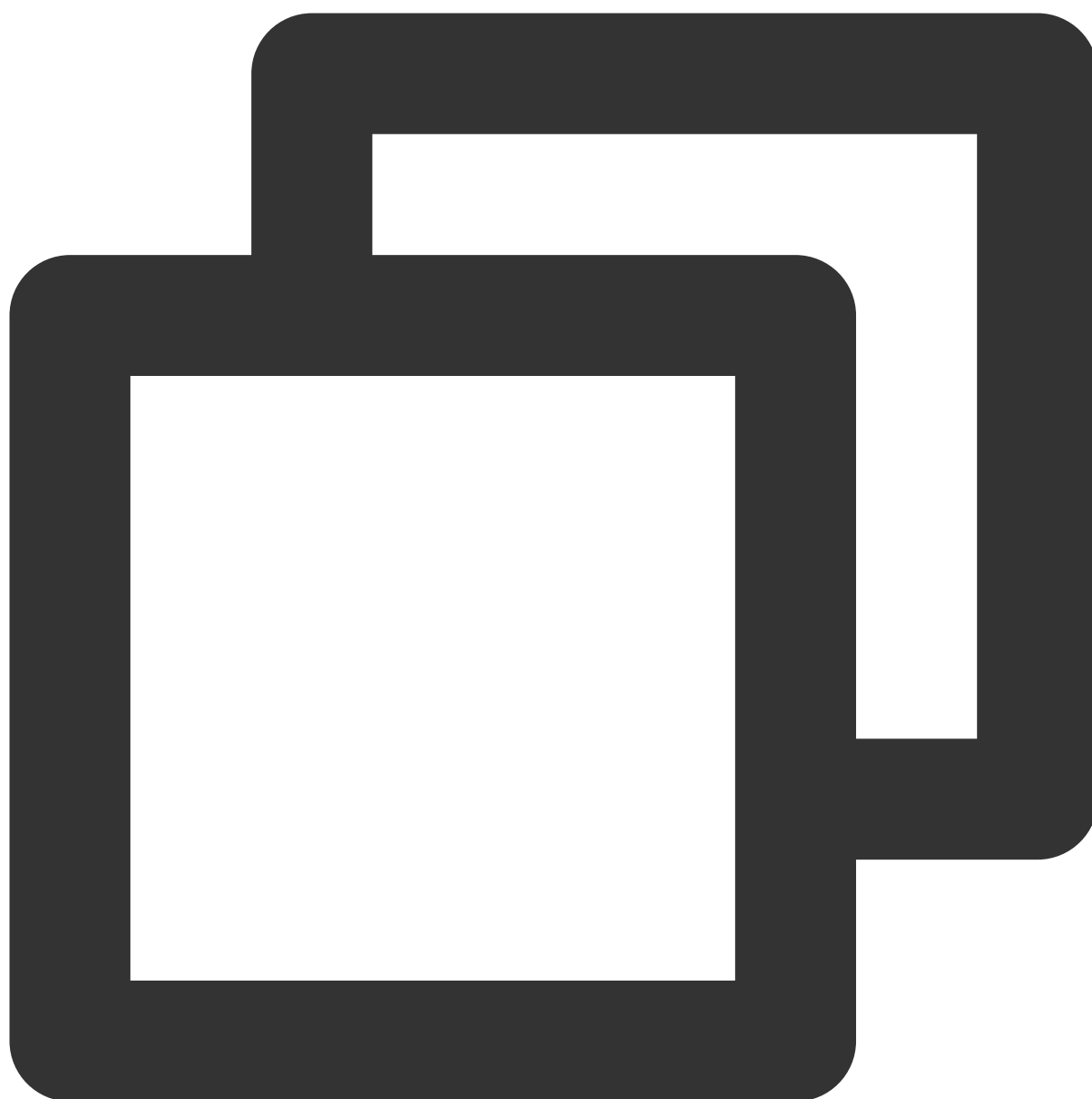
```
        log.Fatal(err)
    }
    p, err := kafka.NewProducer(&kafka.ConfigMap{
        // 设置接入点, 请通过控制台获取对应Topic的接入点。
        "bootstrap.servers": strings.Join(cfg.Servers, ","),
        // 用户不显示配置时, 默认值为1。用户根据自己的业务情况进行设置
        "acks": 1,
        // 请求发生错误时重试次数, 建议将该值设置为大于0, 失败重试最大程度保证消息不丢失
        "retries": 0,
        // 发送请求失败时到下一次重试请求之间的时间
        "retry.backoff.ms": 100,
        // producer 网络请求的超时时间。
        "socket.timeout.ms": 6000,
        // 设置客户端内部重试间隔。
        "reconnect.backoff.max.ms": 3000,
    })
    if err != nil {
        log.Fatal(err)
    }
    defer p.Close()
    // 产生的消息 传递至报告处理程序
    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)
                }
            }
        }
    }()
    // 异步发送消息
    topic := cfg.Topic[0]
    for _, word := range []string{"Confluent-Kafka", "Golang Client Message"} {
        _ = p.Produce(&kafka.Message{
            TopicPartition: kafka.TopicPartition{Topic: &topic, Partition:
                Value: []byte(word),
            }, nil)
    }
    // 等待消息传递
    p.Flush(10 * 1000)
}
```

2. 编译并运行程序发送消息。



```
go run main.go
```

3. 查看运行结果，示例如下。



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

4. 在 [CKafka 控制台](#) 的 **topic 管理** 页面，选择对应的 Topic，单击 **更多 > 消息查询**，查看刚刚发送的消息。

消息查询

消息查询会占用CKafka实例的带宽资源，建议您尽量缩小查询范围查询，不要频繁操作。
消息查询最多展示最近的20条消息。

实例

Topic

查询类型

分区ID

时间

按位点查询

按时间查询

0

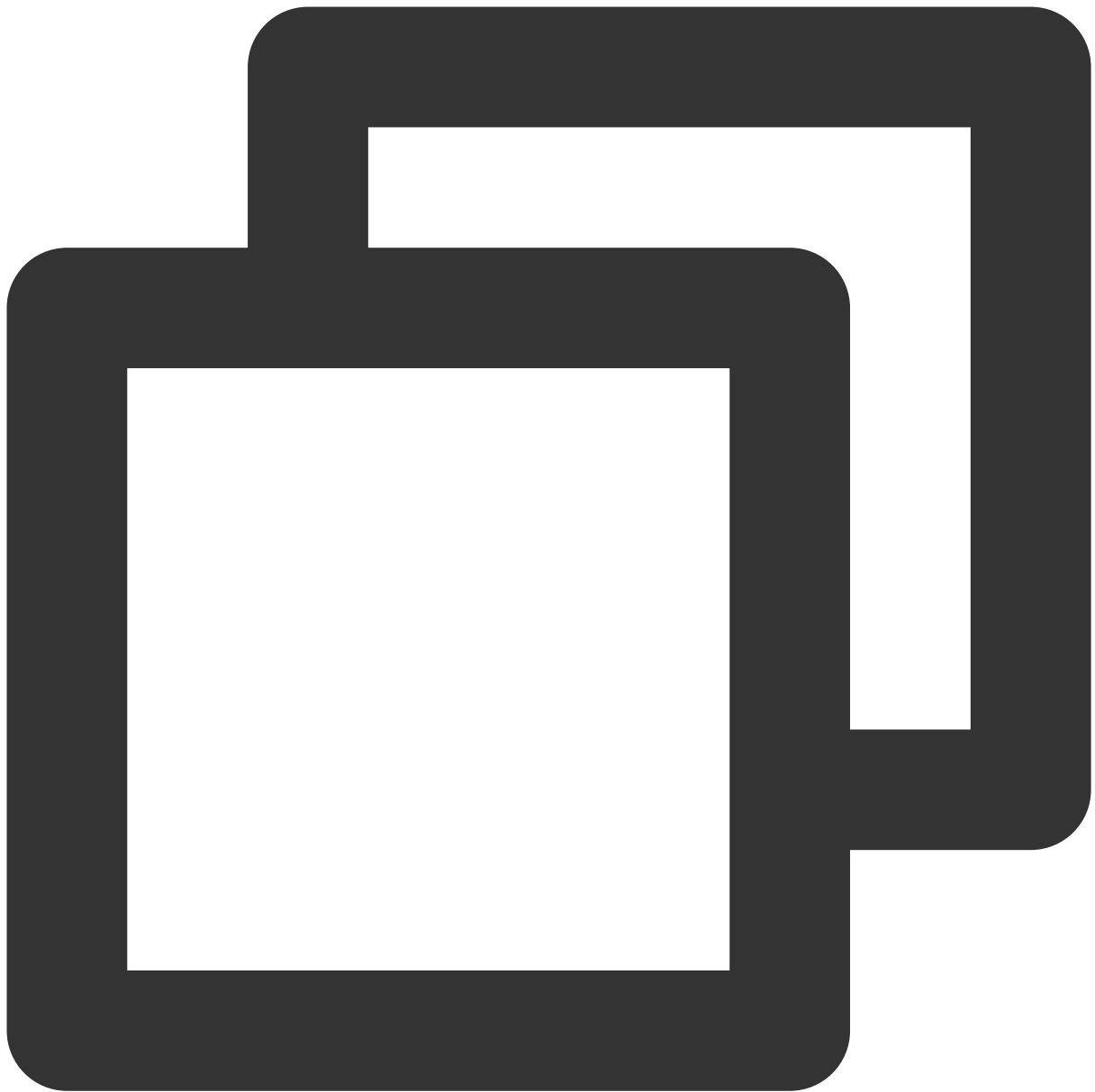
2017-22:54

查询

分区ID	位点	时间戳
0	628	2017-25:31
0	629	2017-25:31

步骤3：消费消息

- 1. 编写消费消息程序。



```
package main
import (
    "fmt"
    "gokafkademio/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{
        // 设置接入点, 请通过控制台获取对应Topic的接入点。
        "bootstrap.servers": strings.Join(cfg.Servers, ","),
        // 设置的消息消费组
        "group.id":          cfg.ConsumerGroupId,
        "auto.offset.reset": "earliest",

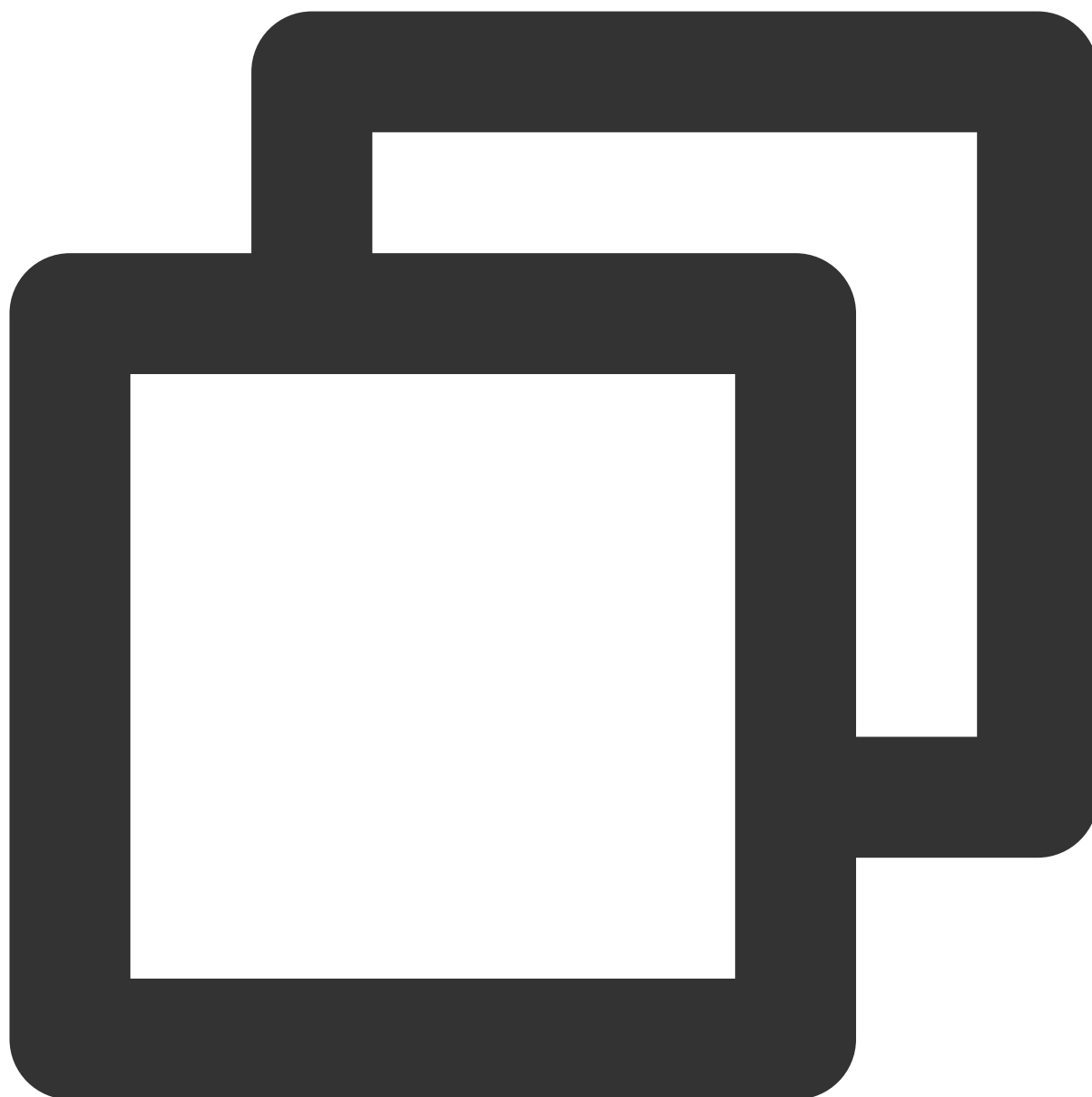
        // 使用 Kafka 消费分组机制时, 消费者超时时间。当 Broker 在该时间内没有收到消费者的心跳时,
        // 发起重新 Rebalance 过程。目前该值的配置必须在 Broker 配置group.min.session.timeout
        "session.timeout.ms": 10000,
    })

    if err != nil {
        log.Fatal(err)
    }
    // 订阅的消息topic 列表
    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 {
            // 客户端将自动尝试恢复所有的 error
            fmt.Printf("Consumer error: %v (%v)\\n", err, msg)
        }
    }

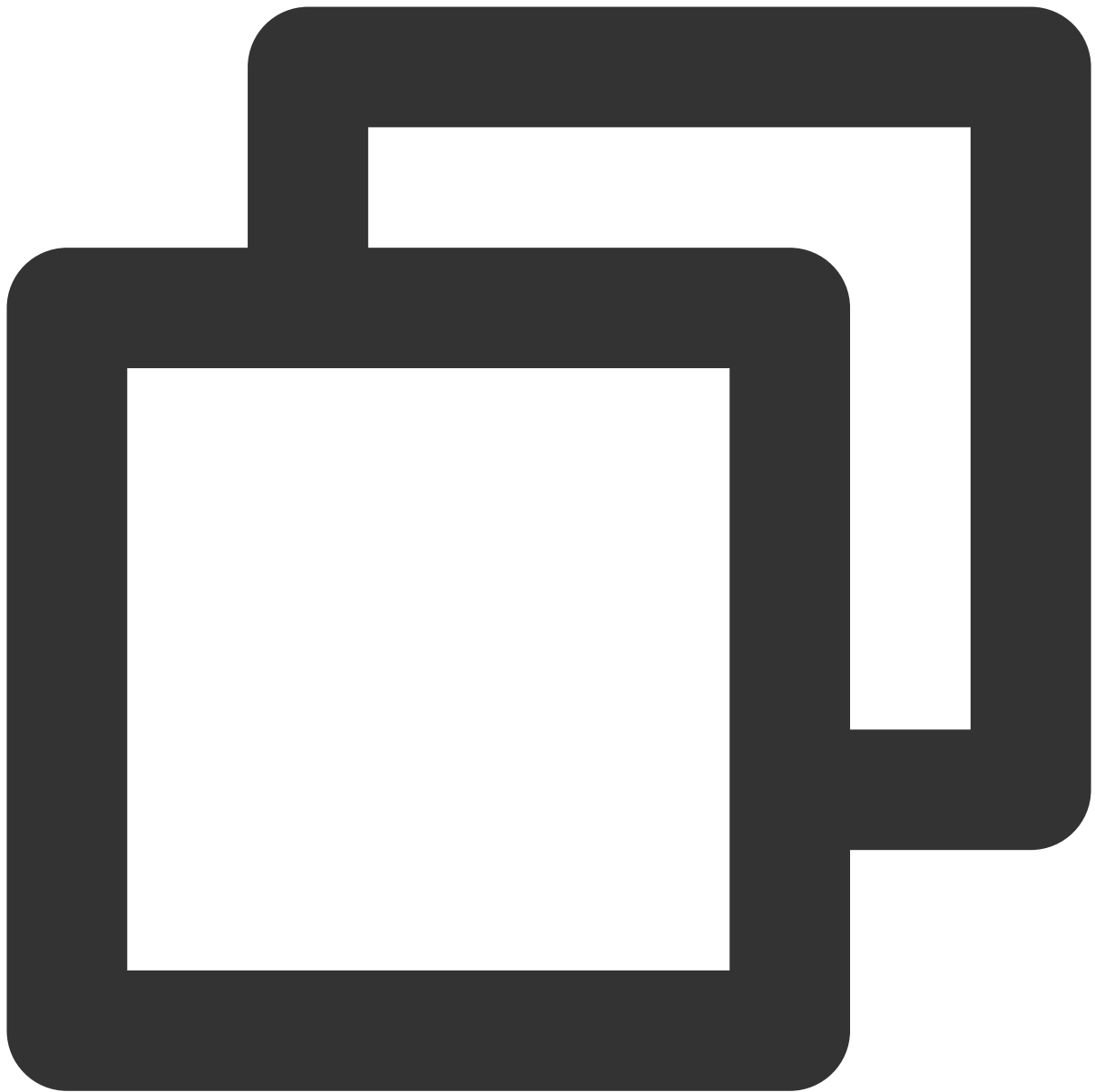
    c.Close()
}
```

2. 编译并运行程序消费消息。



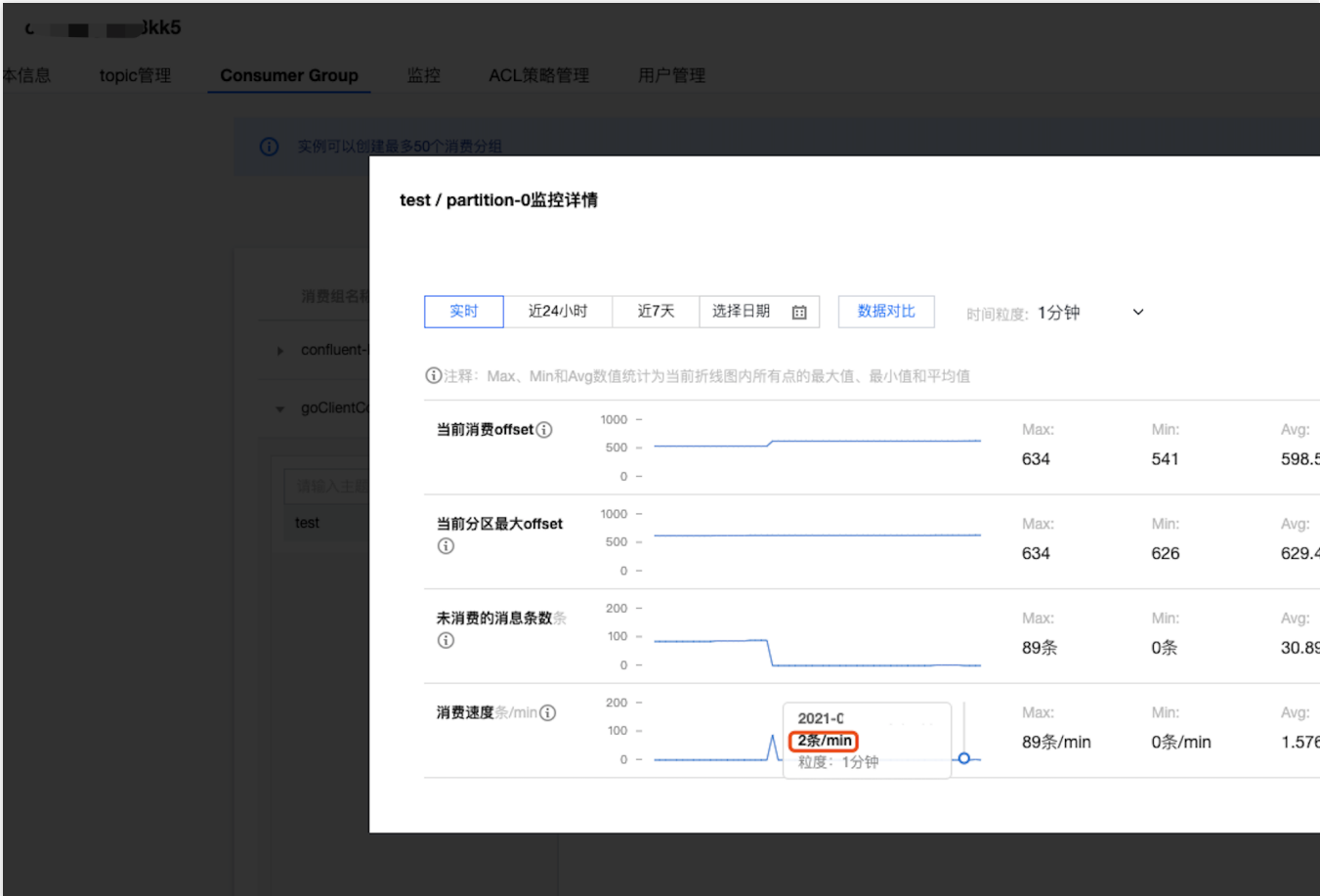
```
go run main.go
```

3. 查看运行结果，示例如下。



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

4. 在 [CKafka 控制台](#) 的 **Consumer Group** 页面，选择对应的消费组名称，在主题名称输入 Topic 名称，单击[查询详情](#)，查看消费详情。



公网 SASL_PLAINTEXT 方式接入

最近更新时间：2024-05-31 14:40:49

操作场景

本文介绍 Go 客户端如何在公网环境下，使用 SASL_PLAINTEXT 方式接入消息队列 CKafka 版收发消息。

前提条件

[安装 Go](#)

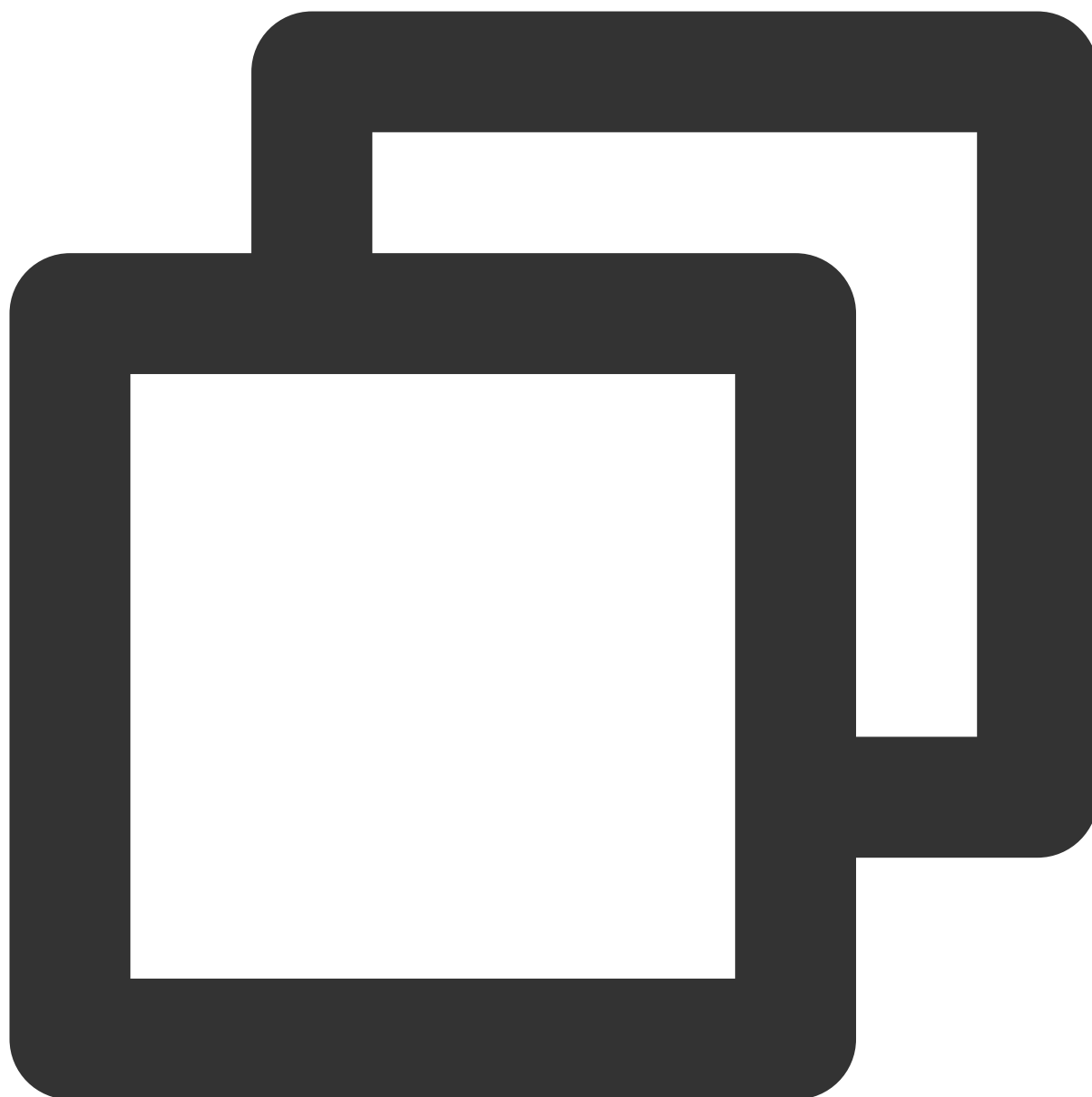
[配置 ACL 策略](#)

[下载 Demo](#)

操作步骤

步骤一：准备 Go 依赖库

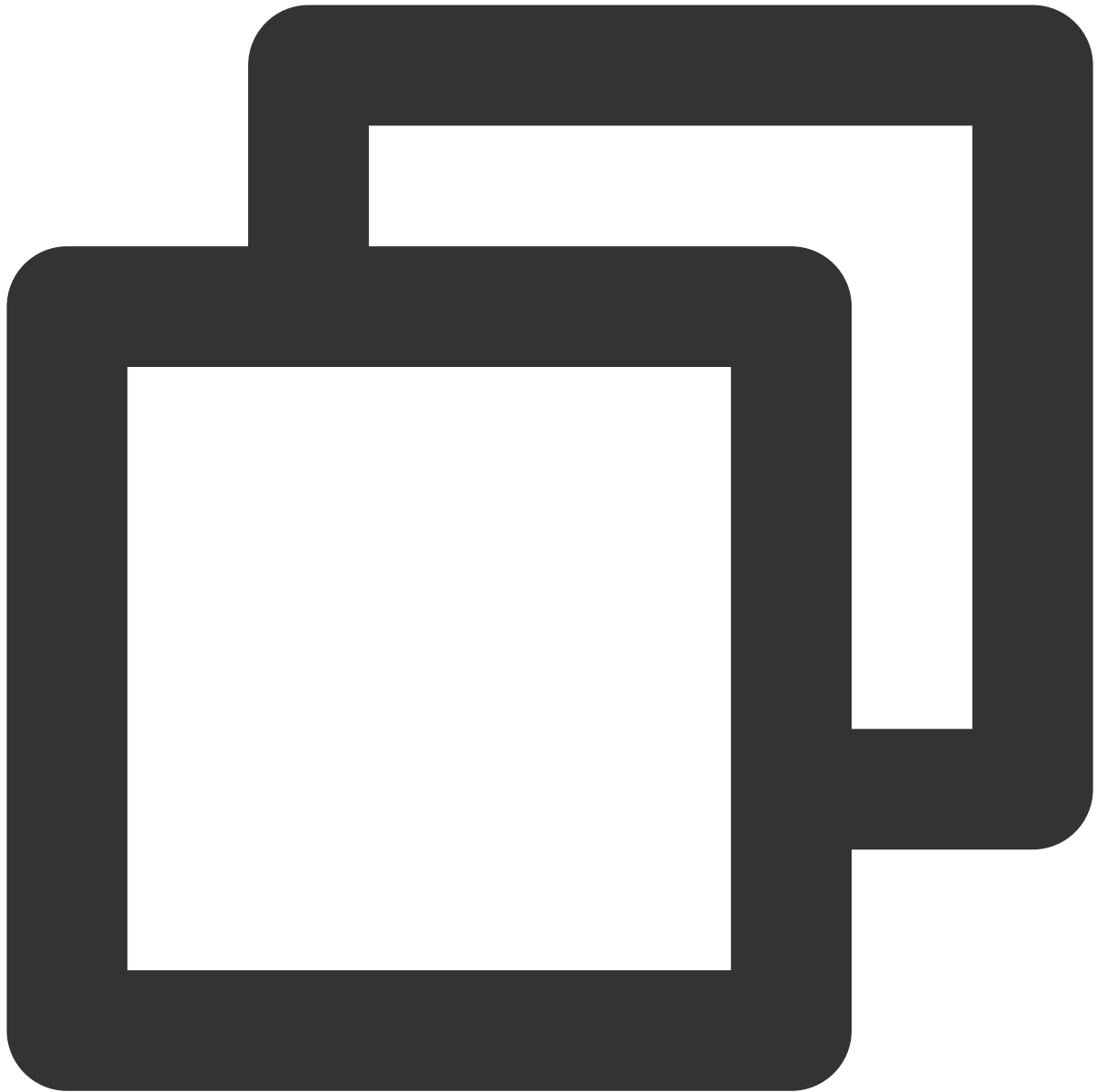
下载 Demo 后，进入 gokafkdemo 目录，执行以下命令安装。



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

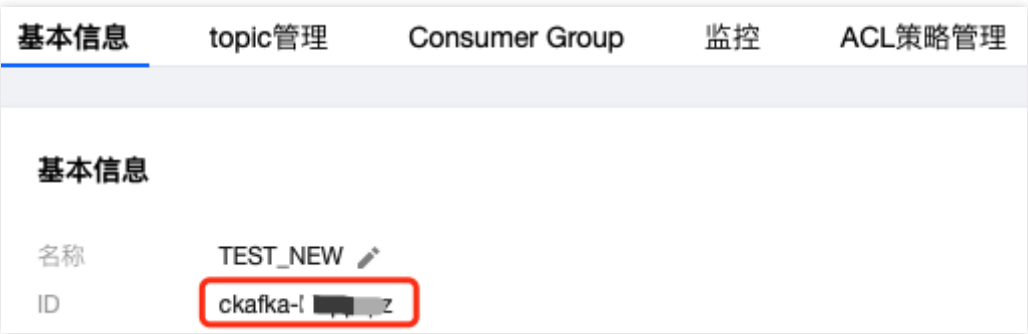
步骤二：准备配置

创建配置文件 kafka.json。



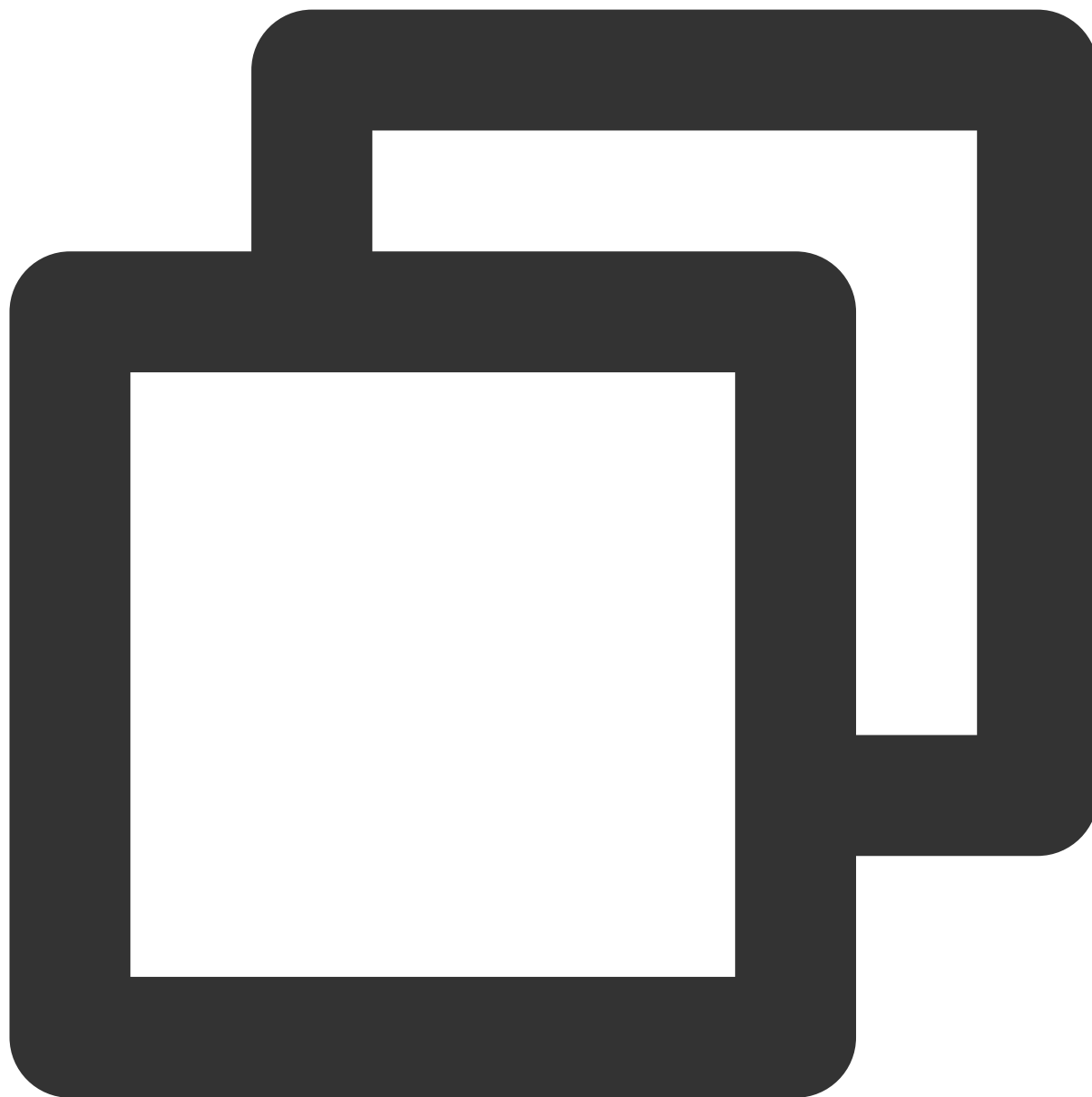
```
{
  "topic": [
    "xxxx"
  ],
  "sasl": {
    "username": "yourUserName",
    "password": "yourPassword",
    "instanceId": "instanceId"
  },
  "bootstrapServers": [
    "xxx.ap-changsha-ec.ckafka.tencentcloudmq.com:6000"
  ]
}
```

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

参数	描述
topic	Topic 名称，您可以在控制台上 topic管理 页面复制。 
sasl.username	用户名，在控制台 用户管理 页面创建用户时设置。
sasl.password	用户密码，在控制台 用户管理 页面创建用户时设置。
sasl.instanceId	实例 ID，在控制台的实例详情页面的基本信息获取。 
bootstrapServers	接入网络，在控制台的实例详情页面接入方式模块的网络列复制。 
consumerGroupId	您可以自定义设置，Demo 运行成功后可以在 Consumer Group 页面看到该消费者。

步骤三：发送消息

1. 编写生产消息程序。



```
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{
        // 设置接入点, 请通过控制台获取对应Topic的接入点。
        "bootstrap.servers": strings.Join(cfg.Servers, ","),
        // SASL 验证机制类型默认选用 PLAIN
        "sasl.mechanism": "PLAIN",
        // 在本地配置 ACL 策略。
        "security.protocol": "SASL_PLAINTEXT",
        // username 是实例 ID + # + 配置的用户名, password 是配置的用户密码。
        "sasl.username": fmt.Sprintf("%s#%", cfg.SASL.InstanceId, cfg.SASL.UserName),
        "sasl.password": cfg.SASL.Password,

        // Kafka producer 的 ack 有 3 种机制, 分别说明如下:
        // -1 或 all:Broker 在 leader 收到数据并同步给所有 ISR 中的 follower 后, 才应答
        // 这种配置提供了最高的数据可靠性, 只要有一个已同步的副本存活就不会有消息丢失。注意: 这
        // 可以配合 Topic 级别参数 min.insync.replicas 使用。
        // 0: 生产者不等待来自 broker 同步完成的确认, 继续发送下一条(批)消息。这种配置生产性
        // 1: 生产者在 leader 已成功收到的数据并得到确认后发送下一条(批)消息。这种配置是
        // 用户不显示配置时, 默认值为1。用户根据自己的业务情况进行设置
        "acks": 1,
        // 请求发生错误时重试次数, 建议将该值设置为大于0, 失败重试最大程度保证消息不丢失
        "retries": 0,
        // 发送请求失败时到下一次重试请求之间的时间
        "retry.backoff.ms": 100,
        // producer 网络请求的超时时间。
        "socket.timeout.ms": 6000,
        // 设置客户端内部重试间隔。
        "reconnect.backoff.max.ms": 3000,
    })
    if err != nil {
        log.Fatal(err)
    }

    defer p.Close()

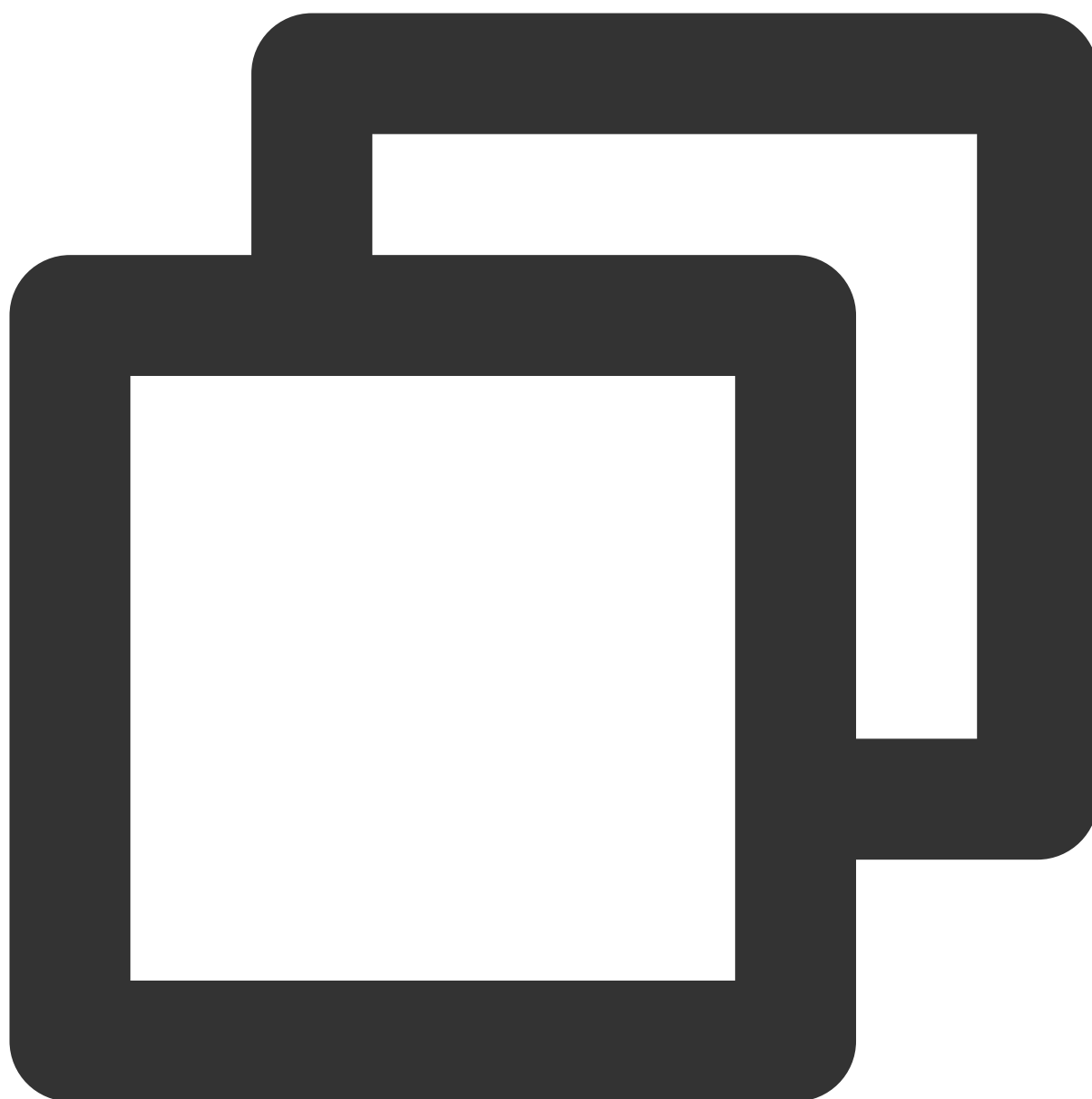
    // 产生的消息 传递至报告处理程序
    go func() {
        for e := range p.Events() {
            switch ev := e.(type) {
```

```
        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)
            }
        }
    }
}()

// 异步发送消息
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)
}

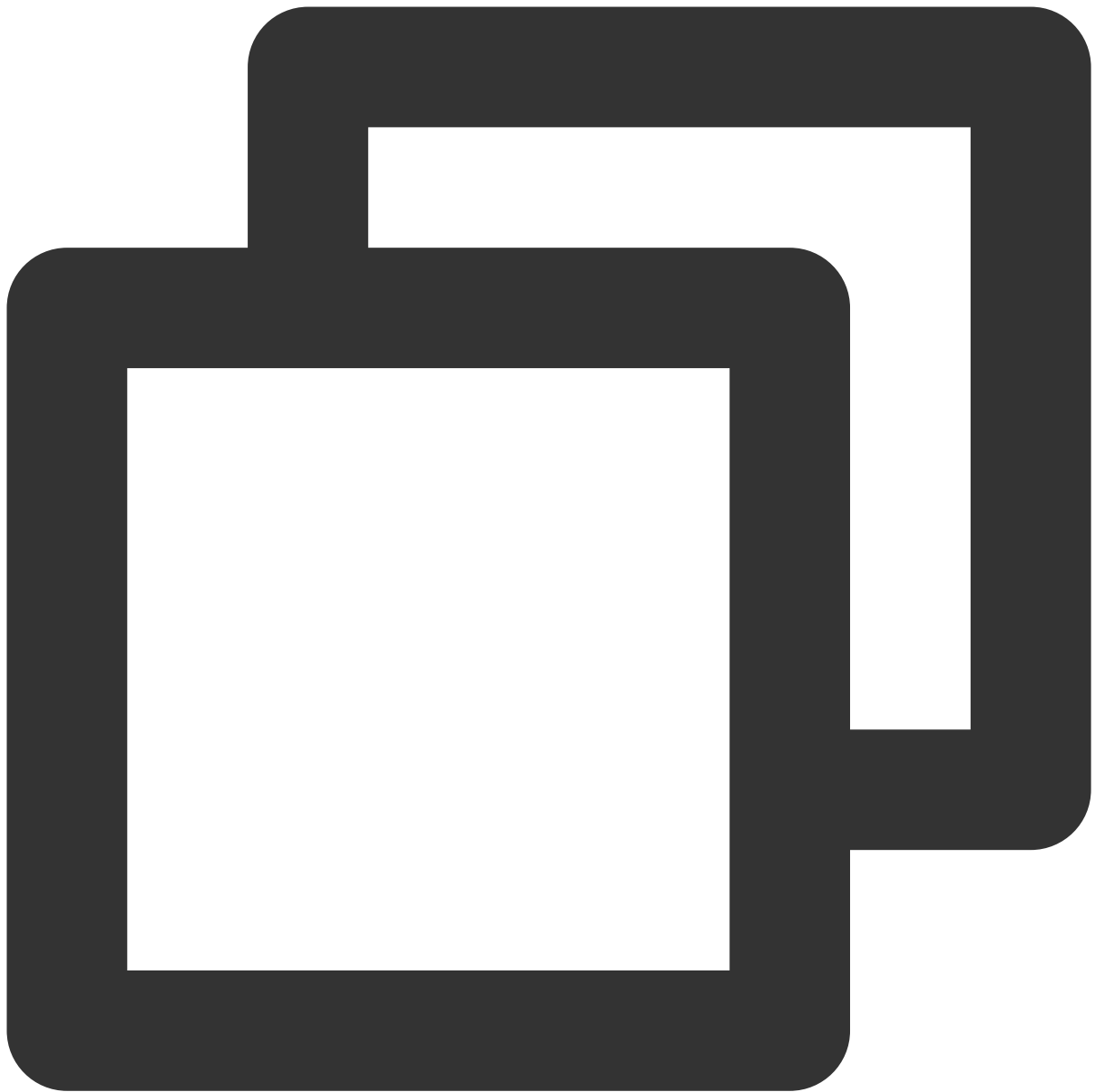
// 等待消息传递
p.Flush(10 * 1000)
```

2. 编译并运行程序发送消息。



```
go run main.go
```

3. 查看运行结果，示例如下。



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

4. 在 [CKafka 控制台](#) 的 **topic 管理** 页面，选择对应的 Topic，单击 **更多 > 消息查询**，查看刚刚发送的消息。

消息查询

消息查询会占用CKafka实例的带宽资源，建议您尽量缩小查询范围查询，不要频繁操作。

消息查询最多展示最近的20条消息。

实例

5

Topic

test

查询类型

按位点查询

按时间查询

分区ID

0

时间

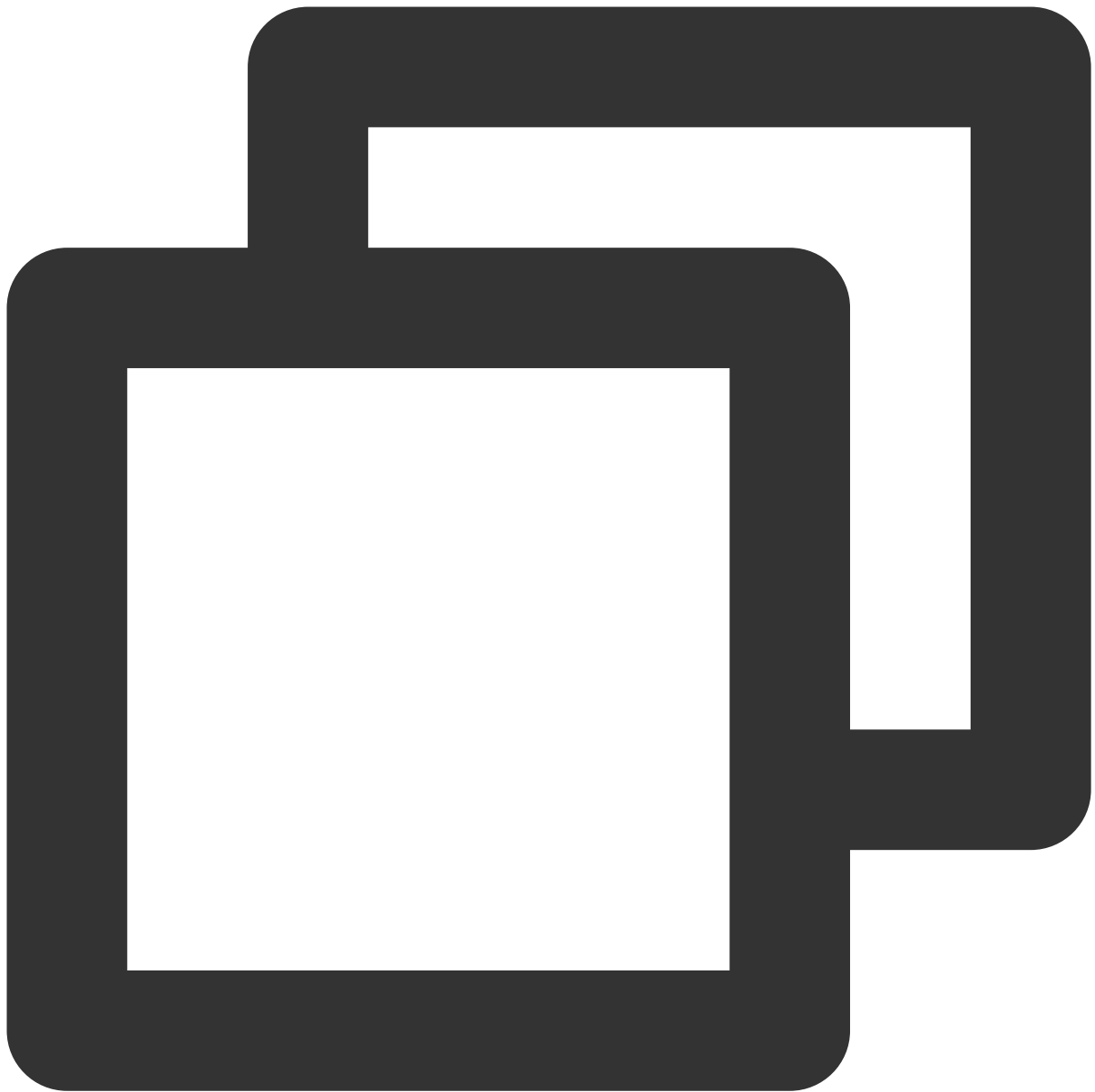
2024-07-17 17:22:54

查询

分区ID	位点	时间戳	操作
0	628	2024-07-17 17:25:31	查看消息
0	629	2024-07-17 17:25:31	查看消息

步骤四：消费消息

- 1. 编写消费消息程序。



```
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{
        // 设置接入点, 请通过控制台获取对应Topic的接入点。
        "bootstrap.servers": strings.Join(cfg.Servers, ","),
        // SASL 验证机制类型默认选用 PLAIN
        "sasl.mechanism": "PLAIN",
        // 在本地配置 ACL 策略。
        "security.protocol": "SASL_PLAINTEXT",
        // username 是实例 ID + # + 配置的用户名, password 是配置的用户密码。
        "sasl.username": fmt.Sprintf("%s#%s", cfg.SASL.InstanceId, cfg.SASL.UserName),
        "sasl.password": cfg.SASL.Password,
        // 设置的消息消费组
        "group.id":          cfg.ConsumerGroupId,
        "auto.offset.reset": "earliest",

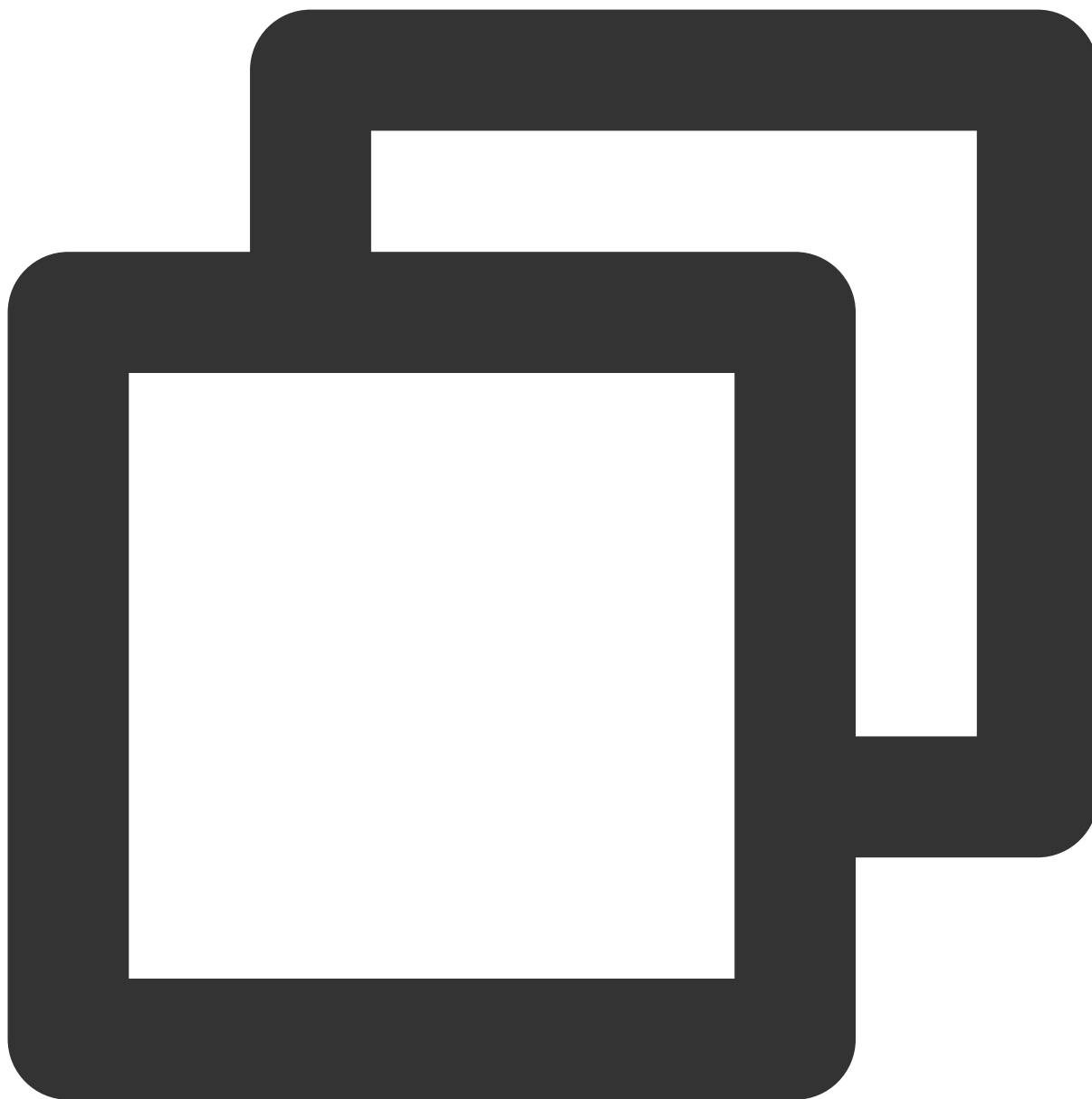
        // 使用 Kafka 消费分组机制时, 消费者超时时间。当 Broker 在该时间内没有收到消费者的心
        // 发起重新 Rebalance 过程。目前该值的配置必须在 Broker 配置group.min.session.ti
        "session.timeout.ms": 10000,
    })

    if err != nil {
        log.Fatal(err)
    }
    // 订阅的消息topic 列表
    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.Value))
        } else {
            // 客户端将自动尝试恢复所有的 error
            fmt.Printf("Consumer error: %v (%v)\\n", err, msg)
        }
    }

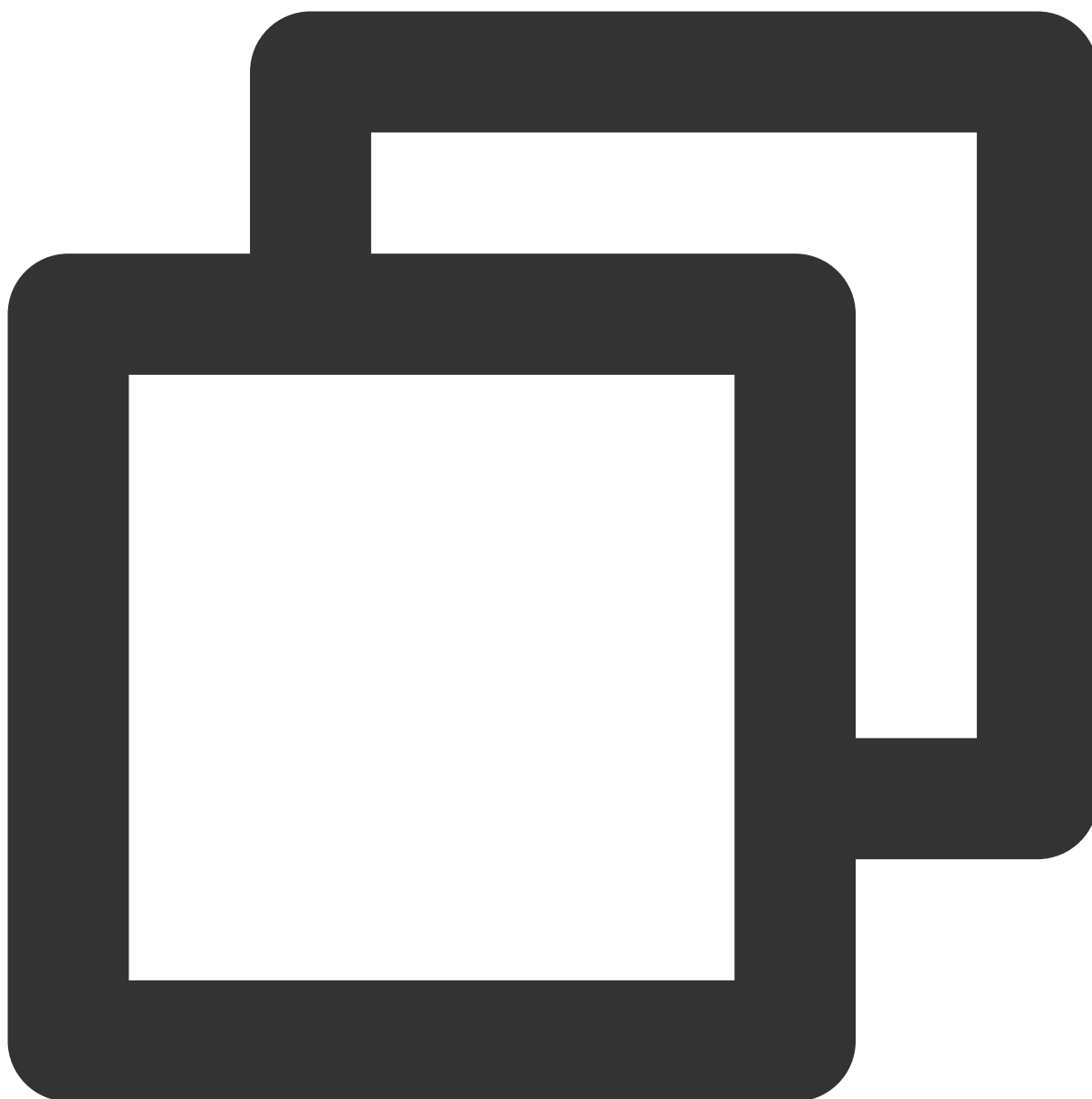
    c.Close()
}
```

2. 编译并运行程序消费消息。



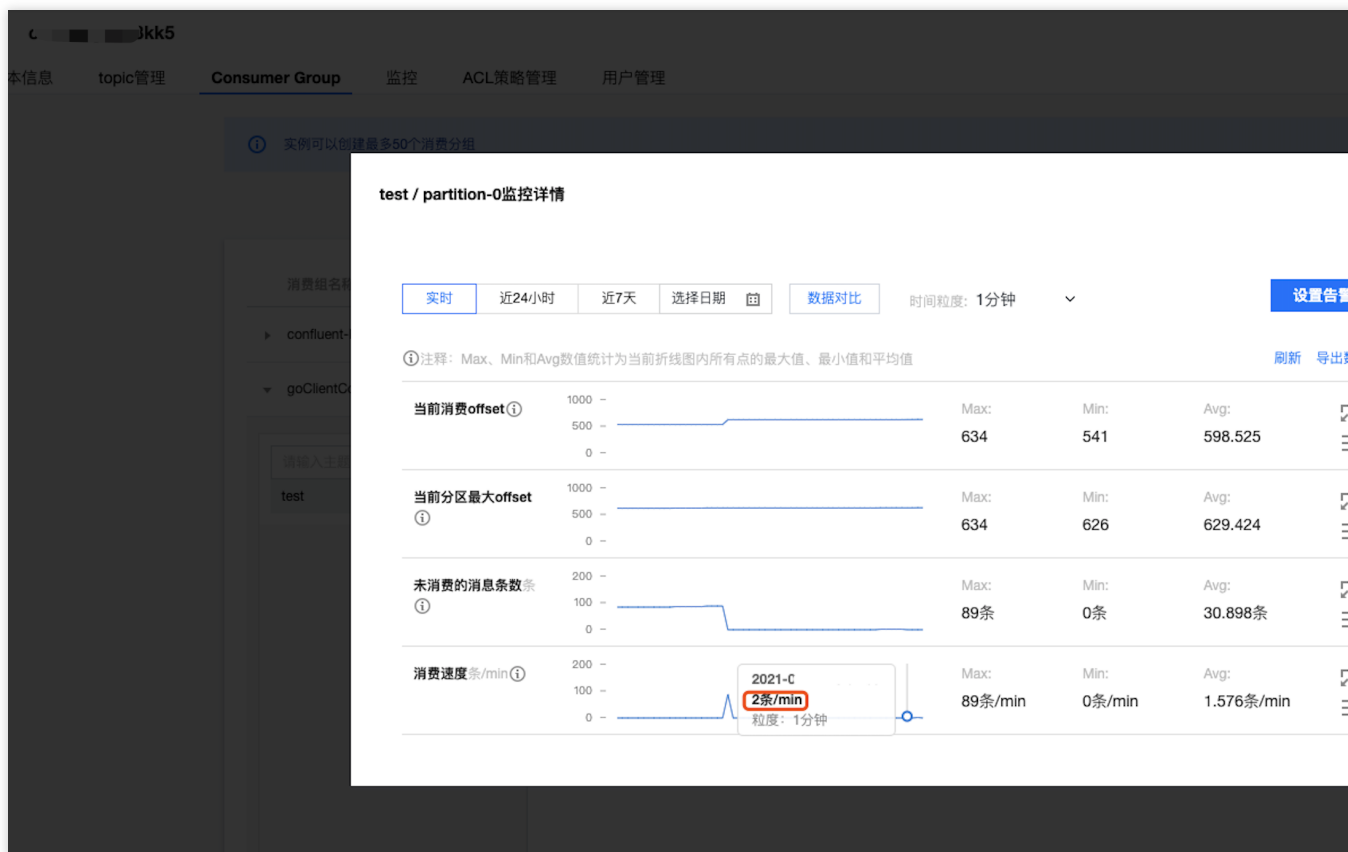
```
go run main.go
```

3. 查看运行结果，示例如下。



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

4. 在 [CKafka 控制台](#) 的 **Consumer Group** 页面，选择对应的消费组名称，在主题名称输入 Topic 名称，单击[查询详情](#)，查看消费详情。



PHP SDK

VPC 网络接入

最近更新时间：2024-05-31 14:40:49

操作场景

该任务以 PHP 客户端为例指导您使用 VPC 网络接入消息队列 CKafka 版并收发消息。

前提条件

[安装 librdkafka](#)

[安装 PHP 5.6 或以上版本](#)

[安装 PEAR](#)

[下载 Demo](#)

操作步骤

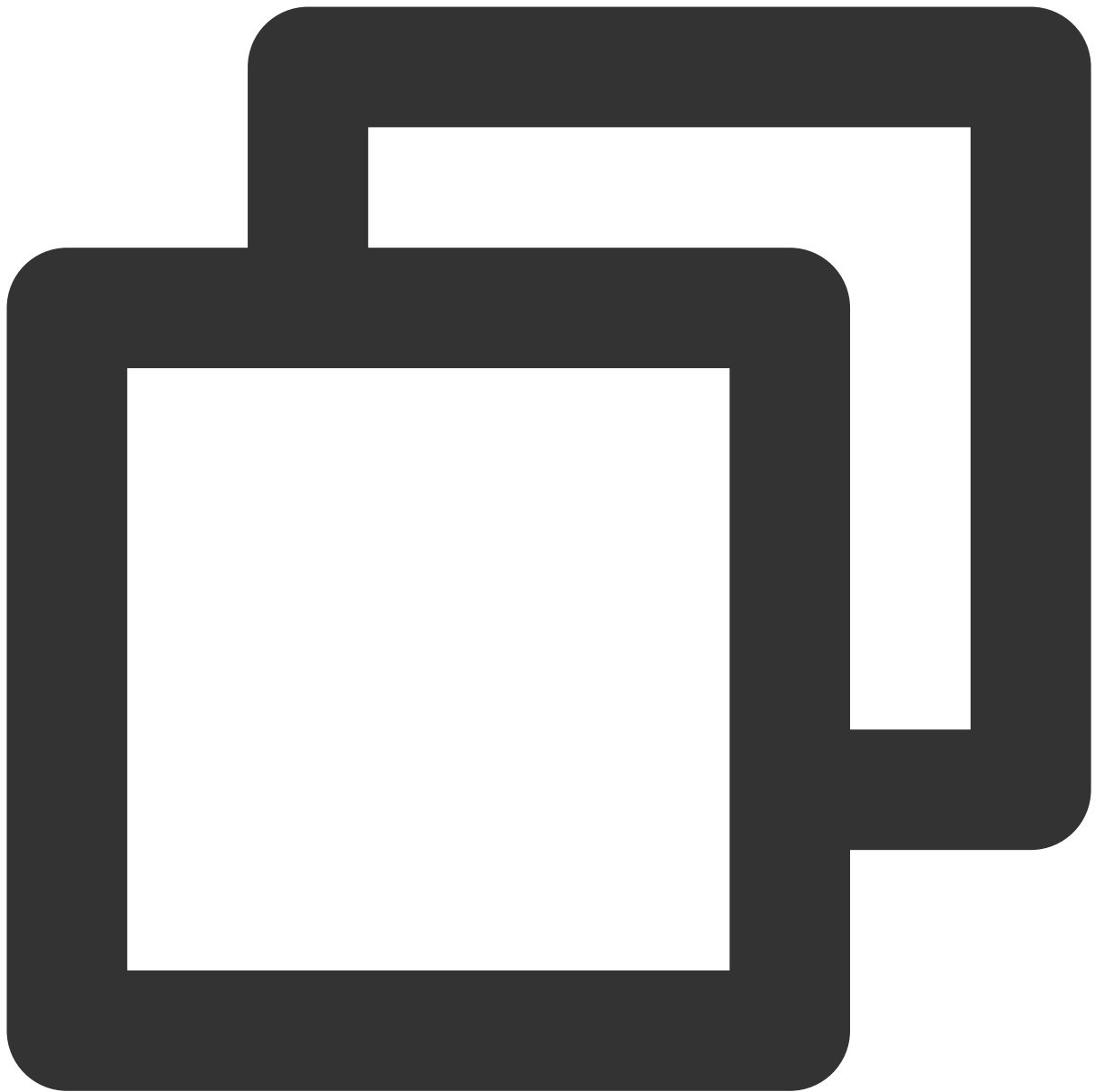
步骤一：添加 Rdkafka 扩展

1. 在 [rdkafka 官方页面](#) 查找最新的 rdkafka php 扩展包版本。

说明：

不同版本的包对 PHP 版本要求不同，这里仅以 4.1.2 为示例。

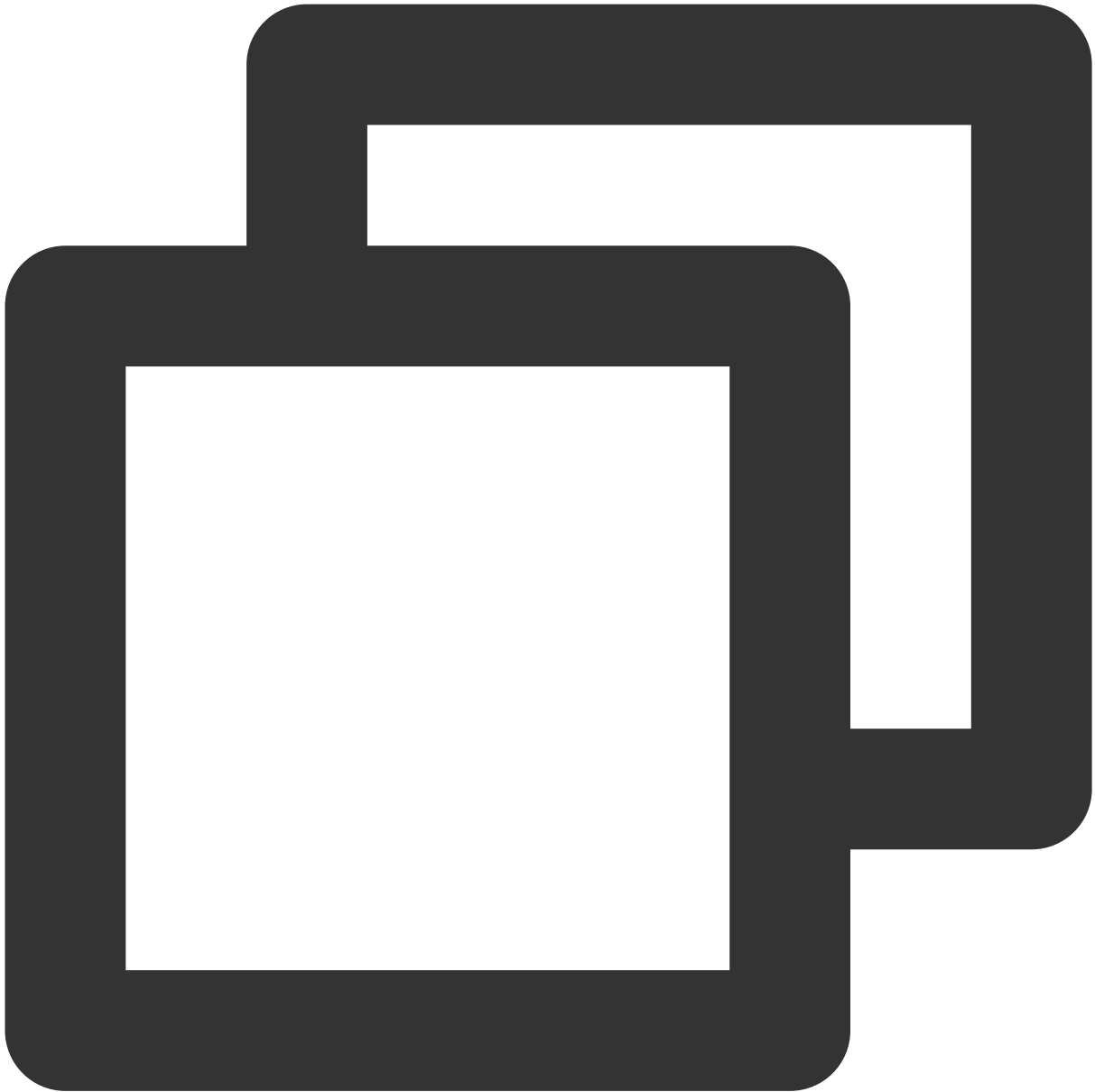
2. 登录 Linux 服务器，安装 rdkafka 扩展。



```
wget --no-check-certificate https://pecl.php.net/get/rdkafka-4.1.2.tgz
pear install rdkafka-4.1.2.tgz
# 安装成功会提示 "install ok" 和 "You should add "extension=rdkafka.so" to php.ini"
# 如果安装失败, 请根据提示解决
# 安装成功后在 php.ini 添加 extension=rdkafka.so
# 执行 php --ini 后, Loaded Configuration File: 显示的就是 php.ini 所在位置
echo 'extension=rdkafka.so' >> /etc/php.ini
```

步骤二：准备配置

- 1. 将下载的 Demo 中的 phpkafkademo 上传至 Linux 服务器。
- 2. 登录 Linux 服务器，进入 phpkafkademo 目录，修改 CKafkaSetting.php 配置文件。



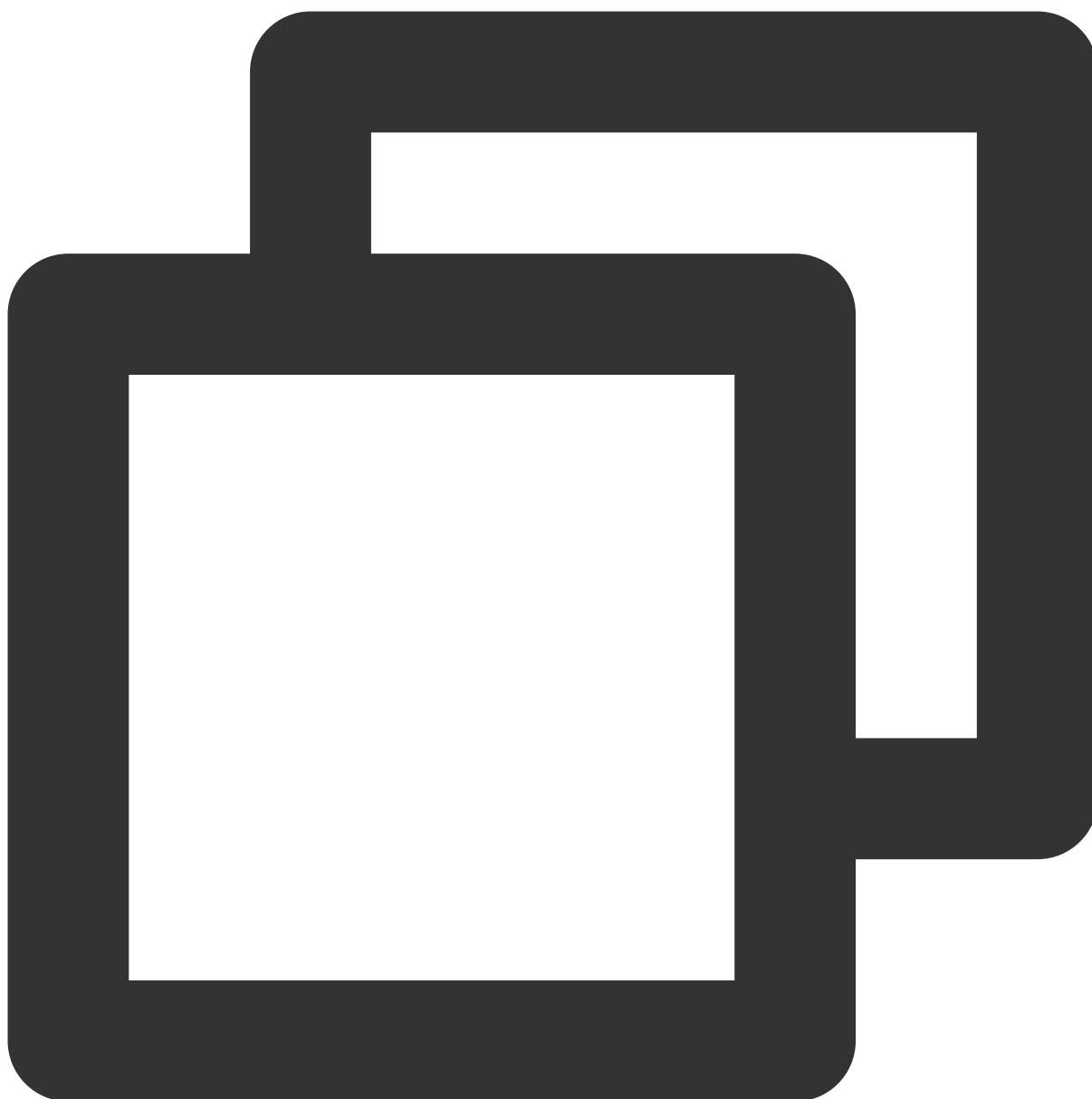
```
<?php
return [
    'bootstrap_servers' => 'bootstrap_servers1:port,bootstrap_servers2:port',
    'topic_name' => 'topic_name',
    'group_id' => 'php-demo',
];
```

参数	描述
----	----

bootstrap_servers	接入方式? 添加路由策略 <table><tr><th>接入类型</th><th>接入方式</th><th>网络</th><th>操作</th></tr><tr><td>VPC网络</td><td>PLAINTEXT</td><td>vpc-l subnet-5gl6yj86 1.....0092</td><td>删除 查看所有IP和端口</td></tr></table>	接入类型	接入方式	网络	操作	VPC网络	PLAINTEXT	vpc-l subnet-5gl6yj86 1.....0092	删除 查看所有IP和端口												
接入类型	接入方式	网络	操作																		
VPC网络	PLAINTEXT	vpc-l subnet-5gl6yj86 1.....0092	删除 查看所有IP和端口																		
topic_name	Topic 名称，您可以在控制台上 topic 管理页面复制。 ckafka- 基本信息topic管理Consumer Group监控事件中心HTTP接入ACL策略管理智能运维专业版集群备份专业版 新建(24/450) 请输入TopicId或名称 <table><tr><th>ID/名称</th><th>监控</th><th>分区数(个)</th><th>副本数(个)</th><th>标签</th><th>备注</th><th>创建时间</th><th>消息保留时间</th><th>状态</th><th>操作</th></tr><tr><td>topic-j oocs</td><td></td><td>3</td><td>2</td><td></td><td></td><td>2023-03-25 23:03:00</td><td>2 天</td><td>正常</td><td>编辑 删除 更多</td></tr></table>	ID/名称	监控	分区数(个)	副本数(个)	标签	备注	创建时间	消息保留时间	状态	操作	topic-j oocs		3	2			2023-03-25 23:03:00	2 天	正常	编辑 删除 更多
ID/名称	监控	分区数(个)	副本数(个)	标签	备注	创建时间	消息保留时间	状态	操作												
topic-j oocs		3	2			2023-03-25 23:03:00	2 天	正常	编辑 删除 更多												
group_id	消费者的组 ID，您可以自定义设置，Demo 运行成功后可以在 Consumer Group 页面看到																				

步骤三：发送消息

1. 编写生产消息程序 Producer.php。



```
<?php

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

$conf = new RdKafka\Conf();
// 设置入口服务，请通过控制台获取对应的服务地址。
$conf->set('bootstrap.servers', $setting['bootstrap_servers']);
// Kafka producer 的 ack 有 3 种机制，分别说明如下：
// -1 或 all:Broker 在 leader 收到数据并同步给所有 ISR 中的 follower 后，才应答给 Produce
// 这种配置提供了最高的数据可靠性，只要有一个已同步的副本存活就不会有消息丢失。注意：这种配置不能确
// 可以配合 Topic 级别参数 min.insync.replicas 使用。
```

```
// 0：生产者不等待来自 broker 同步完成的确认，继续发送下一条（批）消息。这种配置生产性能最高，但类
//（当服务器故障时可能会有数据丢失，如果 leader 已死但是 producer 不知情，则 broker 收不到消息
// 1：生产者在 leader 已成功收到的数据并得到确认后发送下一条（批）消息。这种配置是在生产吞吐和
//（如果leader已死但是尚未复制，则消息可能丢失）
// 用户不显示配置时，默认值为1。用户根据自己的业务情况进行设置
$conf->set('acks', '1');
// 请求发生错误时重试次数，建议将该值设置为大于0，失败重试最大程度保证消息不丢失
$conf->set('retries', '0');
// 发送请求失败时到下一次重试请求之间的时间
$conf->set('retry.backoff.ms', 100);
// producer 网络请求的超时时间。
$conf->set('socket.timeout.ms', 6000);
$conf->set('reconnect.backoff.max.ms', 3000);

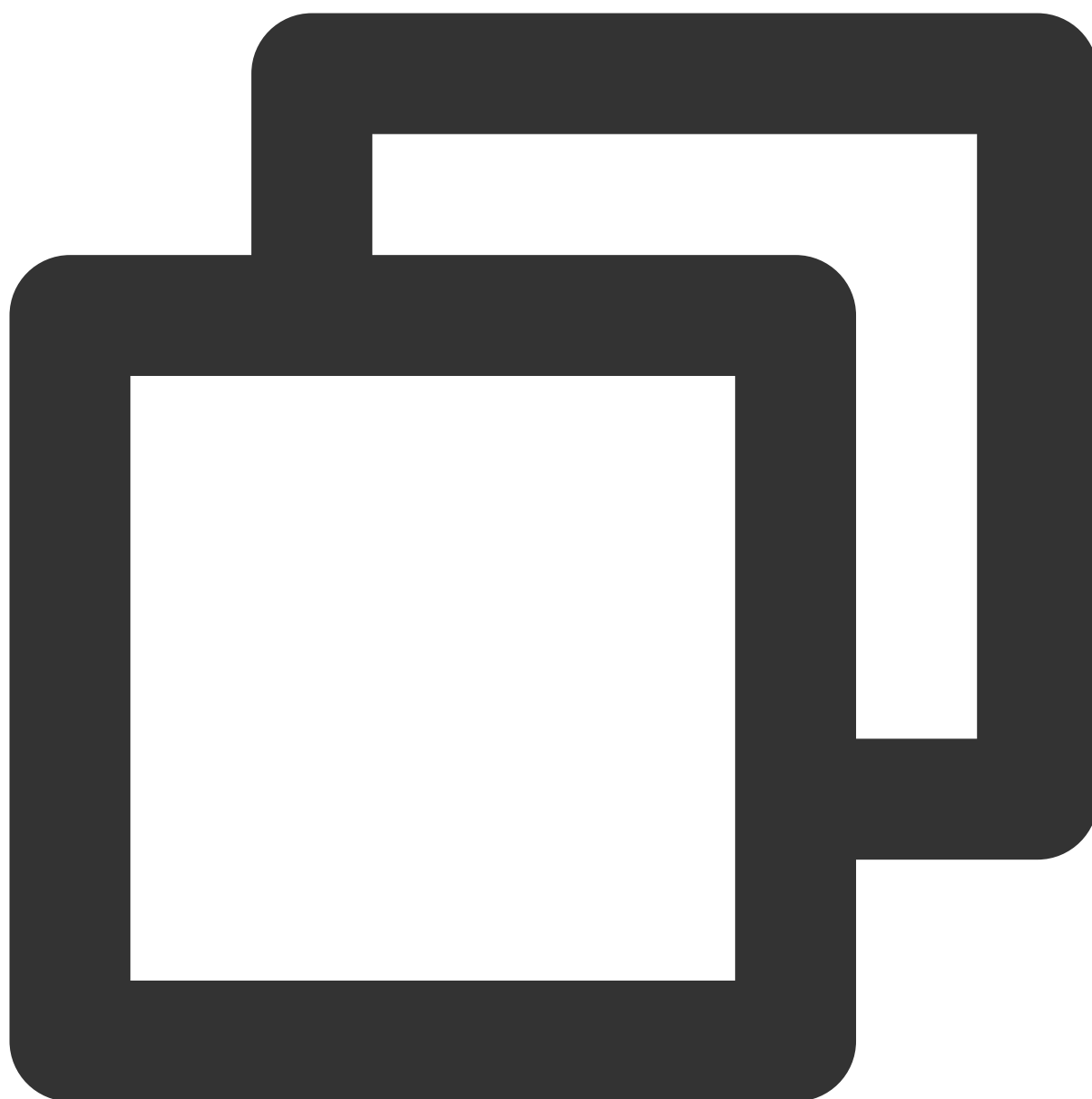
// 注册发送消息的回调
$conf->setDrMsgCb(function ($kafka, $message) {
    echo '【Producer】发送消息：message=' . var_export($message, true) . "\n";
});
// 注册发送消息错误的回调
$conf->setErrorCb(function ($kafka, $err, $reason) {
    echo "【Producer】发送消息错误：err=$err reason=$reason \n";
});

$producer = new RdKafka\Producer($conf);
// Debug 时请设置为 LOG_DEBUG
//$producer->setLogLevel(LOG_DEBUG);
$topicConf = new RdKafka\TopicConf();
$topic = $producer->newTopic($setting['topic_name'], $topicConf);
// 生产消息并发送
for ($i = 0; $i < 5; $i++) {
    // RD_KAFKA_PARTITION_UA 让 kafka 自由选择分区
    $topic->produce(RD_KAFKA_PARTITION_UA, 0, "Message $i");
    $producer->poll(0);
}

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

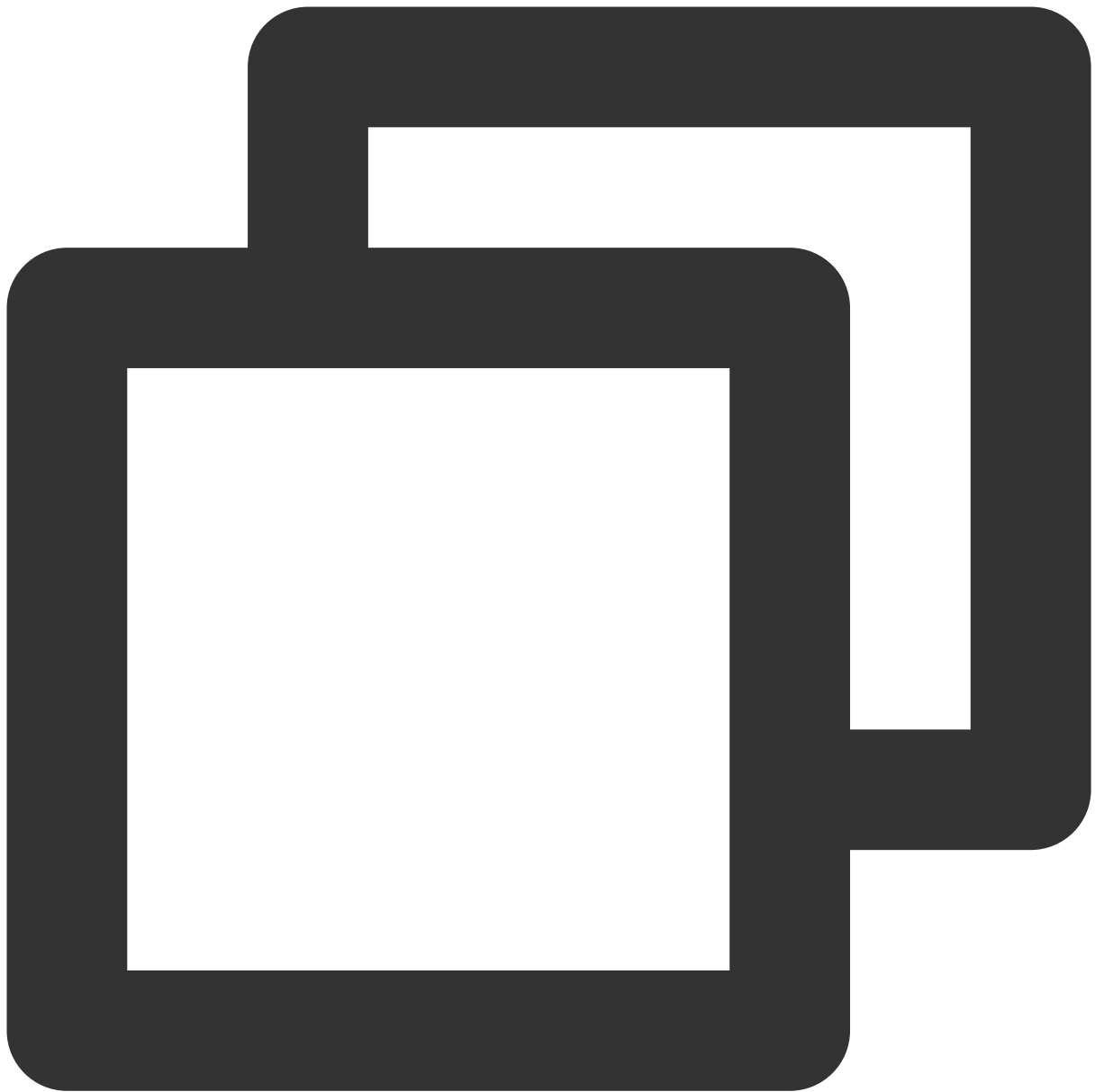
echo "【Producer】消息发送成功\n";
```

2. 运行 Producer.php 发送消息。



```
php Producer.php
```

3. 查看运行结果，示例如下。



```
> 【Producer】发送消息：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】发送消息：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】消息发送成功
```

4. 在 [CKafka 控制台](#) 的 **topic 管理** 页面，选择对应的 Topic，单击**更多 > 消息查询**，查看刚刚发送的消息。

消息查询会占用CKafka实例的带宽资源，建议您尽量缩小查询范围查询，不要频繁操作。
消息查询最多展示最近的20条消息。

实例

ckafka_instance_id

Topic

topic_name

查询类型

按位点查询

按时间查询

分区ID

0

起始位点

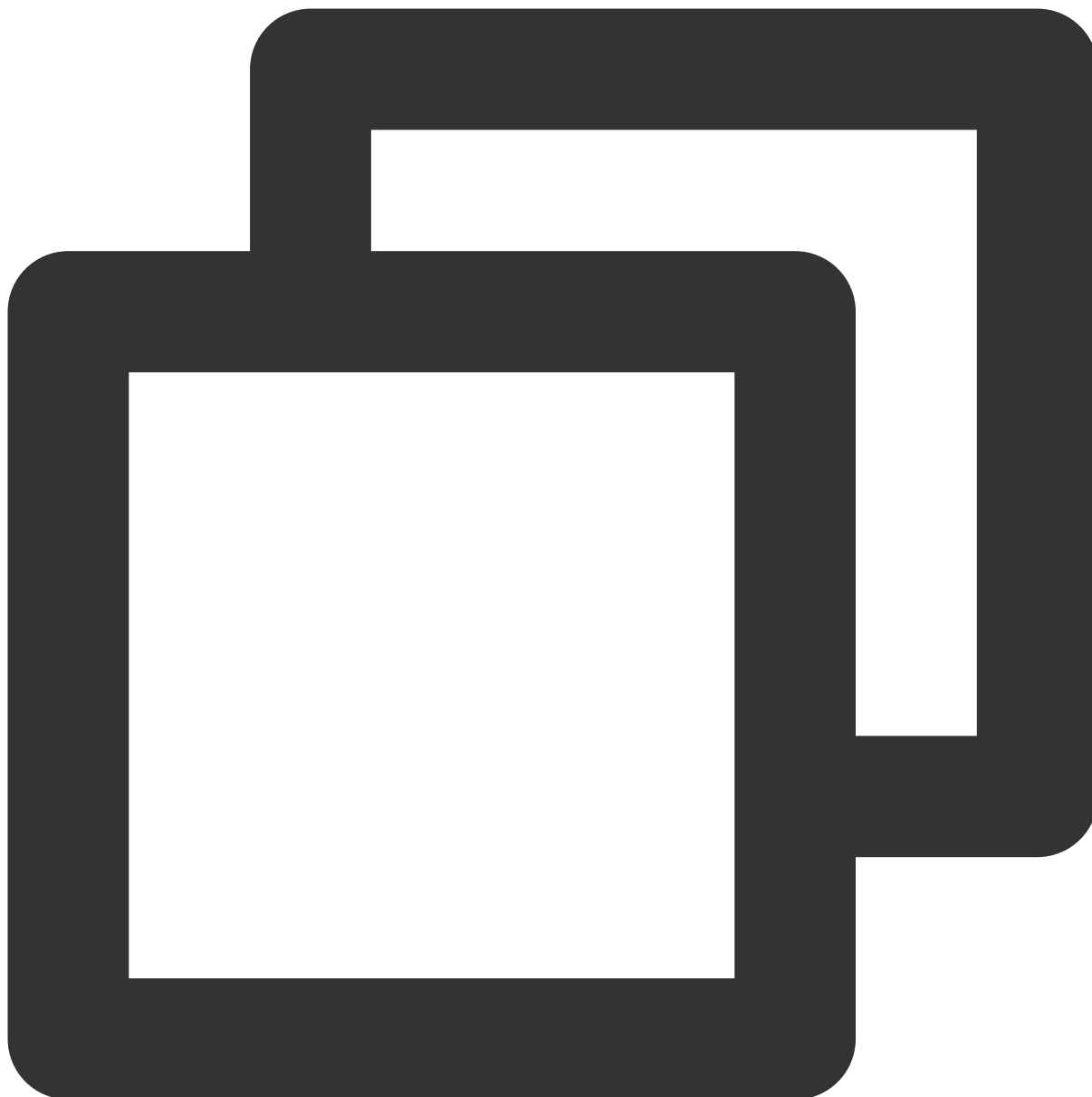
0

查询

分区ID	位点	时间戳
0	634	

步骤四：消费消息

1. 编写消息订阅消费程序 Consumer.php。



```
<?php

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

$conf = new RdKafka\\Conf();
$conf->set('group.id', $setting['group_id']);
// 设置入口服务，请通过控制台获取对应的服务地址。
$conf->set('bootstrap.servers', $setting['bootstrap_servers']);
// 使用 Kafka 消费分组机制时，消费者超时时间。当 Broker 在该时间内没有收到消费者的心跳时，
// 认为该消费者故障失败，Broker 发起重新 Rebalance 过程。
$conf->set('session.timeout.ms', 10000);
```

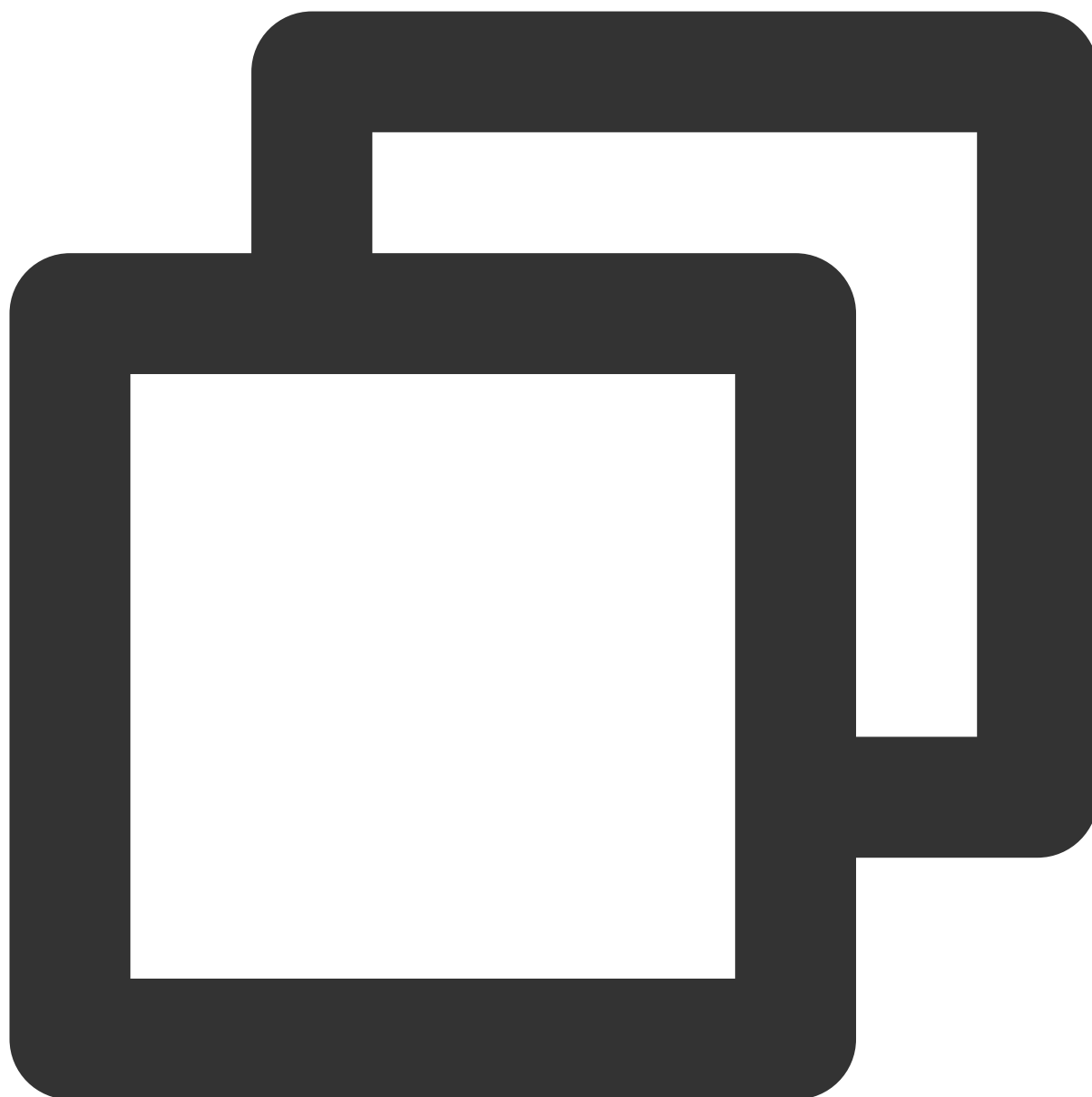
```
// 客户端请求超时时间, 如果超过这个时间没有收到应答, 则请求超时失败
$conf->set('request.timeout.ms', 305000);
// 设置客户端内部重试间隔。
$conf->set('reconnect.backoff.max.ms', 3000);

$topicConf = new RdKafka\TopicConf();
#$topicConf->set('auto.commit.interval.ms', 100);
// offset重置策略, 请根据业务场景酌情设置。设置不当可能导致数据消费缺失。
$topicConf->set('auto.offset.reset', 'earliest');
$conf->setDefaultTopicConf($topicConf);

$consumer = new RdKafka\KafkaConsumer($conf);
// Debug 时请设置为 LOG_DEBUG
//$consumer->setLogLevel(LOG_DEBUG);
$consumer->subscribe([$setting['topic_name']]);

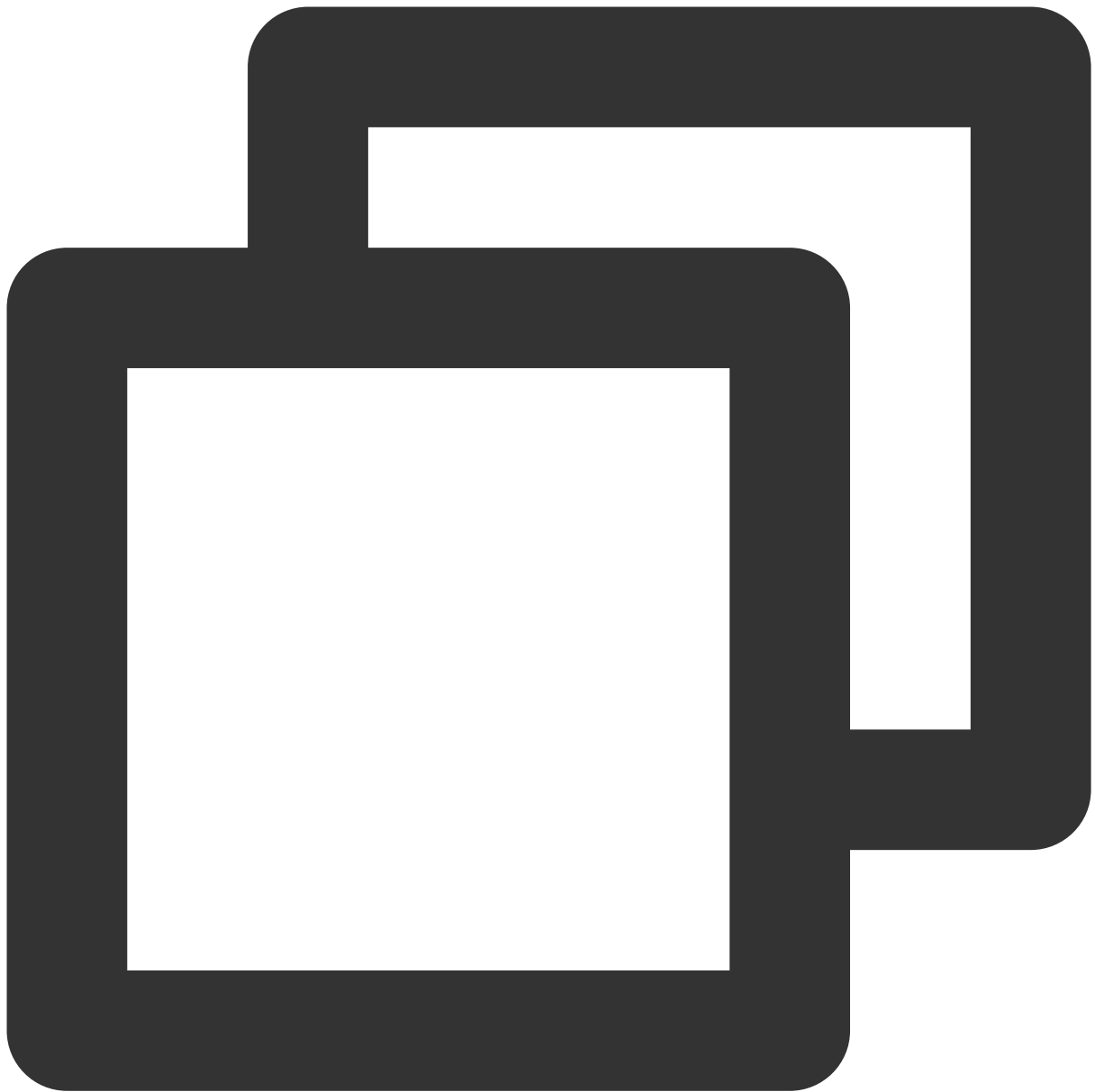
$isConsuming = true;
while ($isConsuming) {
    $message = $consumer->consume(10 * 1000);
    switch ($message->err) {
        case RD_KAFKA_RESP_ERR_NO_ERROR:
            echo "【消费者】接收到消息：" . var_export($message, true) . "\n";
            break;
        case RD_KAFKA_RESP_ERR__PARTITION_EOF:
            echo "【消费者】等待信息消息中\n";
            break;
        case RD_KAFKA_RESP_ERR__TIMED_OUT:
            echo "【消费者】等待超时\n";
            $isConsuming = false;
            break;
        default:
            throw new \Exception($message->errstr(), $message->err);
            break;
    }
}
```

2. 运行 Consumer.php 消费消息。



```
php Consumer.php
```

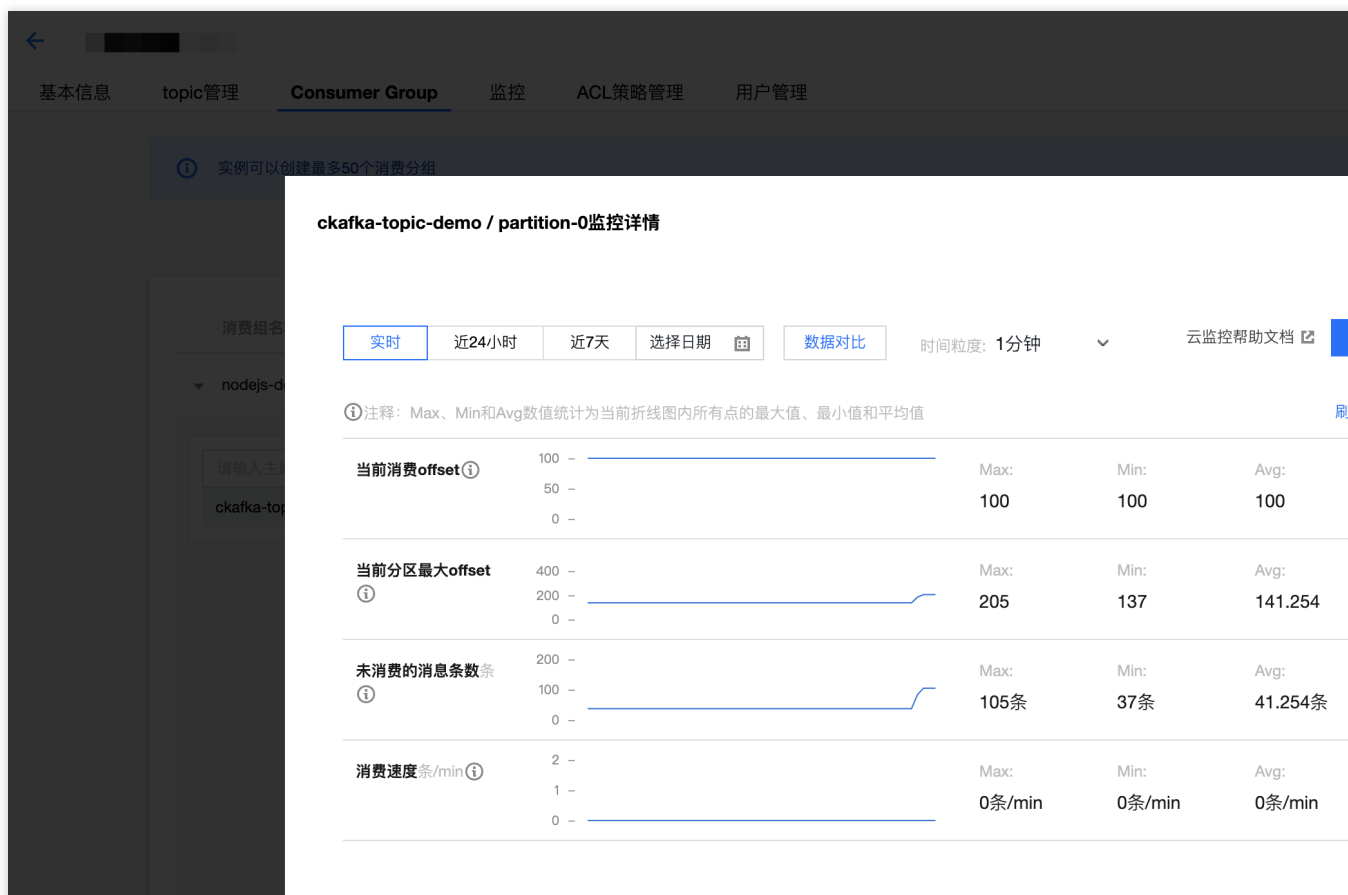
3. 查看运行结果。



```
> 【消费者】接收到消息：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,  
> ))
```

```
> 【消费者】接收到消息：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. 在 [CKafka 控制台](#) 的 **Consumer Group** 页面，选择对应的消费者组名称，在主题名称输入 Topic 名称，单击**查询详情**即可查看消费详情。



公网 SASL_PLAINTEXT 方式接入

最近更新时间：2024-05-31 14:40:49

操作场景

该任务以 PHP 客户端为例，指导您使用公网 SASL_PLAINTEXT 方式接入消息队列 CKafka 版并收发消息。

前提条件

[安装 librdkafka](#)

[安装 PHP 5.6 或以上版本](#)

[安装 PEAR](#)

[配置 ACL 策略](#)

[下载 Demo](#)

操作步骤

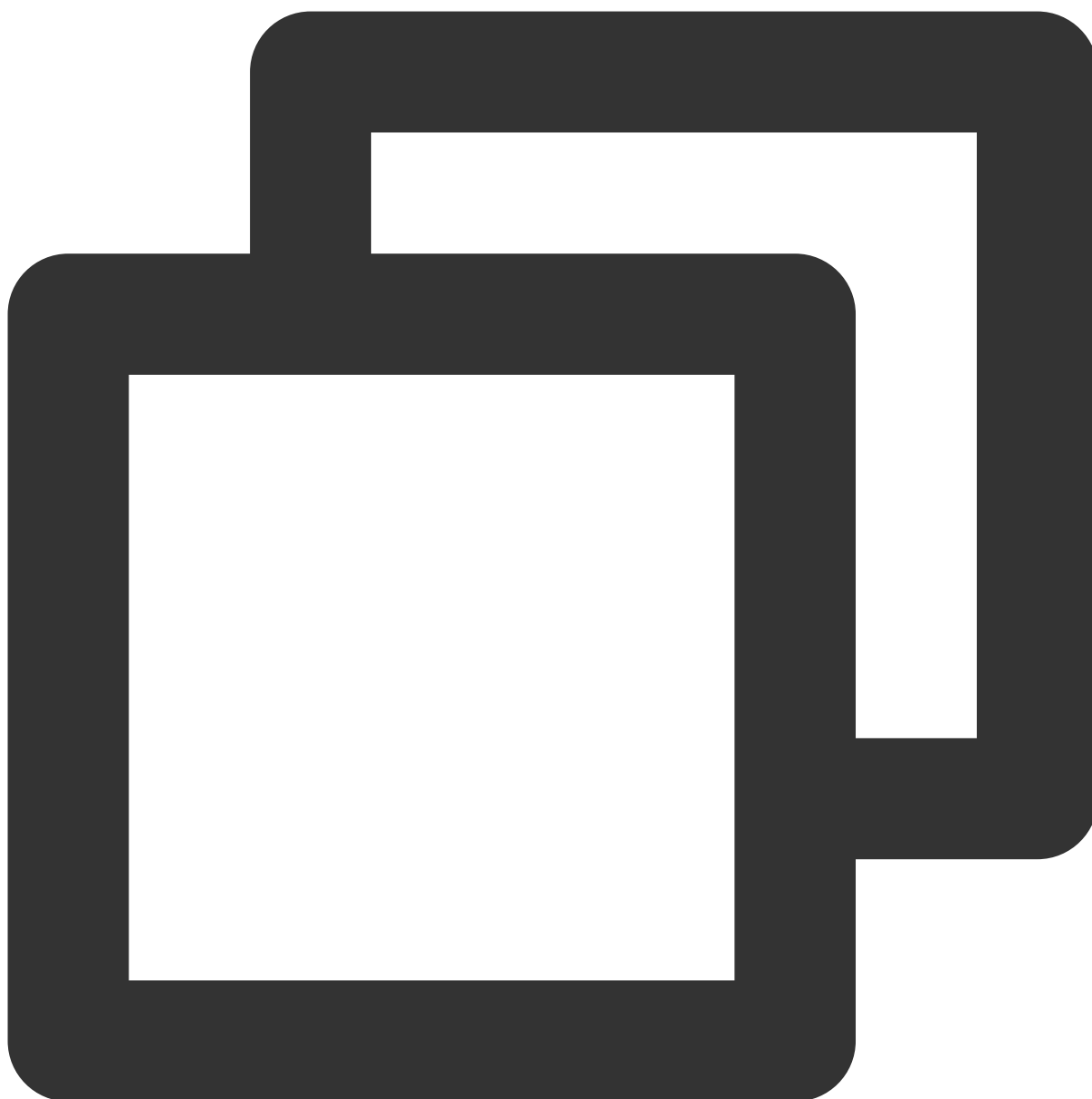
步骤一：添加 Rdkafka 扩展

1. 在 [rdkafka 官方页面](#) 查找最新的 rdkafka php 扩展包版本。

说明：

不同版本的包对 PHP 版本要求不同，这里仅以 4.1.2 为示例。

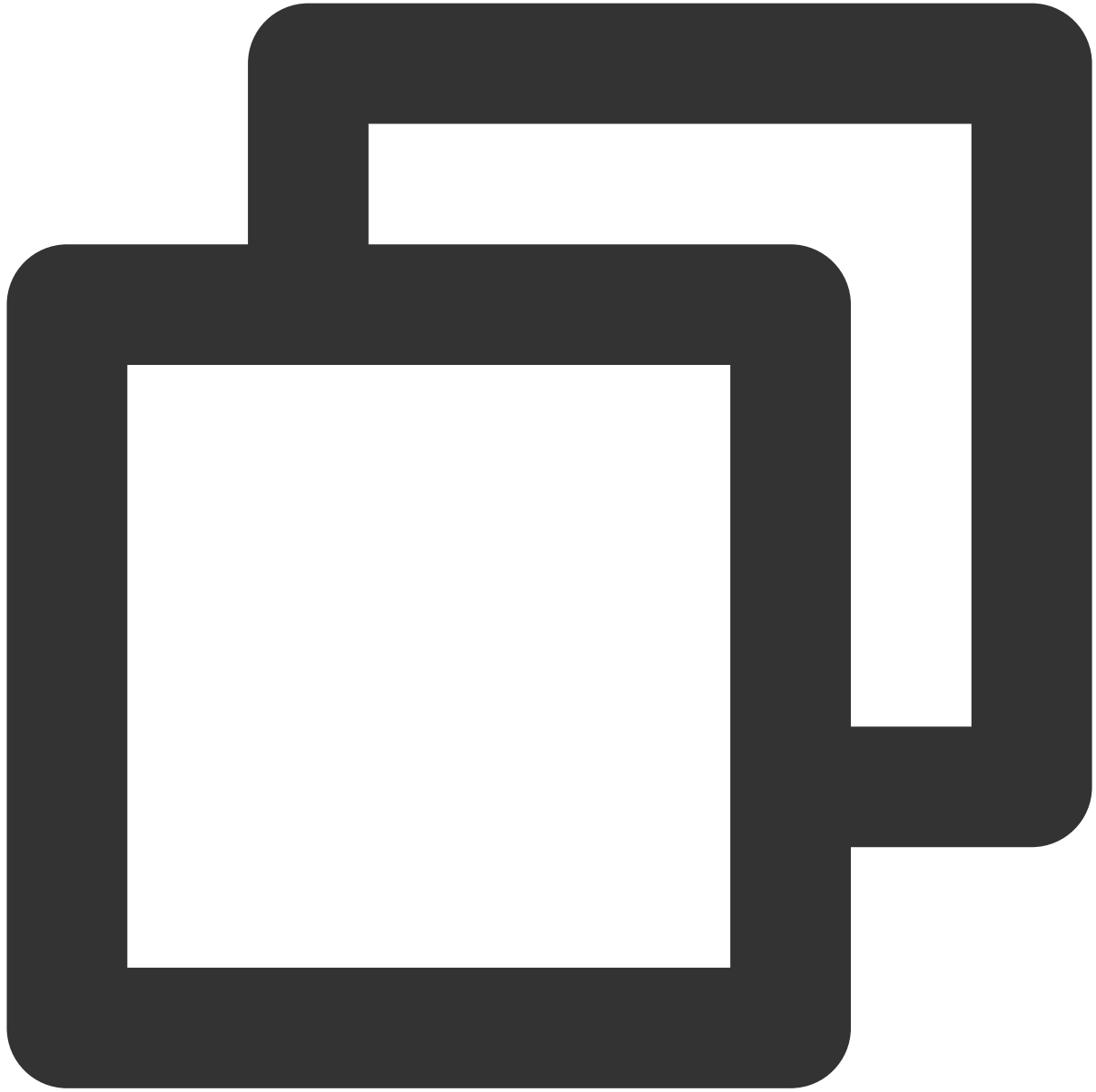
2. 安装 rdkafka 扩展。



```
wget --no-check-certificate https://pecl.php.net/get/rdkafka-4.1.2.tgz
pear install rdkafka-4.1.2.tgz
# 安装成功会提示 "install ok" 和 "You should add "extension=rdkafka.so" to php.ini"
# 如果安装失败, 若提示could not extract the package.xml file from "rdkafka-4.1.2.tgz",
# 其他错误请根据提示解决
# 安装成功后在 php.ini 添加 extension=rdkafka.so
# 执行 php --ini 后, Loaded Configuration File: 显示的就是 php.ini 所在位置
echo 'extension=rdkafka.so' >> /etc/php.ini
```

步骤二：准备配置

创建配置文件 CKafkaSetting.php。

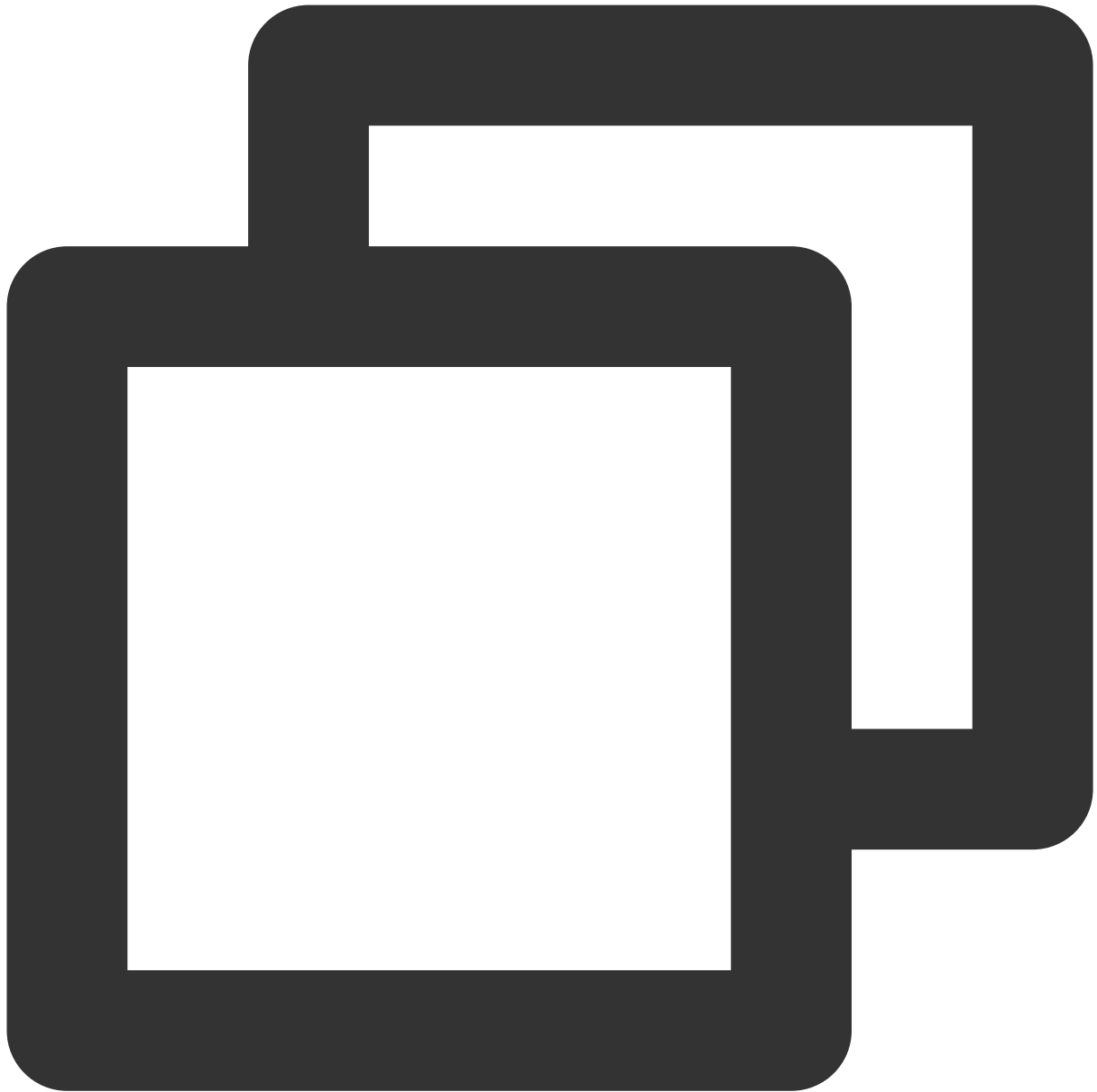


```
<?php
return [
    'bootstrap_servers' => 'bootstrap_servers1:port,bootstrap_servers2:port',
    'topic_name' => 'topic_name',
    'group_id' => 'php-demo',
    'ckafka_instance_id' => 'ckafka_instance_id',
    'sasl_username' => 'username',
    'sasl_password' => 'password'
];
```

参数	描述																		
bootstrap_servers	接入网络，在控制台的实例详情页面接入方式模块的网络列复制。 接入方式② <table><tr><td>接入类型</td><td>接入方式</td><td>网络</td><td>操作</td></tr><tr><td>公网域名接入</td><td>SASL_PLAINTEXT</td><td>ckafka- 119.29.43.52:50001</td><td>删除 查看所有</td></tr></table>	接入类型	接入方式	网络	操作	公网域名接入	SASL_PLAINTEXT	ckafka- 119.29.43.52:50001	删除 查看所有										
接入类型	接入方式	网络	操作																
公网域名接入	SASL_PLAINTEXT	ckafka- 119.29.43.52:50001	删除 查看所有																
topic_name	Topic 名称，您可以在控制台上 topic 管理页面复制。 ckafka- topic- cocs <table><tr><td>ID/名称</td><td>监控</td><td>分区数(个)</td><td>副本数(个)</td><td>标签</td><td>备注</td><td>创建时间</td><td>消息保留时间</td><td>状态</td></tr><tr><td>topic- cocs</td><td></td><td>3</td><td>2</td><td></td><td></td><td>2023-03-25 23:03:00</td><td>2 天</td><td>正常</td></tr></table>	ID/名称	监控	分区数(个)	副本数(个)	标签	备注	创建时间	消息保留时间	状态	topic- cocs		3	2			2023-03-25 23:03:00	2 天	正常
ID/名称	监控	分区数(个)	副本数(个)	标签	备注	创建时间	消息保留时间	状态											
topic- cocs		3	2			2023-03-25 23:03:00	2 天	正常											
group_id	消费者的组 Id，根据业务需求自定义，demo 运行成功后可以在 Consumer Group 页面看																		
ckafka_instance_id	实例 ID，在 CKafka 控制台的实例详情页面的基本信息获取。 基本信息topic管理Consumer Group监控ACL策略管理 基本信息 名称TEST_NEW IDckafka-i																		
sasl_username	用户名，在控制台 用户管理 页面创建用户时设置。																		
sasl_password	用户密码，在控制台 用户管理 页面创建用户时设置。																		

步骤三：发送消息

1. 编写生产消息程序 Producer.php。



```
<?php

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

$conf = new RdKafka\Conf();
// 设置入口服务，请通过控制台获取对应的服务地址。
$conf->set('bootstrap.servers', $setting['bootstrap_servers']);
// ----- 启用 SASL 验证时需要设置 -----
// SASL 验证机制类型默认选用 PLAIN
$conf->set('sasl.mechanism', 'PLAIN');
// 设置用户名：实例 ID + # + 【用户管理】中配置的用户名
```

```
$conf->set('sasl.username', $setting['ckafka_instance_id'] . '#' . $setting['sasl_u
// 设置密码：【用户管理】中配置的密码
$conf->set('sasl.password', $setting['sasl_password']);
// 在本地配置 ACL 策略。
$conf->set('security.protocol', 'SASL_PLAINTEXT');
// ----- 启用 SASL 验证时需要设置 -----
// Kafka producer 的 ack 有 3 种机制，分别说明如下：
// -1 或 all:Broker 在 leader 收到数据并同步给所有 ISR 中的 follower 后，才应答给 Produce
// 这种配置提供了最高的数据可靠性，只要有一个已同步的副本存活就不会有消息丢失。注意：这种配置不能确
// 可以配合 Topic 级别参数 min.insync.replicas 使用。
// 0：生产者不等待来自 broker 同步完成的确认，继续发送下一条（批）消息。这种配置生产性能最高，但类
// （当服务器故障时可能会有数据丢失，如果 leader 已死但是 producer 不知情，则 broker 收不到消息
// 1：生产者在 leader 已成功收到的数据并得到确认后发送下一条（批）消息。这种配置是在生产吞吐和
// （如果leader已死但是尚未复制，则消息可能丢失）
// 用户不显示配置时，默认值为1。用户根据自己的业务情况进行设置
$conf->set('acks', '1');
// 请求发生错误时重试次数，建议将该值设置为大于0，失败重试最大程度保证消息不丢失
$conf->set('retries', '0');
// 发送请求失败时到下一次重试请求之间的时间
$conf->set('retry.backoff.ms', 100);
// producer 网络请求的超时时间。
$conf->set('socket.timeout.ms', 6000);
$conf->set('reconnect.backoff.max.ms', 3000);

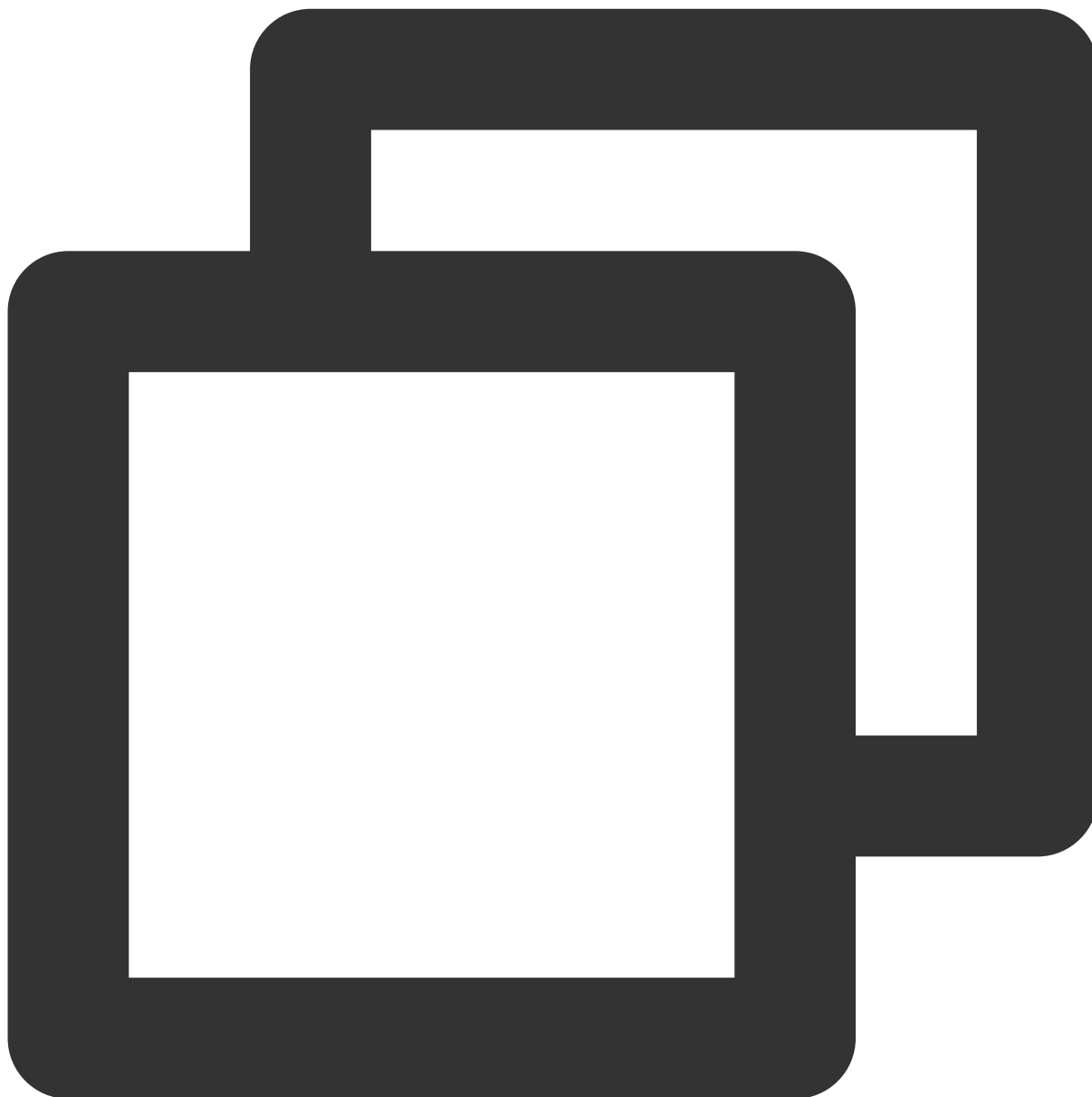
// 注册发送消息的回调
$conf->setDrMsgCb(function ($kafka, $message) {
    echo '【Producer】发送消息:message=' . var_export($message, true) . "\n";
});
// 注册发送消息错误的回调
$conf->setErrorCb(function ($kafka, $err, $reason) {
    echo "【Producer】发送消息错误:err=$err reason=$reason \n";
});

$producer = new RdKafka\Producer($conf);
// Debug 时请设置为 LOG_DEBUG
//$producer->setLogLevel(LOG_DEBUG);
$topicConf = new RdKafka\TopicConf();
$topic = $producer->newTopic($setting['topic_name'], $topicConf);
// 生产消息并发送
for ($i = 0; $i < 5; $i++) {
    // RD_KAFKA_PARTITION_UA 让 kafka 自由选择分区
    $topic->produce(RD_KAFKA_PARTITION_UA, 0, "Message $i");
    $producer->poll(0);
}

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

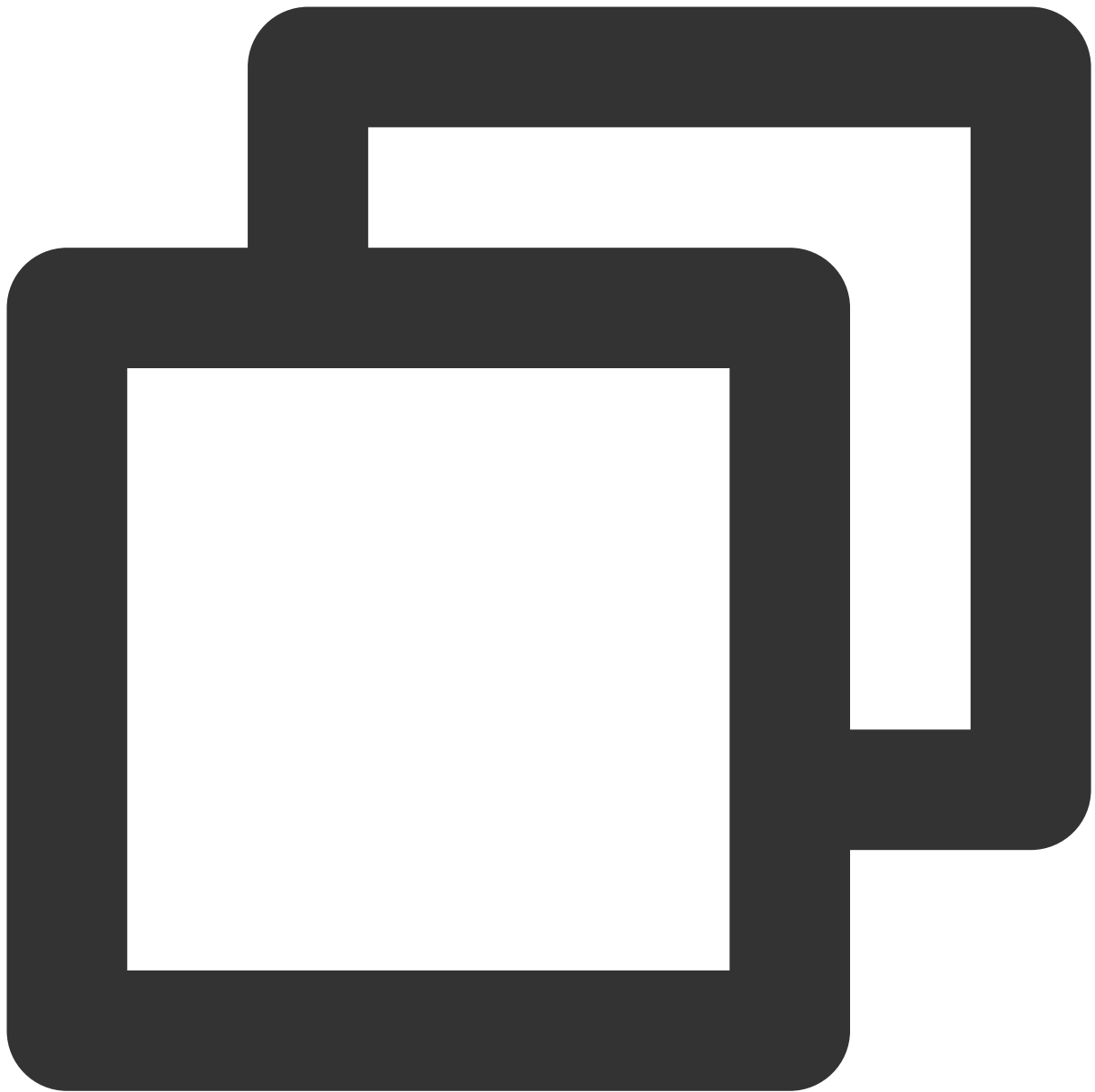
```
}  
  
echo "【Producer】消息发送成功\\n";
```

2. 运行 Producer.php 发送消息。



```
php Producer.php
```

3. 查看运行结果。



```
> 【Producer】发送消息：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】发送消息：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】消息发送成功
```

4. 在 [CKafka 控制台](#) 的 **topic 管理** 页面，选择对应的 Topic，单击**更多 > 消息查询**，查看刚刚发送的消息。

消息查询会占用CKafka实例的带宽资源，建议您尽量缩小查询范围查询，不要频繁操作。
消息查询最多展示最近的20条消息。

实例

ckafka_instance_id

Topic

topic_name

查询类型

按位点查询

按时间查询

分区ID

0

起始位点

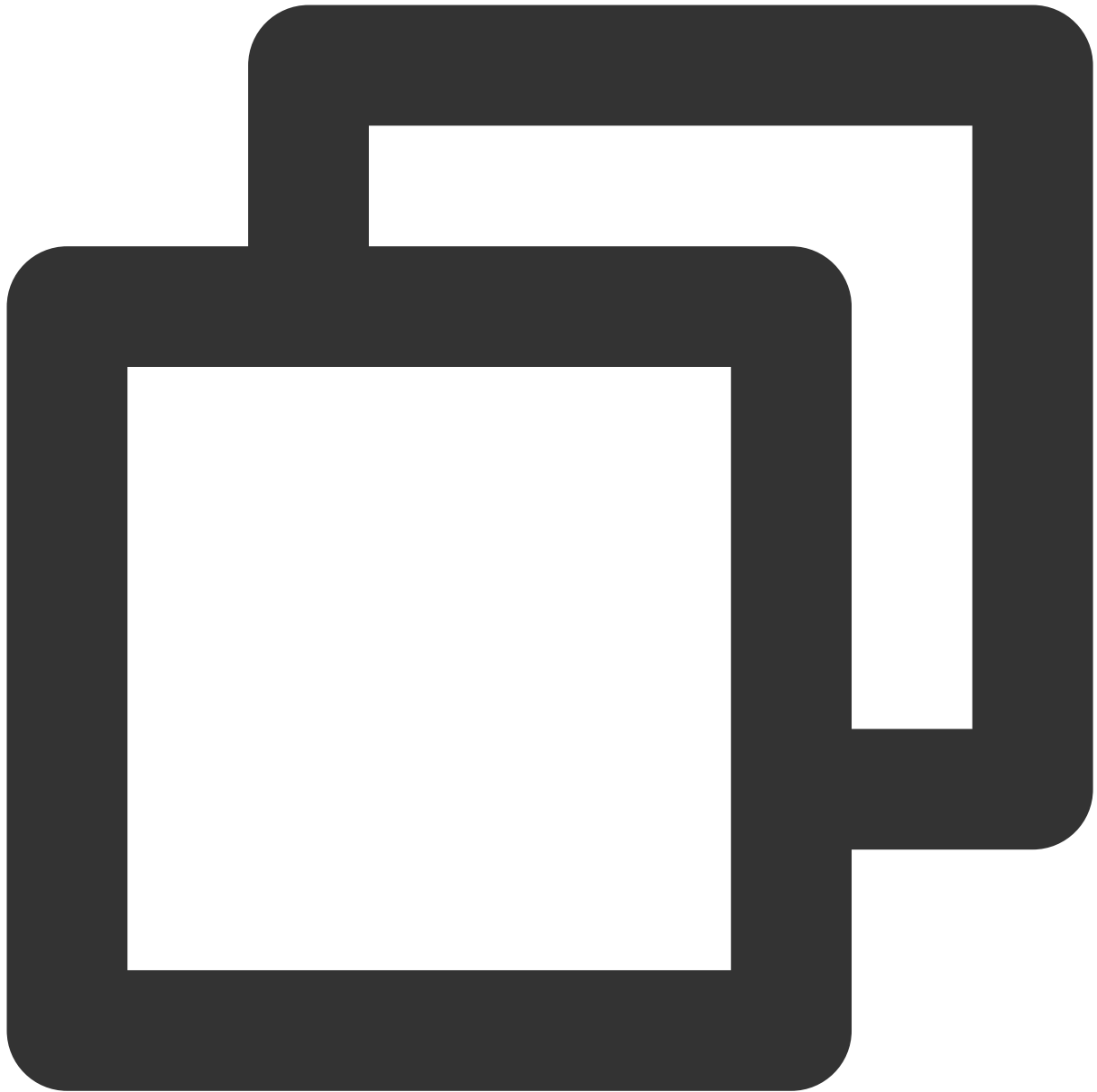
0

查询

分区ID	位点	时间戳
0	634	

步骤四：消费消息

1. 编写消息订阅消费程序 Consumer.php。



```
<?php

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

$conf = new RdKafka\\Conf();
$conf->set('group.id', $setting['group_id']);
// 设置入口服务，请通过控制台获取对应的服务地址。
$conf->set('bootstrap.servers', $setting['bootstrap_servers']);
// ----- 启用 SASL 验证时需要设置 -----
// SASL 验证机制类型默认选用 PLAIN
$conf->set('sasl.mechanism', 'PLAIN');
```

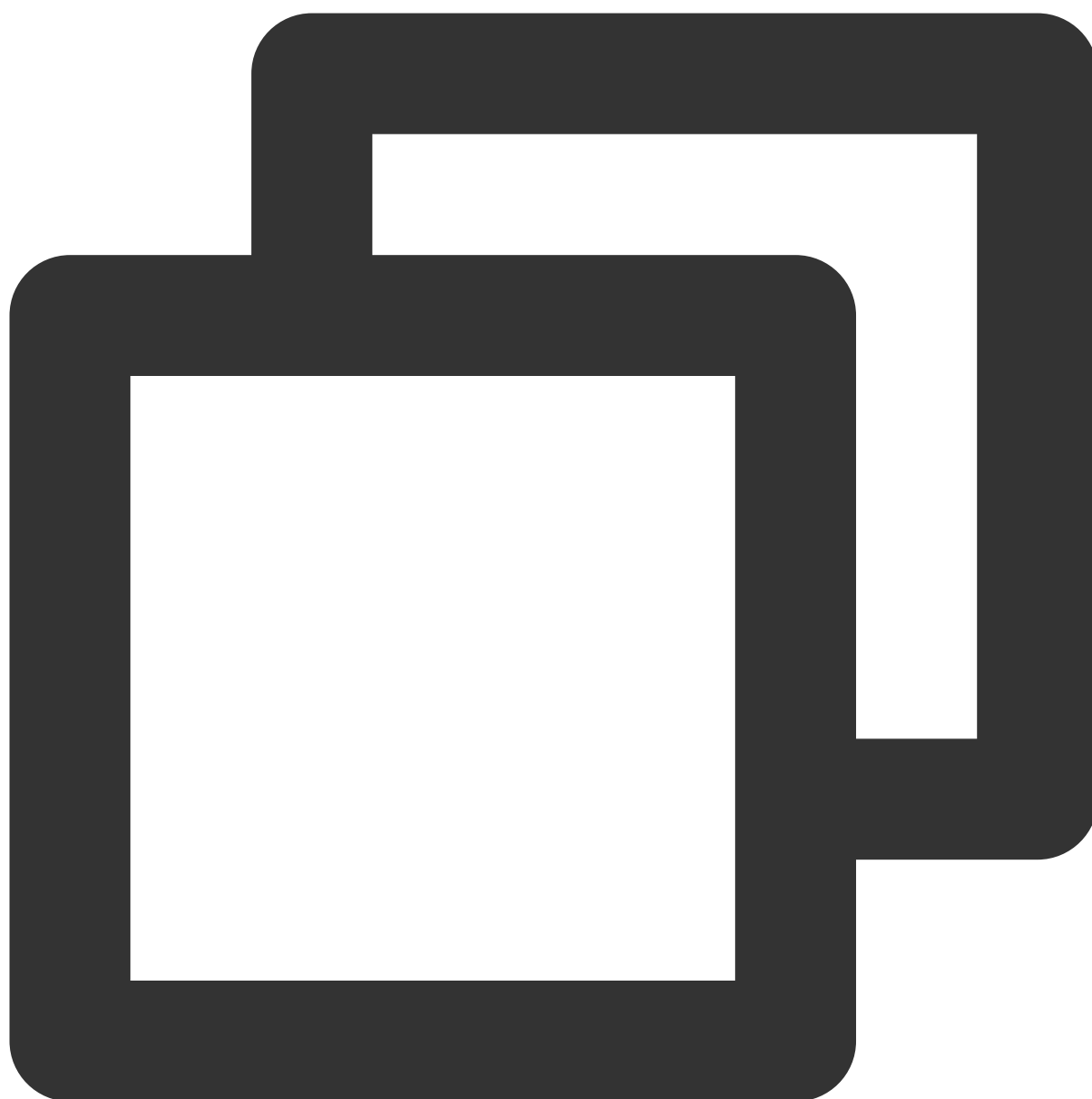
```
// 设置用户名：实例 ID + # + 【用户管理】中配置的用户名
$conf->set('sasl.username', $setting['ckafka_instance_id'] . '#' . $setting['sasl_u
// 设置密码：【用户管理】中配置的密码
$conf->set('sasl.password', $setting['sasl_password']);
// 在本地配置 ACL 策略。
$conf->set('security.protocol', 'SASL_PLAINTEXT');
// ----- 启用 SASL 验证时需要设置 -----
// 使用 Kafka 消费分组机制时，消费者超时时间。当 Broker 在该时间内没有收到消费者的心跳时，
// 认为该消费者故障失败，Broker 发起重新 Rebalance 过程。
$conf->set('session.timeout.ms', 10000);
// 客户端请求超时时间，如果超过这个时间没有收到应答，则请求超时失败
$conf->set('request.timeout.ms', 305000);
// 设置客户端内部重试间隔。
$conf->set('reconnect.backoff.max.ms', 3000);

$topicConf = new RdKafka\TopicConf();
#$topicConf->set('auto.commit.interval.ms', 100);
// offset重置策略，请根据业务场景酌情设置。设置不当可能导致数据消费缺失。
$topicConf->set('auto.offset.reset', 'earliest');
$conf->setDefaultTopicConf($topicConf);

$consumer = new RdKafka\KafkaConsumer($conf);
// Debug 时请设置为 LOG_DEBUG
//$consumer->setLogLevel(LOG_DEBUG);
$consumer->subscribe([$setting['topic_name']]);

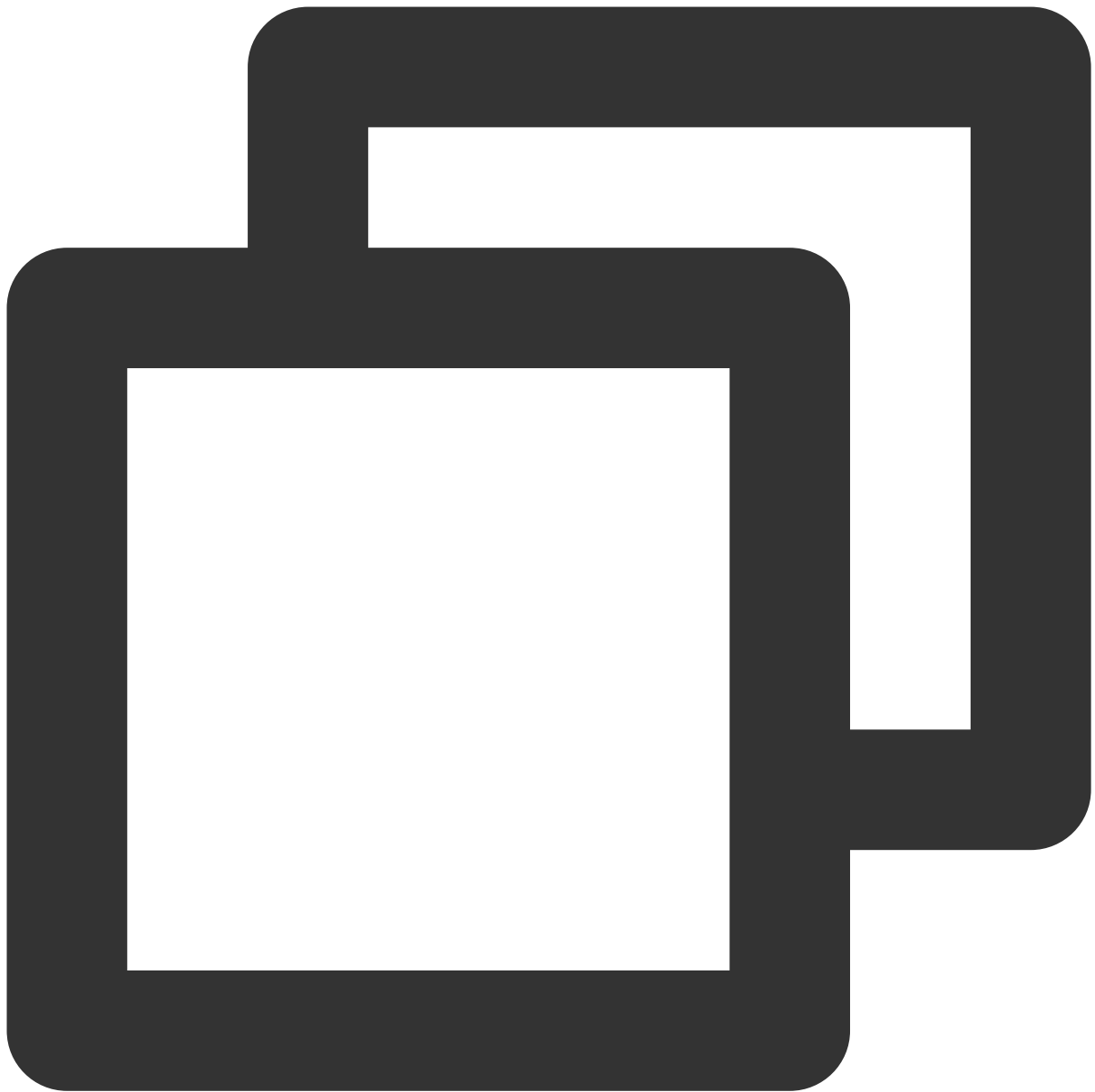
$isConsuming = true;
while ($isConsuming) {
    $message = $consumer->consume(10 * 1000);
    switch ($message->err) {
        case RD_KAFKA_RESP_ERR_NO_ERROR:
            echo "【消费者】接收到消息：" . var_export($message, true) . "\n";
            break;
        case RD_KAFKA_RESP_ERR__PARTITION_EOF:
            echo "【消费者】等待信息消息中\n";
            break;
        case RD_KAFKA_RESP_ERR__TIMED_OUT:
            echo "【消费者】等待超时\n";
            $isConsuming = false;
            break;
        default:
            throw new \Exception($message->errstr(), $message->err);
            break;
    }
}
```

2. 运行 Consumer.php 消费消息。



```
php Consumer.php
```

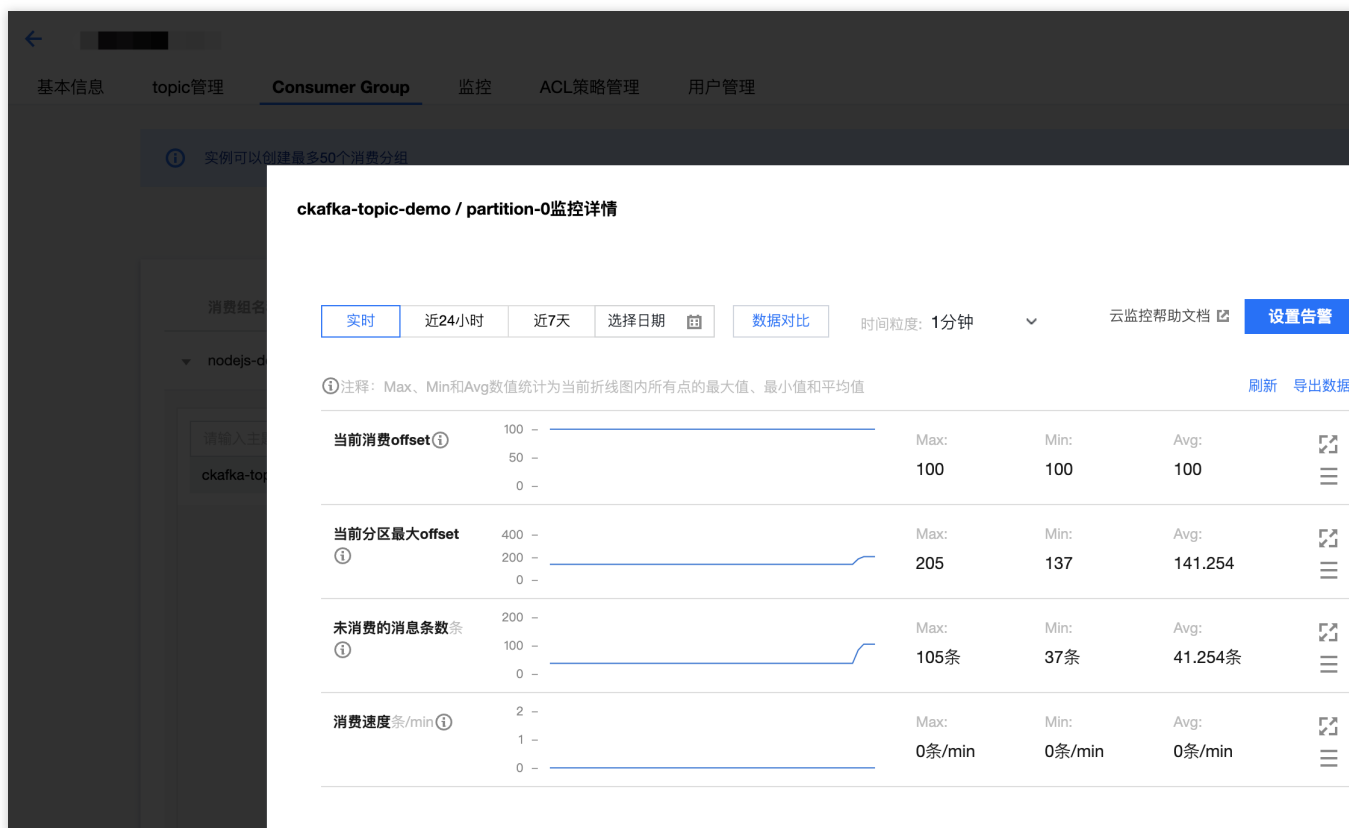
3. 查看运行结果。



```
> 【消费者】接收到消息：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,  
> ))
```

```
> 【消费者】接收到消息：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. 在 [CKafka 控制台](#) 的 **Consumer Group** 页面，选择对应的消费者组，在主题名称输入 Topic 名称，单击[查询详情](#)即可查看消费详情。



C++ SDK

VPC 网络接入

最近更新时间：2024-05-31 14:40:49

操作场景

该任务以 C++ 客户端为例指导您使用 VPC 网络接入消息队列 CKafka 版并收发消息。

前提条件

[安装 GCC](#)

[下载 Demo](#)

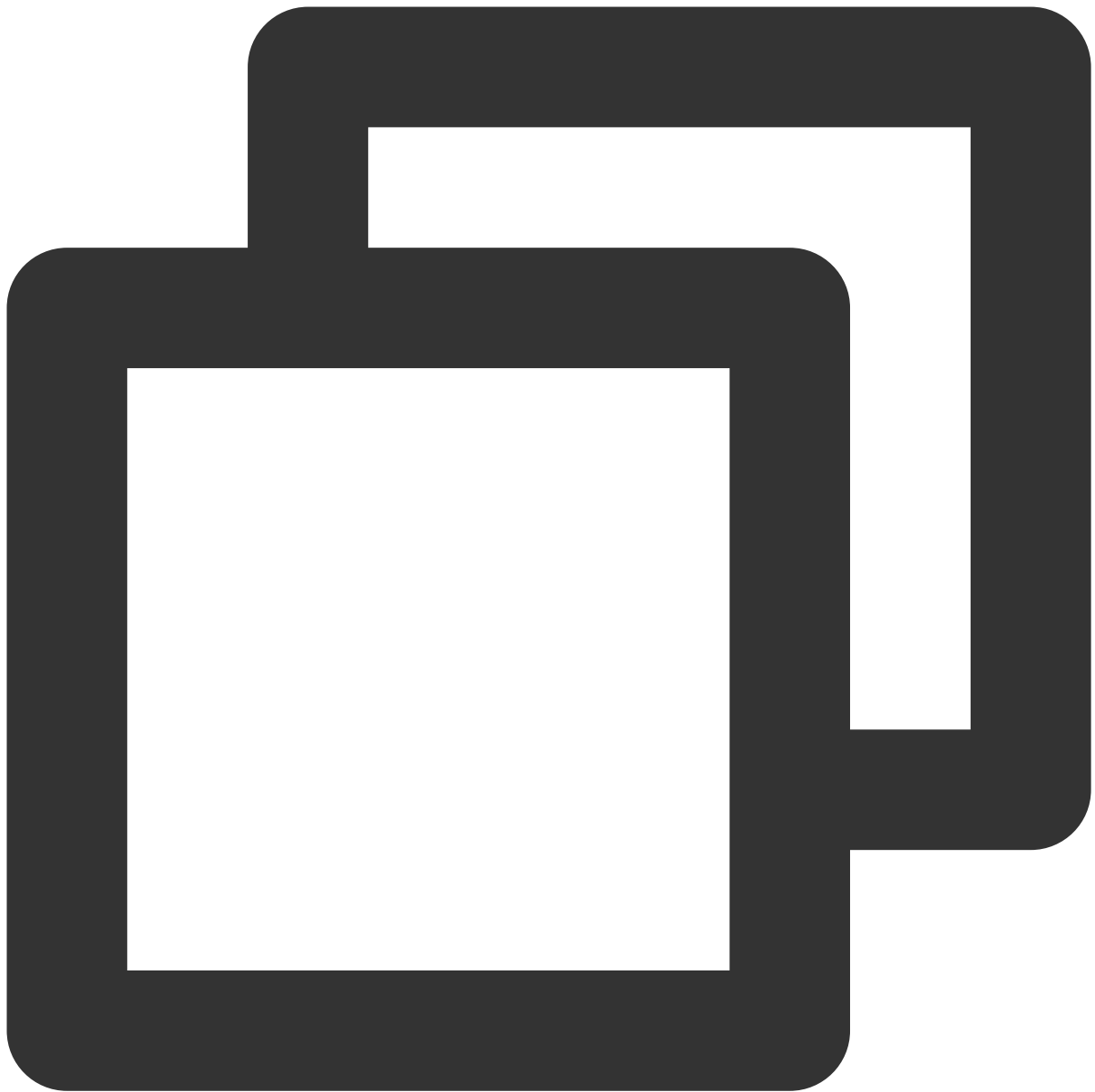
操作步骤

步骤一：安装 C/C++ 依赖库

1. 将下载的 Demo 中的 cppkfkademo 上传至 Linux 服务器。
2. 登录 Linux 服务器，安装 [librdkafka](#)。

步骤二：发送消息

1. 创建 producer.c 文件。



```
/*
 * 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, "%s Message delivery failed: %s\\n",
                rd_kafka_err2str(rkmessage->err));
    else
        fprintf(stderr,
                "%s Message delivered (%zd bytes, "
                "partition %"PRIu32")\\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, "%s 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, sizeof(errstr)) != RD_KAFKA_CONF_OK) {
        fprintf(stderr, "%s\\n", errstr);
        return 1;
    }
    if(rd_kafka_conf_set(conf, "sasl.mechanisms", "PLAIN", errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK) {
        fprintf(stderr, "%s\\n", errstr);
        return 1;
    }
    if(rd_kafka_conf_set(conf, "sasl.username", user, errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK) {
        fprintf(stderr, "%s\\n", errstr);
        return 1;
    }
    if(rd_kafka_conf_set(conf, "sasl.password", passwd, errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK) {
        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_CONF_OK) {
//    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)) != RD_KAFKA_CONF_OK) {
        fprintf(stderr, "%s\\n", errstr);
        return 1;
    }
    if(rd_kafka_conf_set(conf, "retries", "3", errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK) {
        fprintf(stderr, "%s\\n", errstr);
        return 1;
    }

    if(rd_kafka_conf_set(conf, "retry.backoff.ms", "1000", errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK) {
        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,
                "%s: Failed to create new producer: %s\\n", errstr);
        return 1;
    }

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

    fprintf(stderr,
            "%s Type some text and hit enter to produce message\\n"
            "%s Or just hit enter to only serve delivery reports\\n"
            "%s 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,
            "%s 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, "%s 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, "%s Flushing final messages..\\n");
rd_kafka_flush(rk, 10*1000 /* wait for max 10 seconds */);

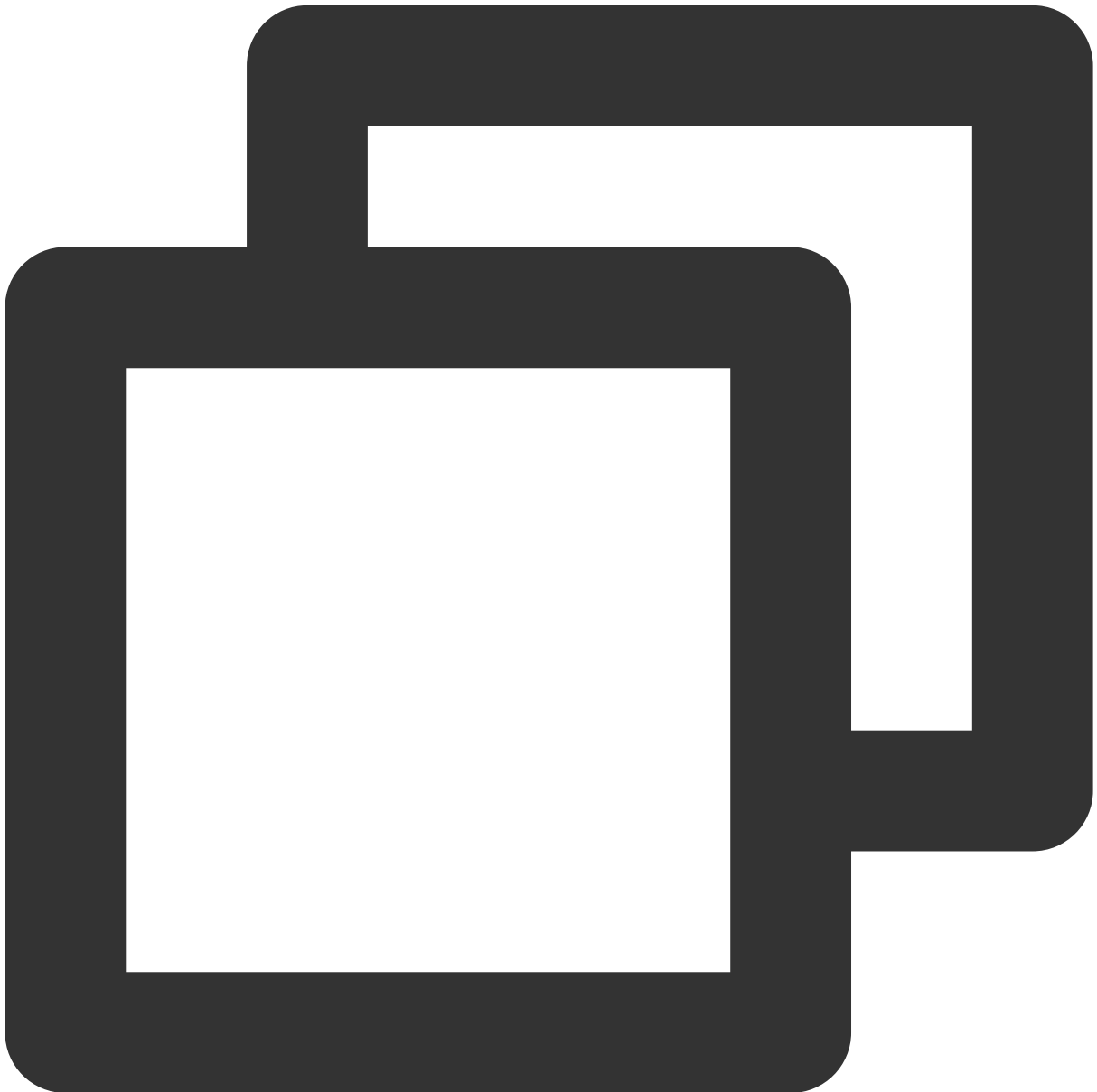
/* If the output queue is still not empty there is an issue
 * with producing messages to the clusters. */
if (rd_kafka_outq_len(rk) > 0)
    fprintf(stderr, "%s %d message(s) were not delivered\\n",
```

```
rd_kafka_outq_len(rk));

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

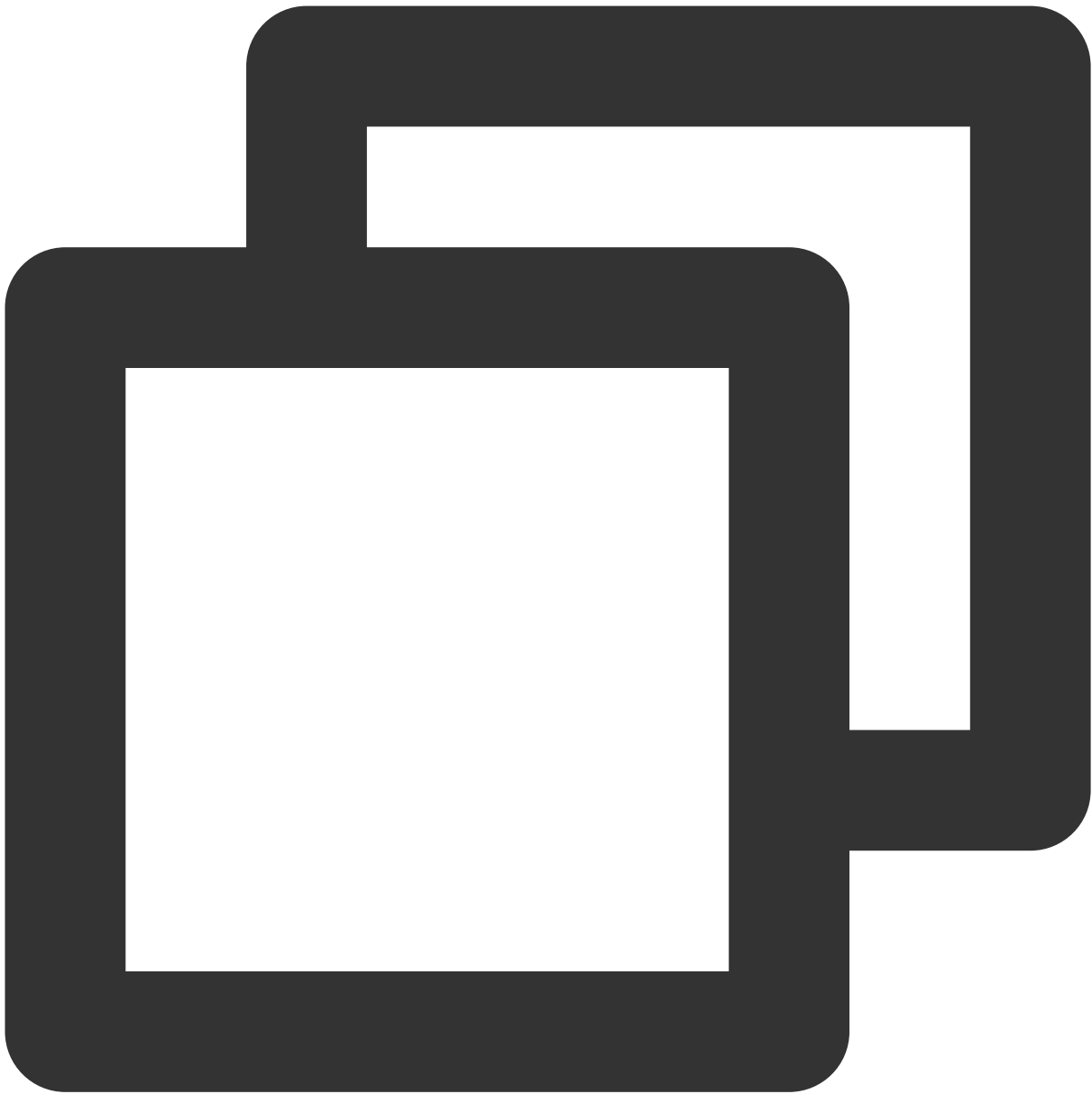
return 0;
}
```

2. 执行以下命令编译 `producer.c`。



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

3. 执行以下命令发送消息。



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

参数	描述
broker	接入网络，在控制台的实例详情页面接入方式模块的网络列复制。

topic

接入方式②

添加路由策略

接入类型	接入方式	网络	操作
VPC网络	PLAINTEXT	vpc-l- subnet-5gl6yj86 10.0.0.0/24 10.0.0.0/24	删除 查看所有IP和端口

Topic 名称，您可以在控制台上 **topic 管理** 页面复制。

ckafka-plain

基本信息

topic管理

Consumer Group

监控

事件中心

HTTP接入

ACL策略管理

智能运维 专业版

集群备份 专业版

新建(24/450)

请输入TopicId或名称

ID/名称	监控	分区数(个)	副本数(个)	标签	备注	创建时间	消息保留时间	状态	操作
topic-l- cocs		3	2			2023-03-25 23:03:00	2 天	正常	编辑 删除 更多

运行结果如下：

```
[root@VM_1_28_centos ~]# ./produce 7779 test
% 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. 在 Ckafka 控制台 **topic 管理** 页面，选择对应的 Topic，单击**更多 > 消息查询**，查看刚刚发送的消息。

实例

ckafka_instance_id

Topic

topic_name

查询类型

按位点查询

按起始时间查询

分区ID

请输入分区ID

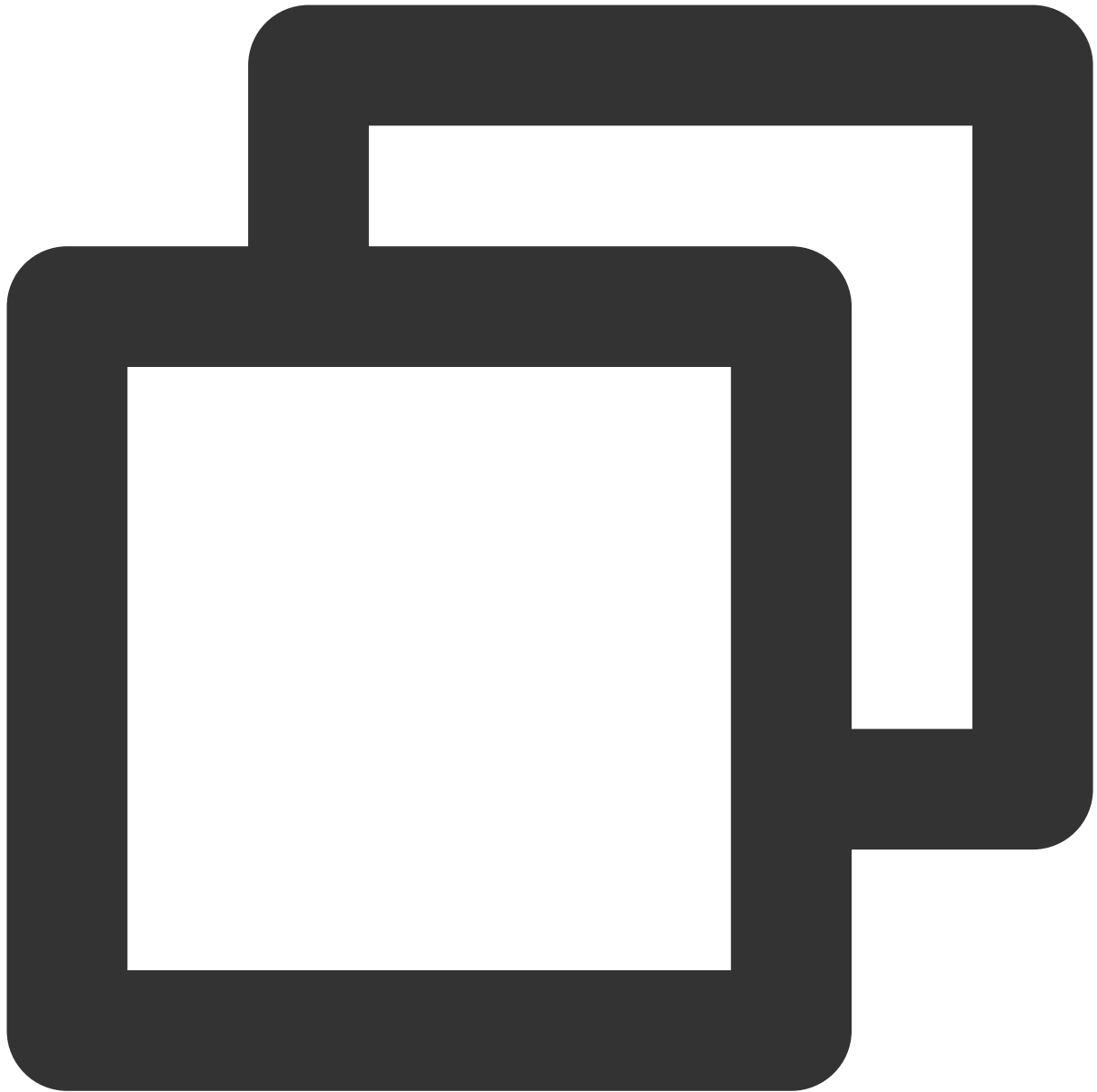
起始位点

请输入起始位点

查询

步骤三：消费消息

1. 创建 consumer.c 文件。



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

        /* Proper message. */
        printf("Message on %s [%s] at offset %s:\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);

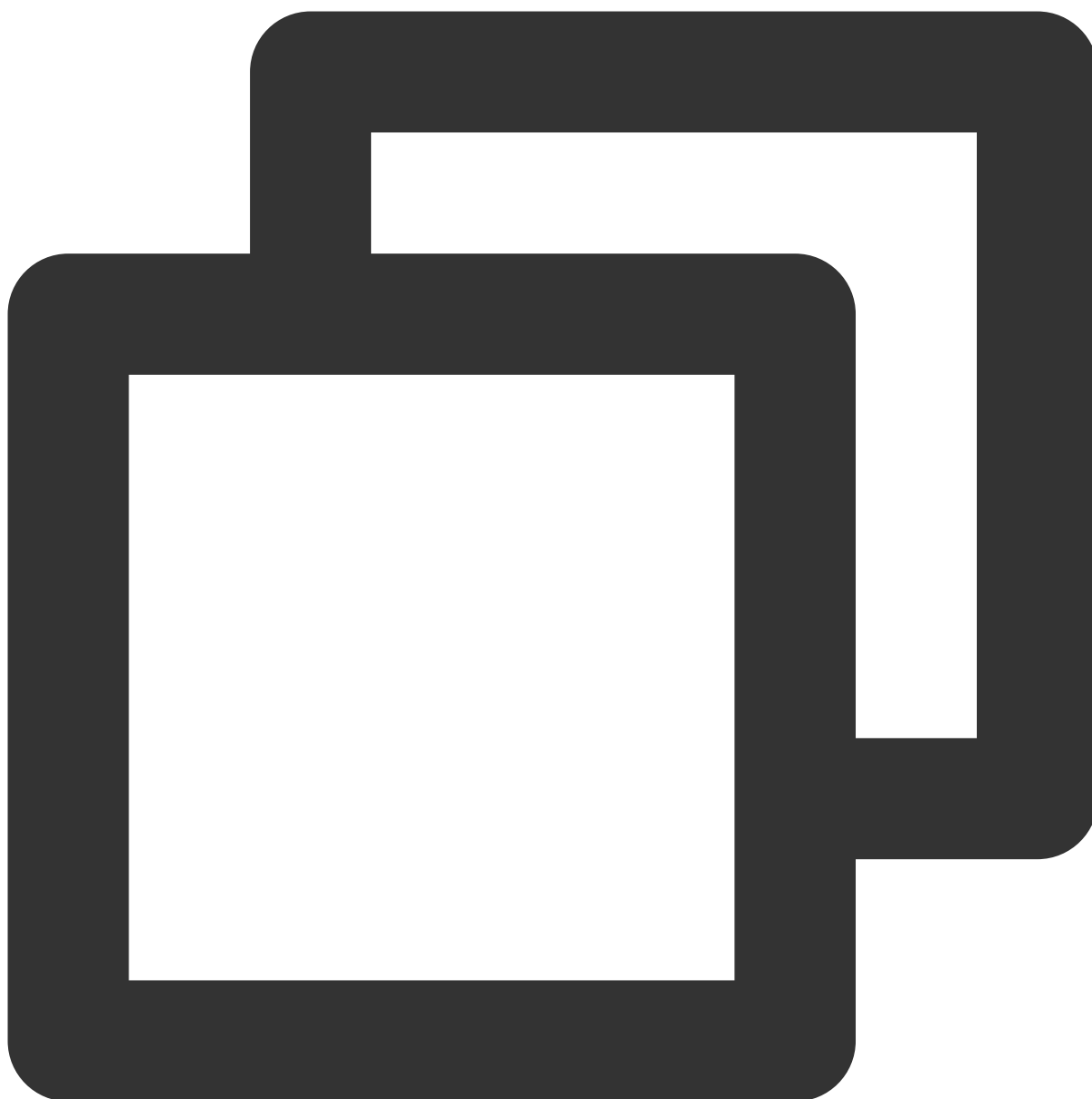
rd_kafka_message_destroy(rkm);
}

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

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

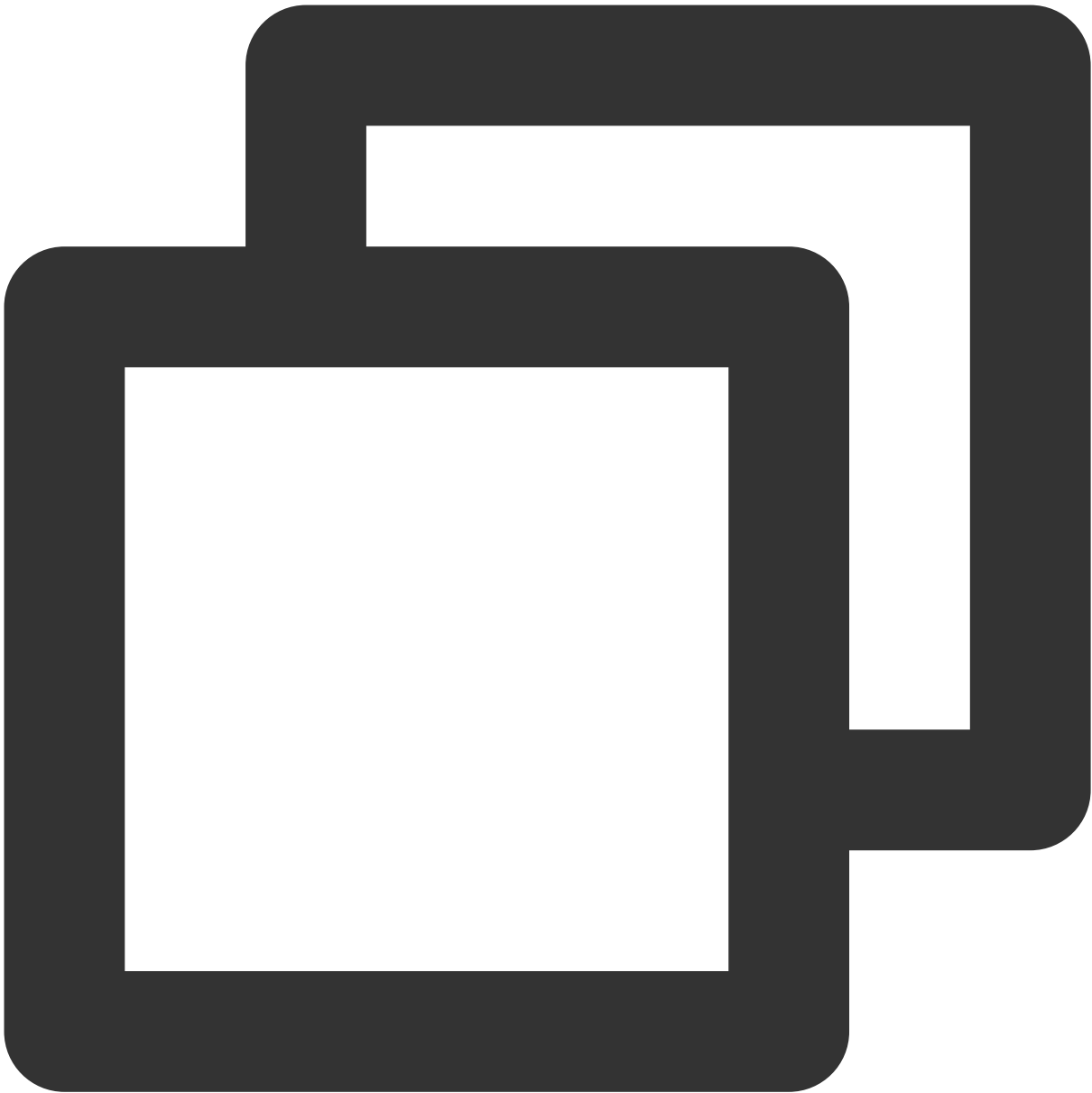
return 0;
}
```

2. 执行以下命令编译 consumer.c。



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

3. 执行以下命令发送消息。



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

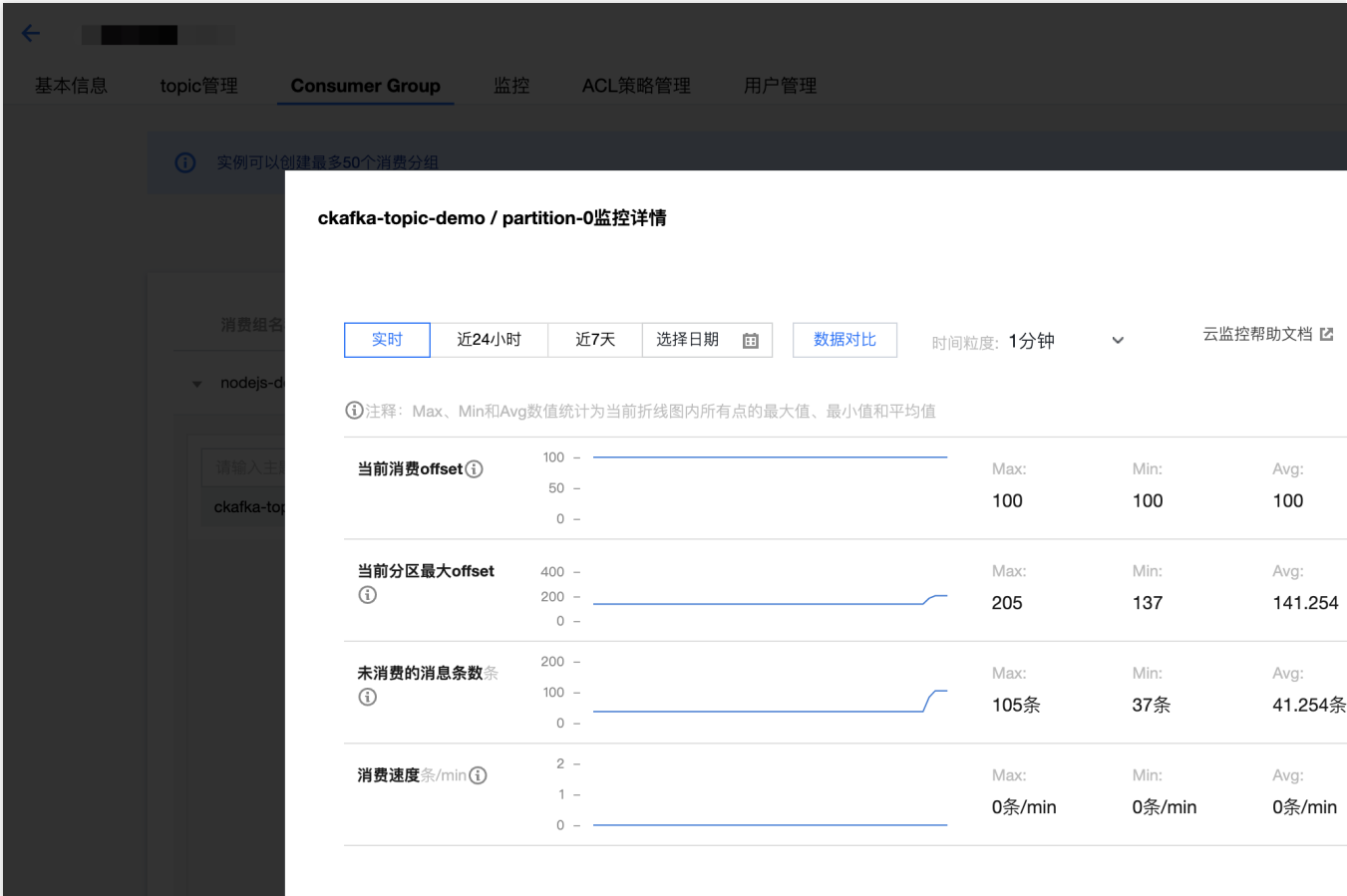
参数	描述
broker	接入网络，在控制台的实例详情页面接入方式模块的网络列复制。

	接入方式② 添加路由策略 <table><tr><th>接入类型</th><th>接入方式</th><th>网络</th><th>操作</th></tr><tr><td>VPC网络</td><td>PLAINTEXT</td><td>vpc-l subnet-5gl6yj86 1.....0092 </td><td>删除 查看所有IP和端口</td></tr></table>	接入类型	接入方式	网络	操作	VPC网络	PLAINTEXT	vpc-l subnet-5gl6yj86 1.....0092 	删除 查看所有IP和端口												
接入类型	接入方式	网络	操作																		
VPC网络	PLAINTEXT	vpc-l subnet-5gl6yj86 1.....0092 	删除 查看所有IP和端口																		
group.id	消费分组名称，您可以自定义设置，Demo 运行成功后可以在 Consumer Group 页面看到该消费者。																				
topic1 topic2..	Topic 名称，您可以在控制台上 topic 管理 页面复制。 ckafka-1 基本信息topic管理Consumer Group监控事件中心HTTP接入ACL策略管理智能运维专业版集群备份专业版 新建(24/450) 请输入TopicId或名称 <table><tr><th>ID/名称</th><th>监控</th><th>分区数(个)</th><th>副本数(个)</th><th>标签</th><th>备注</th><th>创建时间</th><th>消息保留时间</th><th>状态</th><th>操作</th></tr><tr><td>topic-j cocs </td><td></td><td>3</td><td>2</td><td></td><td></td><td>2023-03-25 23:03:00</td><td>2 天</td><td>正常</td><td>编辑 删除 更多</td></tr></table>	ID/名称	监控	分区数(个)	副本数(个)	标签	备注	创建时间	消息保留时间	状态	操作	topic-j cocs 		3	2			2023-03-25 23:03:00	2 天	正常	编辑 删除 更多
ID/名称	监控	分区数(个)	副本数(个)	标签	备注	创建时间	消息保留时间	状态	操作												
topic-j cocs 		3	2			2023-03-25 23:03:00	2 天	正常	编辑 删除 更多												

运行结果如下：

```
[root@VM_1_28_centos ~]# ./consumer consumer_gr
% Subscribed to 1 topic(s), waiting for rebalance and messages...
% Consumer error: Broker: No more messages
Message on test [1] at offset 0:
Value: test
Message on test [1] at offset 1:
Value: test123
% Consumer error: Broker: No more messages
```

4. 在 CKafka 控制台 **Consumer Group** 页面，选择对应的消费者组，在主题名称输入 Topic 名称，单击**查询详情**即可查看消费详情。



公网 SASL_PLAINTEXT 方式接入

最近更新时间：2024-05-31 14:40:49

操作场景

该任务以 C++ 客户端为例，指导您使用公网 SASL_PLAINTEXT 方式接入消息队列 CKafka 版并收发消息。

前提条件

[安装 GCC](#)

[配置 ACL 策略](#)

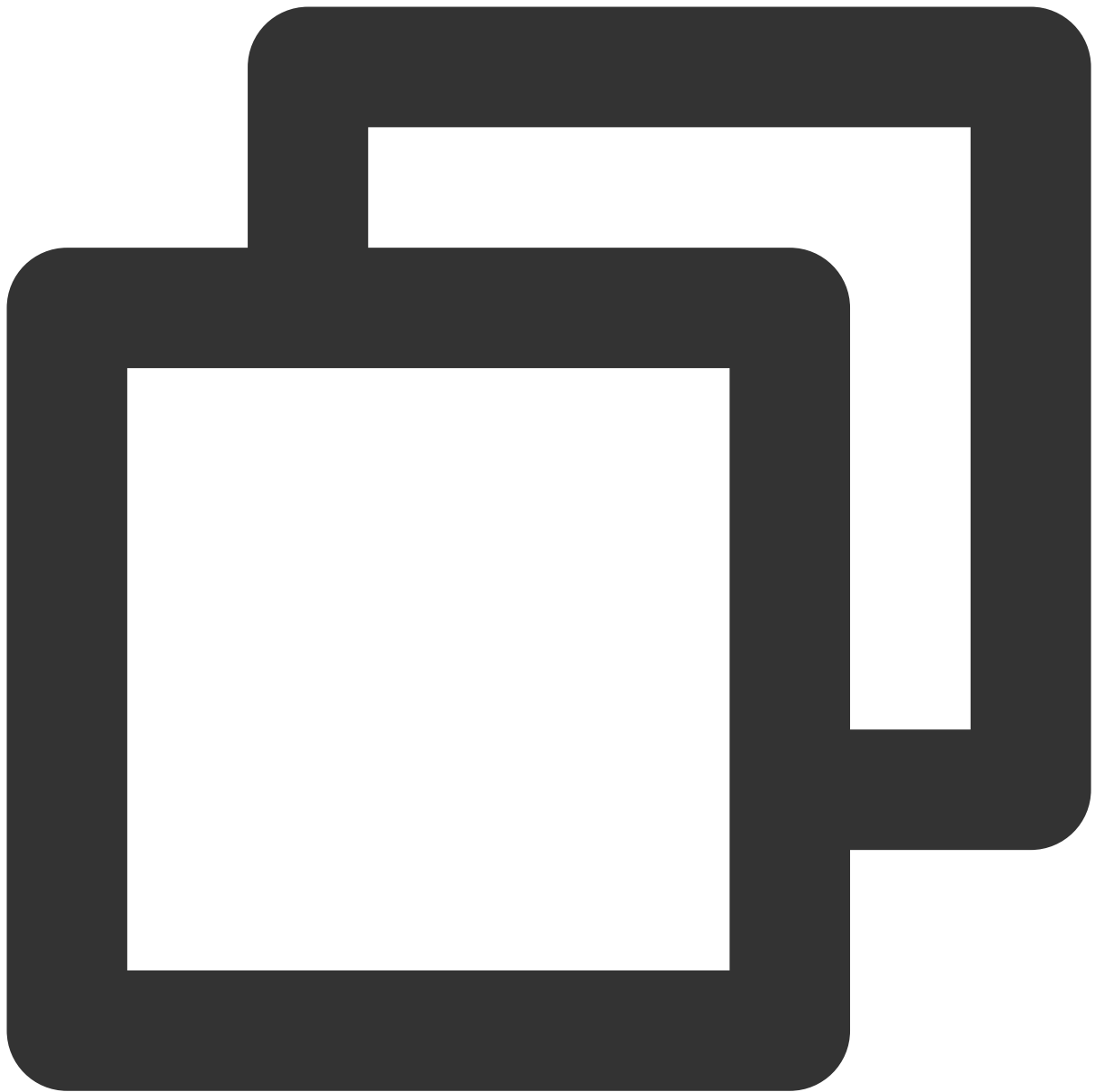
[下载 Demo](#)

操作步骤

步骤一：安装 C/C++ 依赖库

[安装 librdkafka](#)

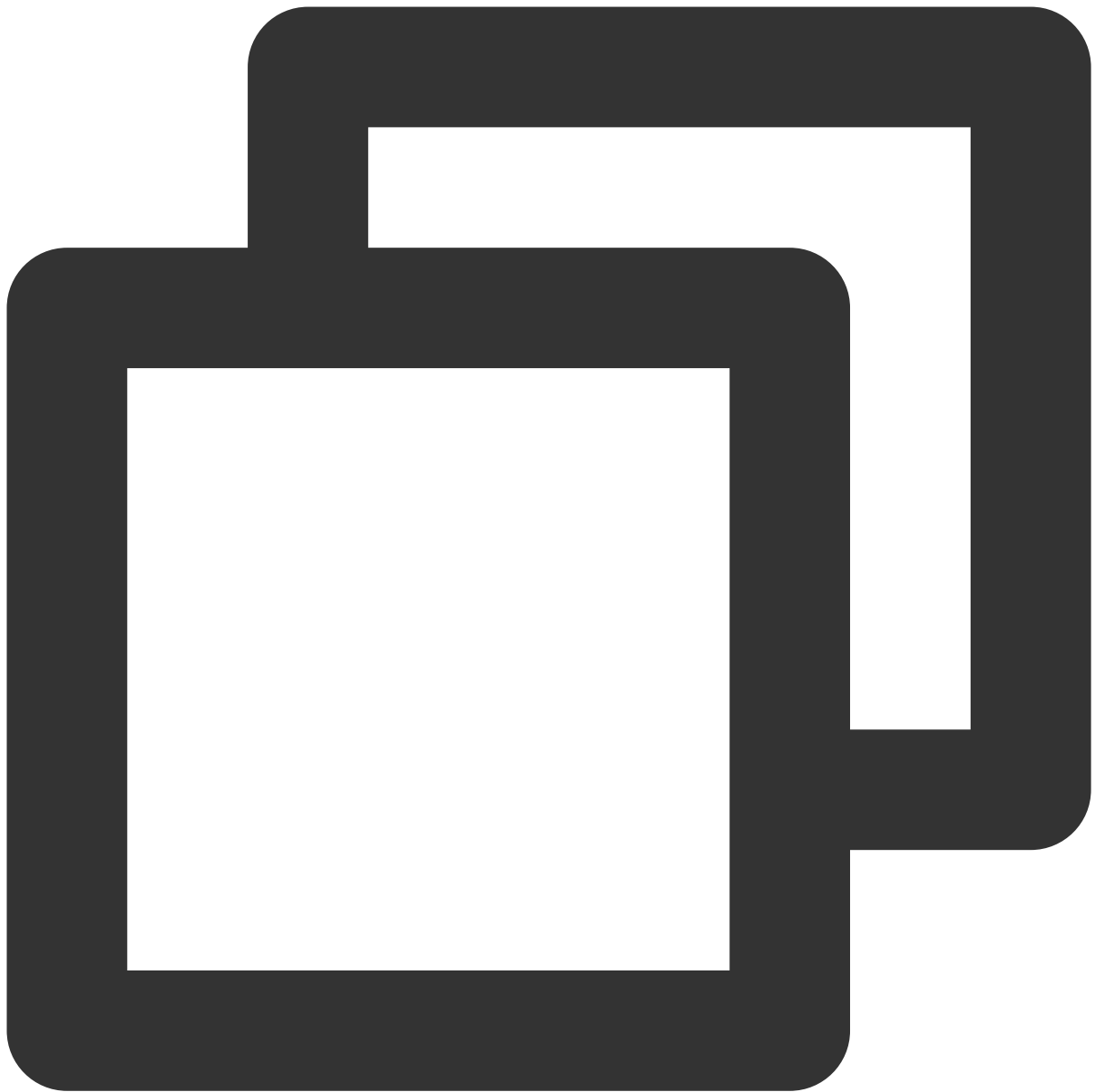
步骤二：安装 SSL/SASL 依赖



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

步骤三：发送消息

1. 创建 producer.c 文件。



```
/*
 * 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, "%s Message delivery failed: %s\\n",
                rd_kafka_err2str(rkmessage->err));
    else
        fprintf(stderr,
                "%s Message delivered (%zd bytes, "
                "partition %"PRIu32")\\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, "%s 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, sizeof(errstr)) != RD_KAFKA_CONF_OK) {
        fprintf(stderr, "%s\\n", errstr);
        return 1;
    }
    if(rd_kafka_conf_set(conf, "sasl.mechanisms", "PLAIN", errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK) {
        fprintf(stderr, "%s\\n", errstr);
        return 1;
    }
    if(rd_kafka_conf_set(conf, "sasl.username", user, errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK) {
        fprintf(stderr, "%s\\n", errstr);
        return 1;
    }
    if(rd_kafka_conf_set(conf, "sasl.password", passwd, errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK) {
        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_CONF_OK) {
//    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)) != RD_KAFKA_CONF_OK) {
        fprintf(stderr, "%s\\n", errstr);
        return 1;
    }
    if(rd_kafka_conf_set(conf, "retries", "3", errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK) {
        fprintf(stderr, "%s\\n", errstr);
        return 1;
    }

    if(rd_kafka_conf_set(conf, "retry.backoff.ms", "1000", errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK) {
        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,
                "%s: Failed to create new producer: %s\\n", errstr);
        return 1;
    }

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

    fprintf(stderr,
            "%s Type some text and hit enter to produce message\\n"
            "%s Or just hit enter to only serve delivery reports\\n"
            "%s 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,
            "%s 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, "%s 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, "%s Flushing final messages..\\n");
rd_kafka_flush(rk, 10*1000 /* wait for max 10 seconds */);

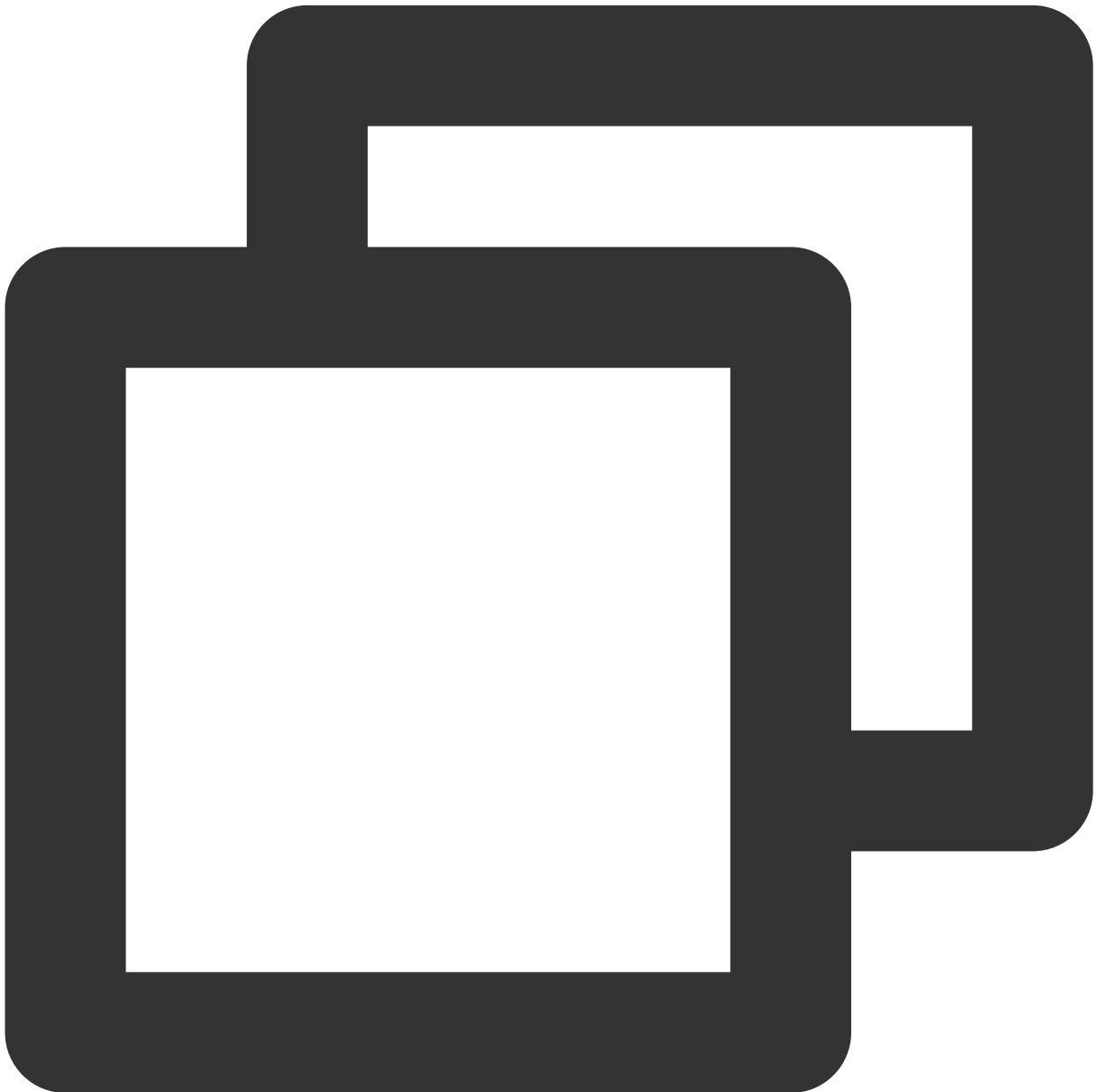
/* If the output queue is still not empty there is an issue
 * with producing messages to the clusters. */
if (rd_kafka_outq_len(rk) > 0)
    fprintf(stderr, "%s %d message(s) were not delivered\\n",
```

```
rd_kafka_outq_len(rk));

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

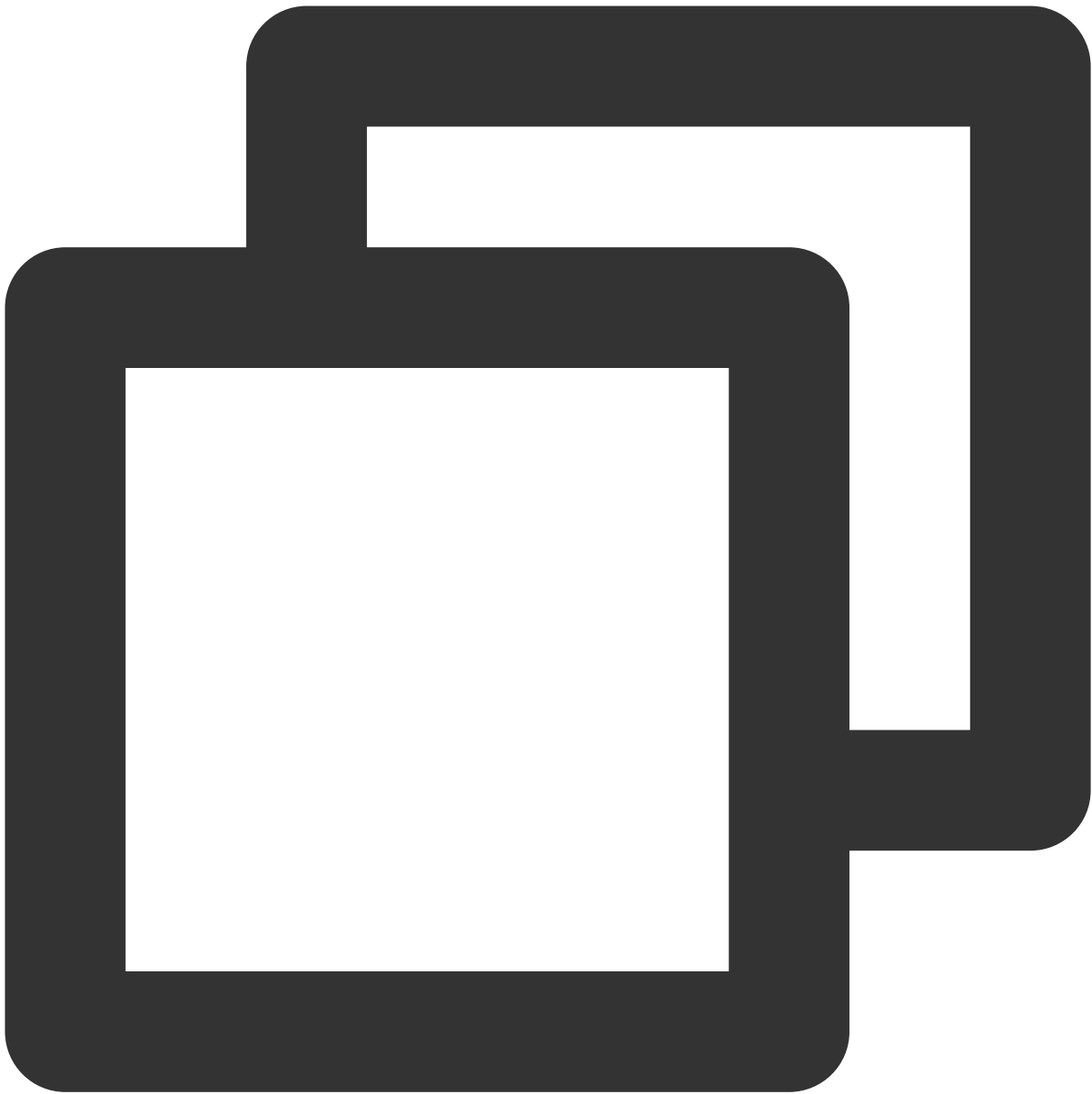
return 0;
}
```

2. 执行以下命令编译 `producer.c`。



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

3. 执行以下命令发送消息。



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

参数	描述
broker	接入网络，在控制台的实例详情页面接入方式模块的网络列复制。

	接入方式? 添加路由策略 <table><tr><td>接入类型</td><td>接入方式</td><td>网络</td><td>操作</td></tr><tr><td>公网域名接入</td><td>SASL_PLAINTEXT</td><td>ckafka-l4xppqrz...</td><td>删除</td></tr></table>	接入类型	接入方式	网络	操作	公网域名接入	SASL_PLAINTEXT	ckafka-l4xppqrz...	删除													
接入类型	接入方式	网络	操作																			
公网域名接入	SASL_PLAINTEXT	ckafka-l4xppqrz...	删除																			
topic	Topic 名称，您可以在控制台上 topic 管理 页面复制。 基本信息topic管理Consumer Group监控事件中心HTTP接入ACL策略管理智能运维专业版 新建(12/450) <table><tr><th>ID名称</th><th>监控</th><th>分区数(个)</th><th>副本数(个)</th><th>标签</th><th>备注</th><th>创建时间</th></tr><tr><td>topic- </td><td></td><td>100</td><td>3</td><td></td><td></td><td>2022-11-0...</td></tr><tr><td>topic- </td><td></td><td>1</td><td>1</td><td></td><td></td><td>2022-11-0...</td></tr></table>	ID名称	监控	分区数(个)	副本数(个)	标签	备注	创建时间	topic-		100	3			2022-11-0...	topic-		1	1			2022-11-0...
ID名称	监控	分区数(个)	副本数(个)	标签	备注	创建时间																
topic-		100	3			2022-11-0...																
topic-		1	1			2022-11-0...																
username	username 是 实例 ID + # + 配置的用户名，实例 ID 在 CKafka 控制台的实例详情页面的基本信 取，用户名在控制台 用户管理 页面创建用户时设置																					
password	password 是配置的用户密码，在控制台 用户管理 页面创建用户时设置。																					

运行结果如下：

```
% 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
argc: 5 ***** test message
% Enqueued message (13 bytes) for topic test

% Message delivered (13 bytes, partition 0)
```

4. 在 [CKafka 控制台](#) **topic 管理** 页面，选择对应的 Topic，单击**更多** > **消息查询**，查看刚刚发送的消息。

实例  kafka_instance_id

Topic topic_name

按位点查询

按时间查询

分区ID	0
------	---

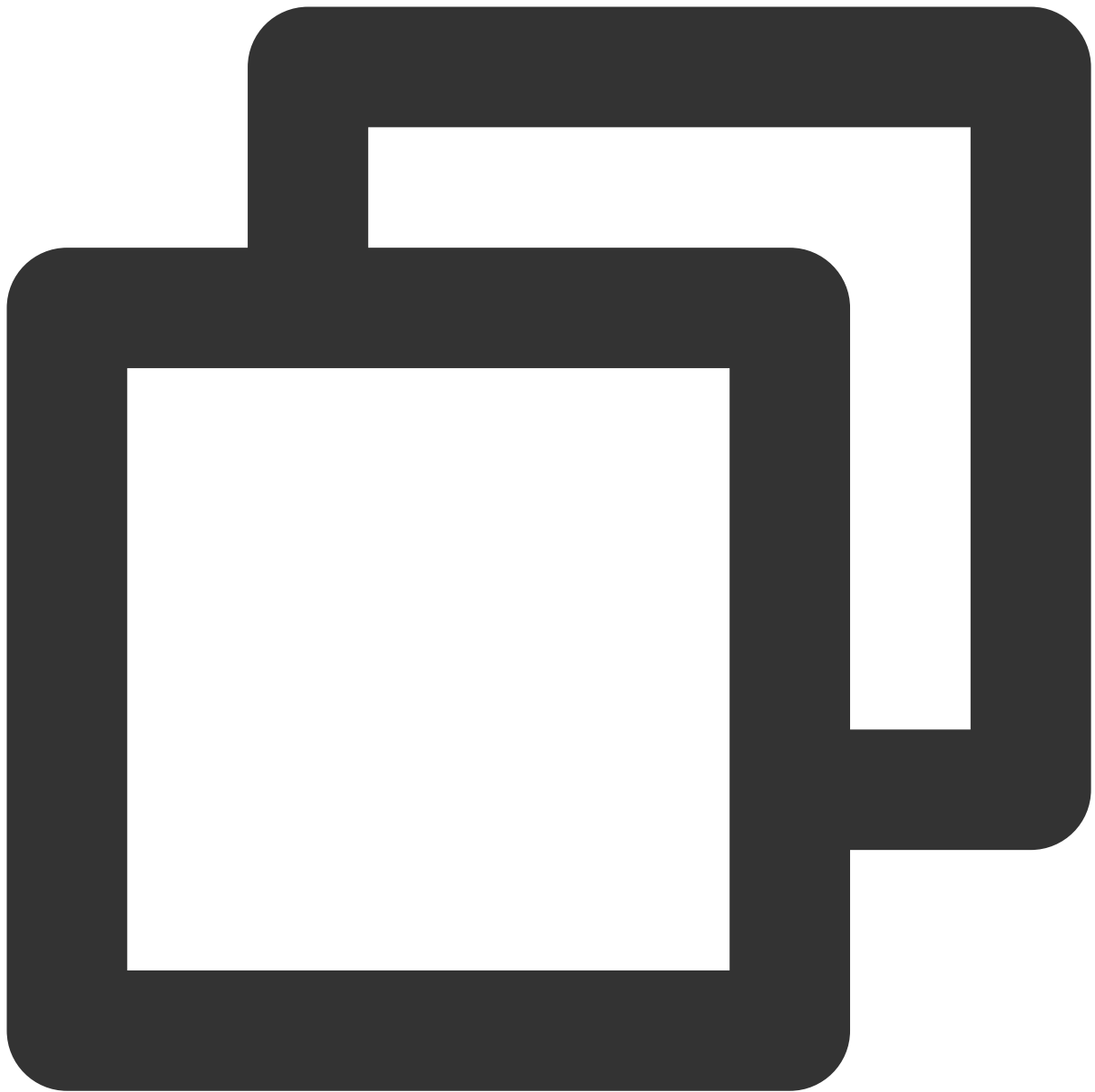
起始位点	0
------	---

查询

分区ID	位点	时间戳
0	634	2024-01-01 12:00:00

步骤四：消费消息

- ### 1. 创建 consumer.c 文件。



```
/*
 * 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>...\n",
                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,
        "%s 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,
            "%s 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,
        "%s 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);

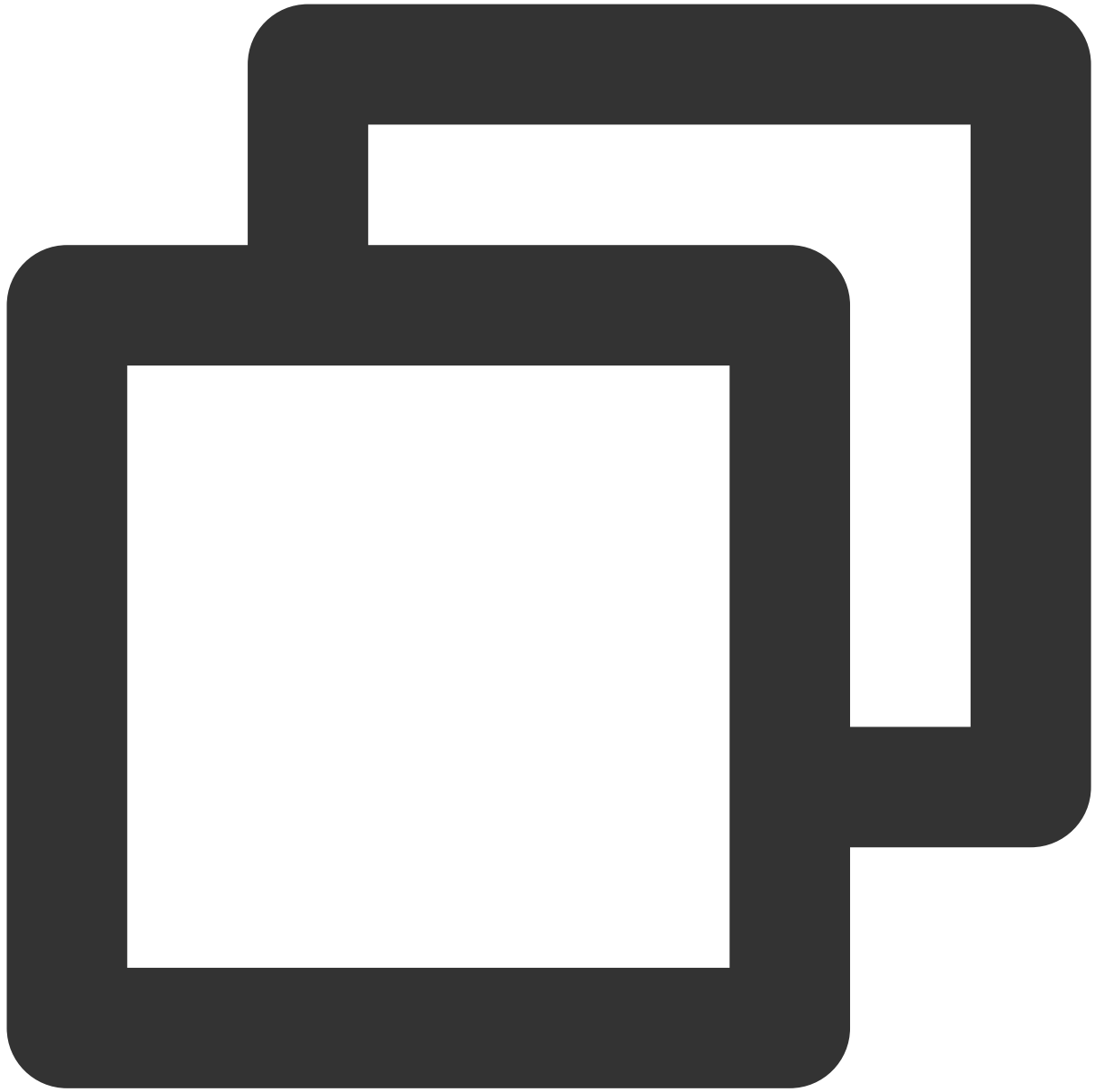
rd_kafka_message_destroy(rkm);
}

/* 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);

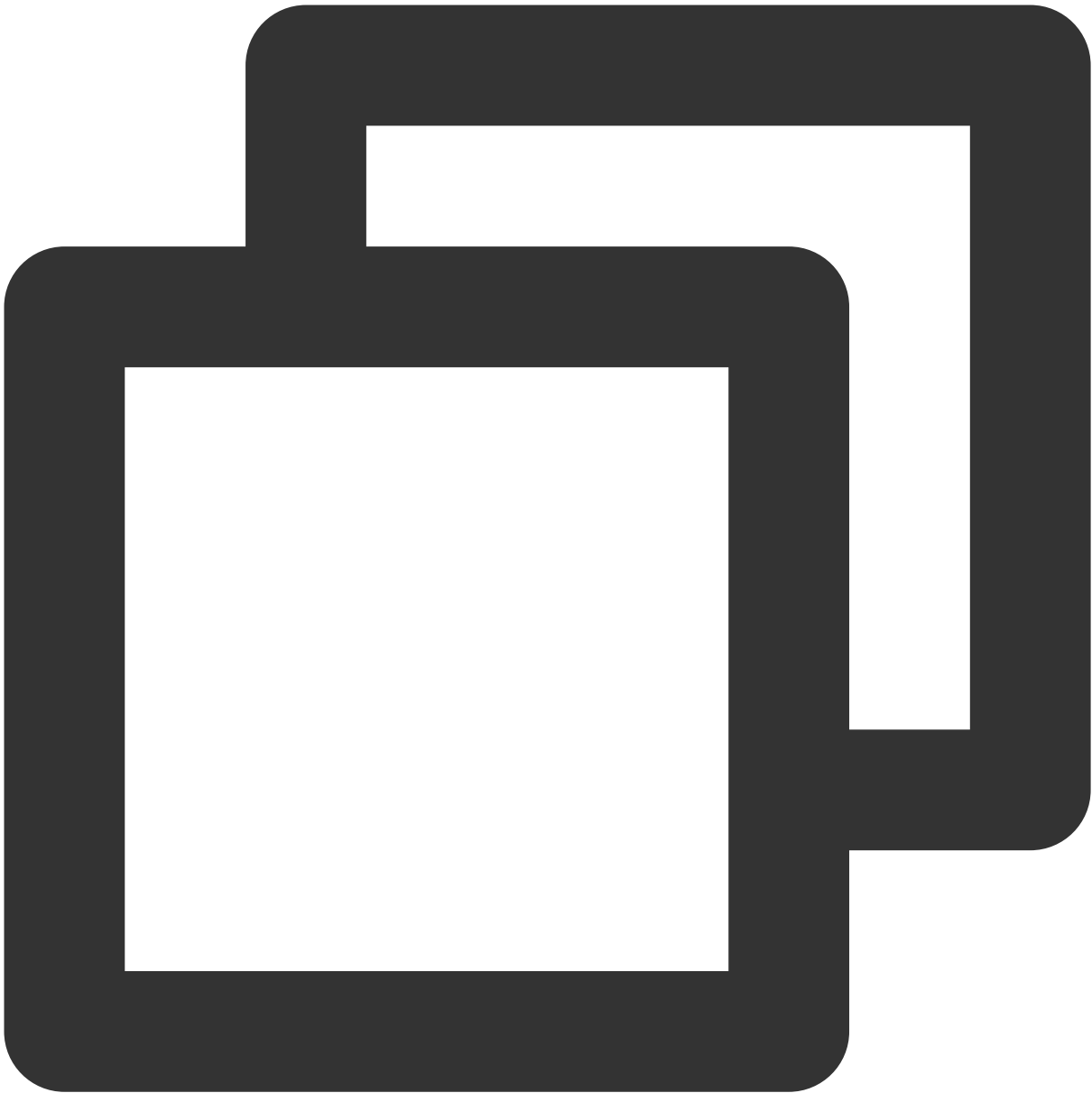
return 0;
}
```

2. 执行以下命令编译 consumer.c。



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

3. 执行以下命令发送消息。



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

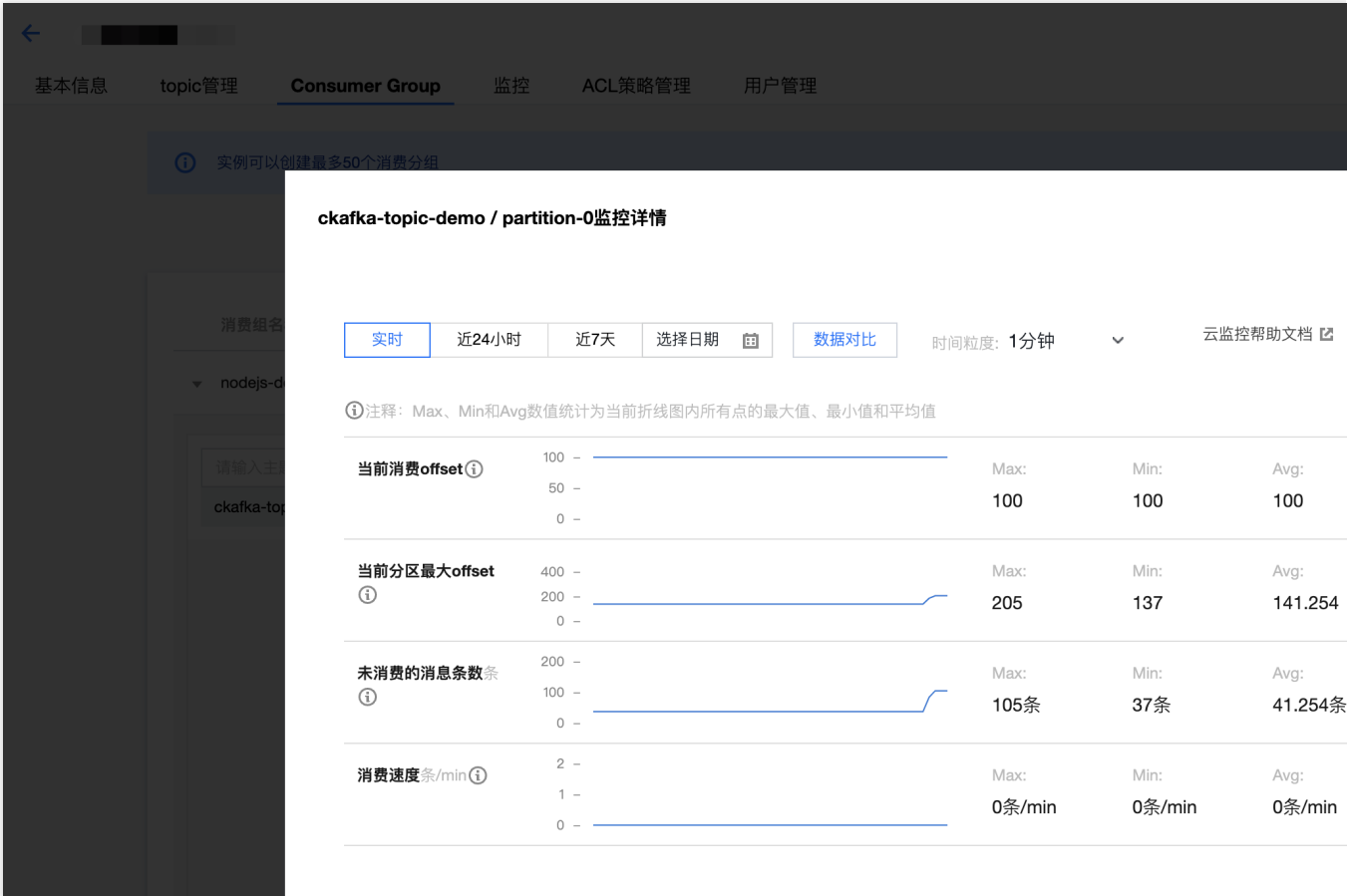
参数	描述
broker	接入网络，在控制台的实例详情页面接入方式模块的网络列复制。

	接入方式? 添加路由策略 <table><tr><td>接入类型</td><td>接入方式</td><td>网络</td><td>操作</td></tr><tr><td>公网域名接入</td><td>SASL_PLAINTEXT</td><td>ckafka-l4xppqrz...</td><td>删除</td></tr></table>	接入类型	接入方式	网络	操作	公网域名接入	SASL_PLAINTEXT	ckafka-l4xppqrz...	删除													
接入类型	接入方式	网络	操作																			
公网域名接入	SASL_PLAINTEXT	ckafka-l4xppqrz...	删除																			
group.id	消费分组名称，您可以自定义设置，Demo 运行成功后可以在 Consumer Group 页面看到该消费者																					
username	username 是 实例 ID + # + 配置的用户名，实例 ID 在 CKafka 控制台的实例详情页面的基本																					
password	password 是配置的用户密码，在控制台用户管理页面创建用户时设置。																					
topic1 topic2..	Topic 名称，您可以在控制台上 topic 管理页面复制。 基本信息topic管理Consumer Group监控事件中心HTTP接入ACL策略管理智能运维专业版 新建(12/450) <table><tr><th>ID/名称</th><th>监控</th><th>分区数(个)</th><th>副本数(个)</th><th>标签</th><th>备注</th><th>创建时间</th></tr><tr><td>topic- ID</td><td> </td><td>100</td><td>3</td><td></td><td></td><td>2022-11-0</td></tr><tr><td>topic- ID</td><td> </td><td>1</td><td>1</td><td></td><td></td><td>2022-11-0</td></tr></table>	ID/名称	监控	分区数(个)	副本数(个)	标签	备注	创建时间	topic- ID		100	3			2022-11-0	topic- ID		1	1			2022-11-0
ID/名称	监控	分区数(个)	副本数(个)	标签	备注	创建时间																
topic- ID		100	3			2022-11-0																
topic- ID		1	1			2022-11-0																

运行结果如下：

```
[root@VM-8-16-centos kafka]# ./consumer ckafka-2.ap-shanghai.ckafka.tencentcloud.cn ckafka-consumer-group-demo test
% Subscribed to 1 topic(s), waiting for rebalance and messages...
Message on test [0] at offset 2007:
Value: test message
% Consumer error: Broker: No more messages
```

4. 在 [CKafka 控制台](#)的 **Consumer Group** 页面，选择对应的消费者组名称，在主题名称输入 Topic，单击**查询详情**即可查看消费详情。



Node.js SDK

VPC 网络接入

最近更新时间：2024-05-31 14:40:49

操作场景

该任务以 Node.js 客户端为例指导您使用 VPC 网络接入消息队列 CKafka 版并收发消息。

前提条件

[安装 GCC](#)

[安装 Node.js](#)

[下载 Demo](#)

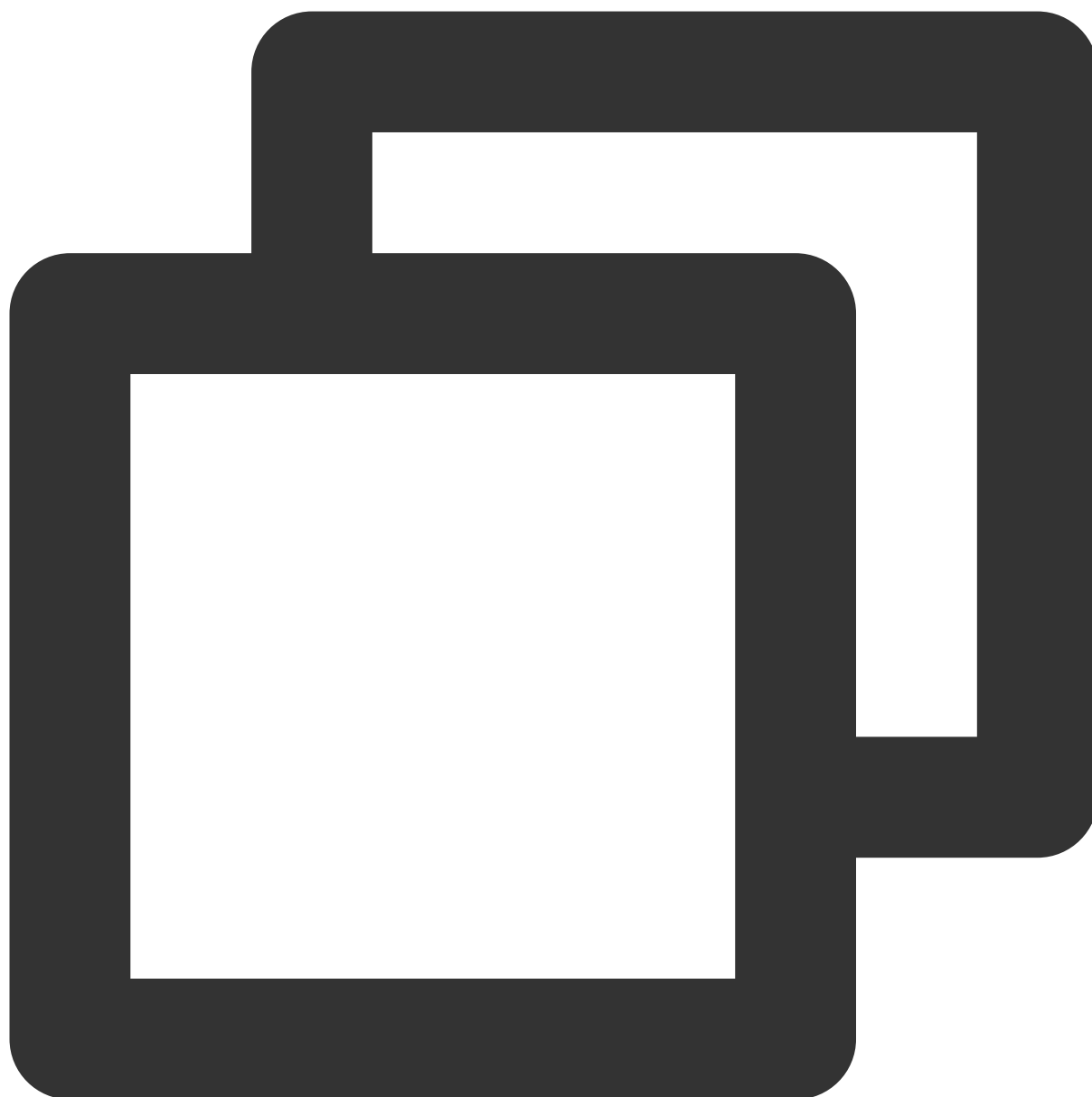
操作步骤

准备工作

1. 将下载的 Demo 中的 nodejskafkademo 上传至 Linux 服务器。
2. 登录 Linux 服务器，进入 nodejskafkademo 目录。

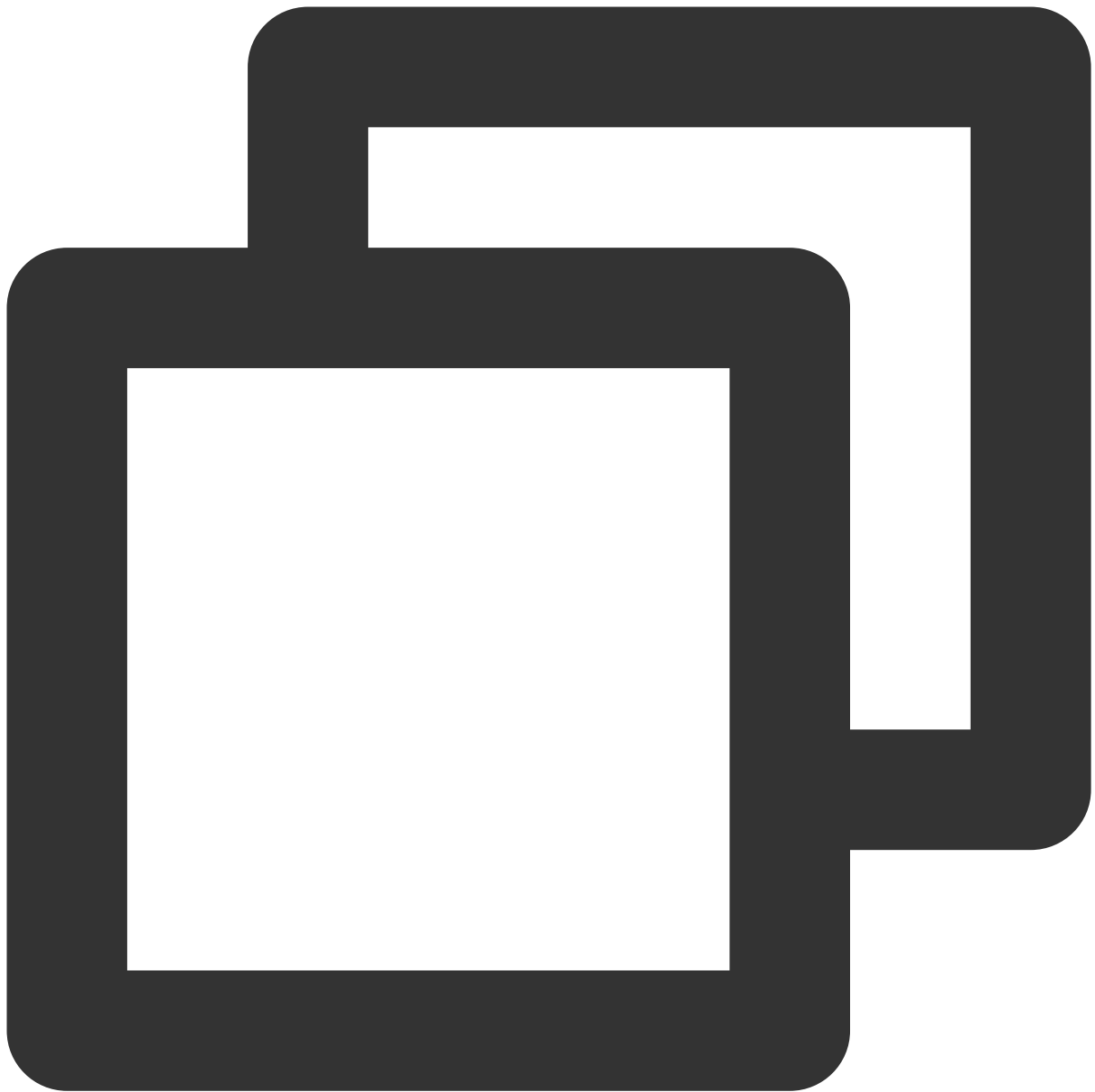
步骤1：安装 C++ 依赖库

1. 执行以下命令切换到 yum 源配置目录 `/etc/yum.repos.d/`。



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

2. 创建 yum 源配置文件 `confluent.repo`。

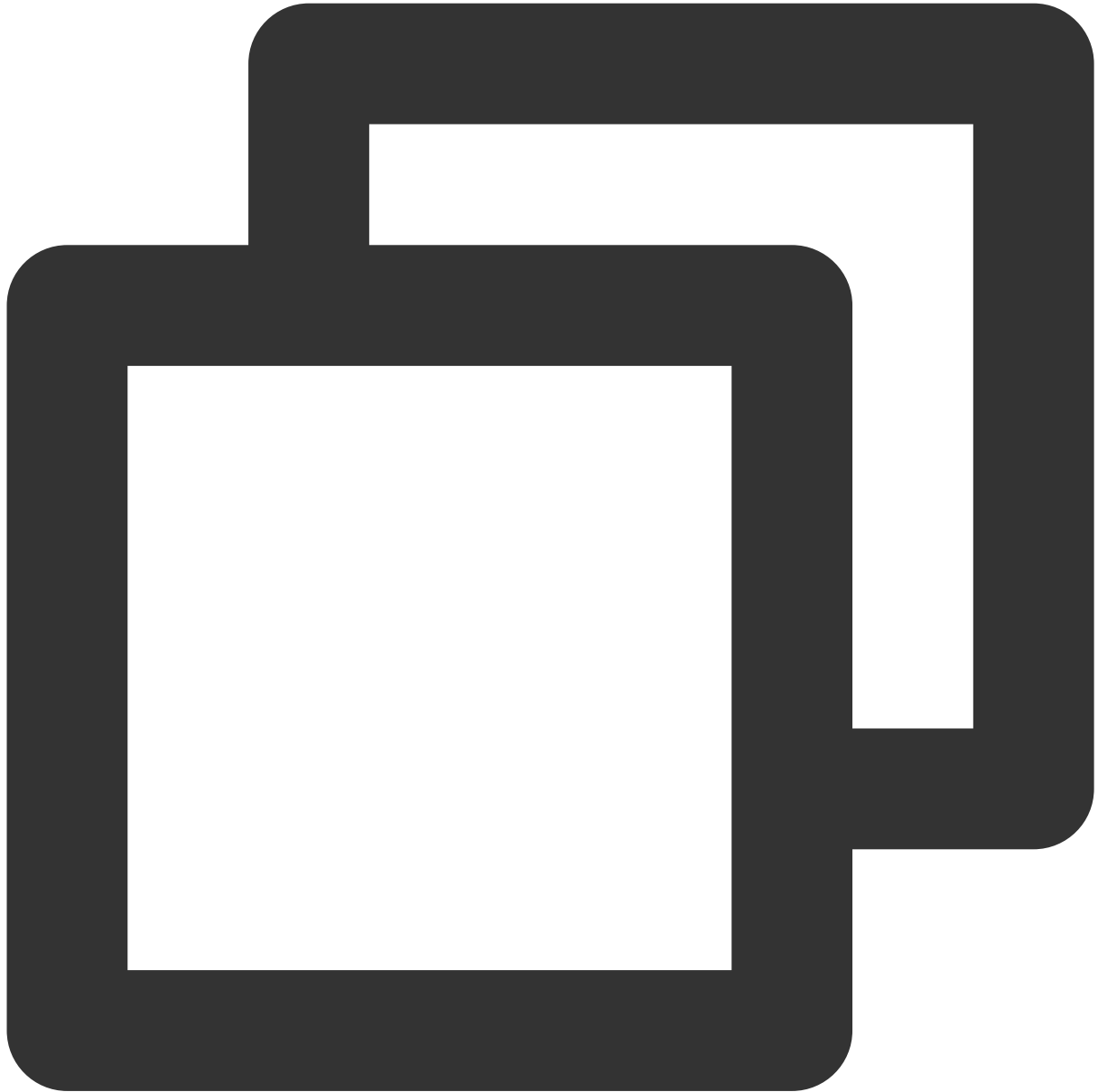


```
[Confluent.dist]
name=Confluent repository (dist)
baseurl=https://packages.confluent.io/rpm/5.1/7
gpgcheck=1
gpgkey=https://packages.confluent.io/rpm/5.1/archive.key
enabled=1
[Confluent]
name=Confluent repository
baseurl=https://packages.confluent.io/rpm/5.1
gpgcheck=1
gpgkey=https://packages.confluent.io/rpm/5.1/archive.key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



```
enabled=1
```

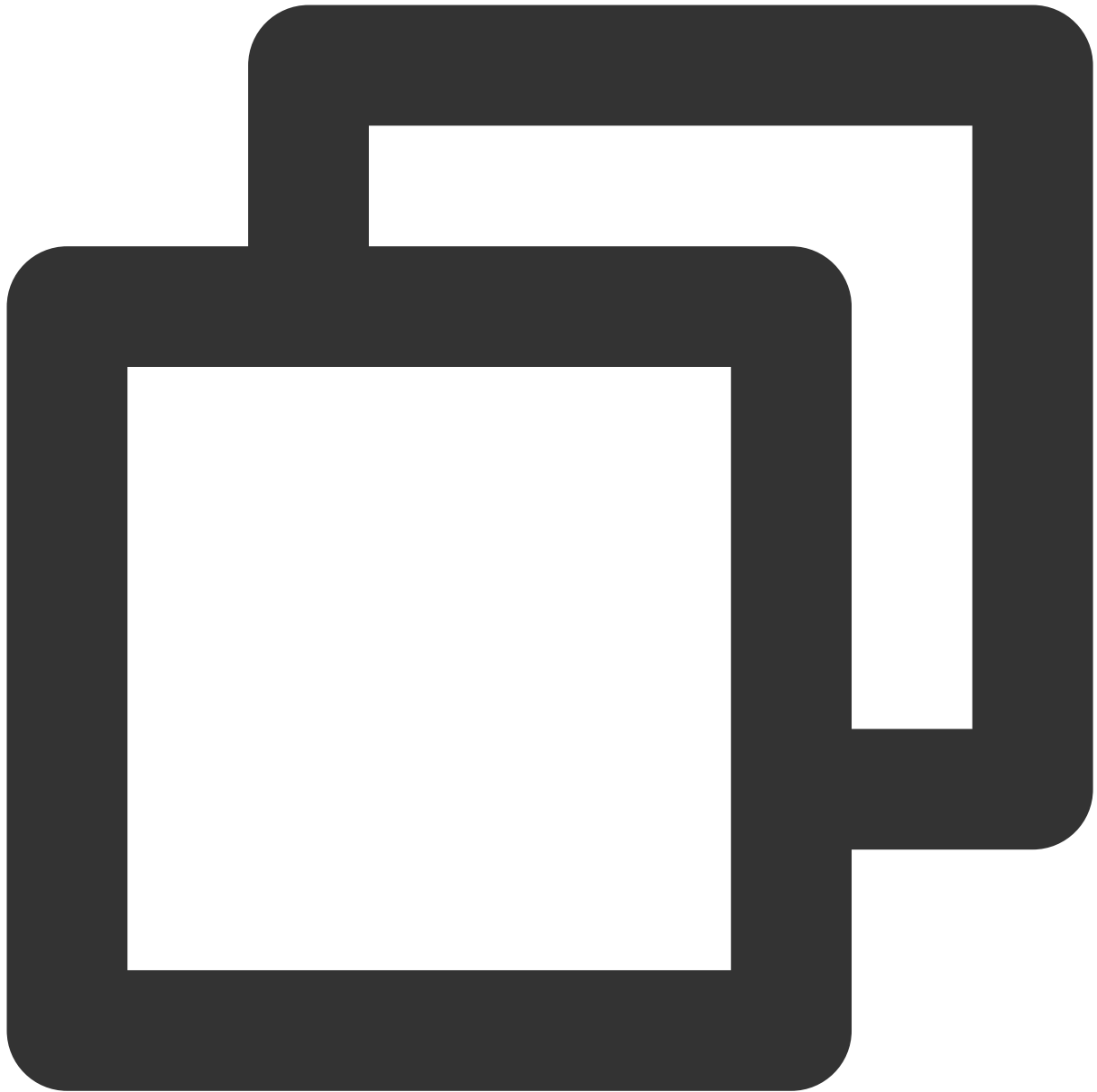
3. 执行以下命令安装 C++ 依赖库。



```
yum install librdkafka-devel
```

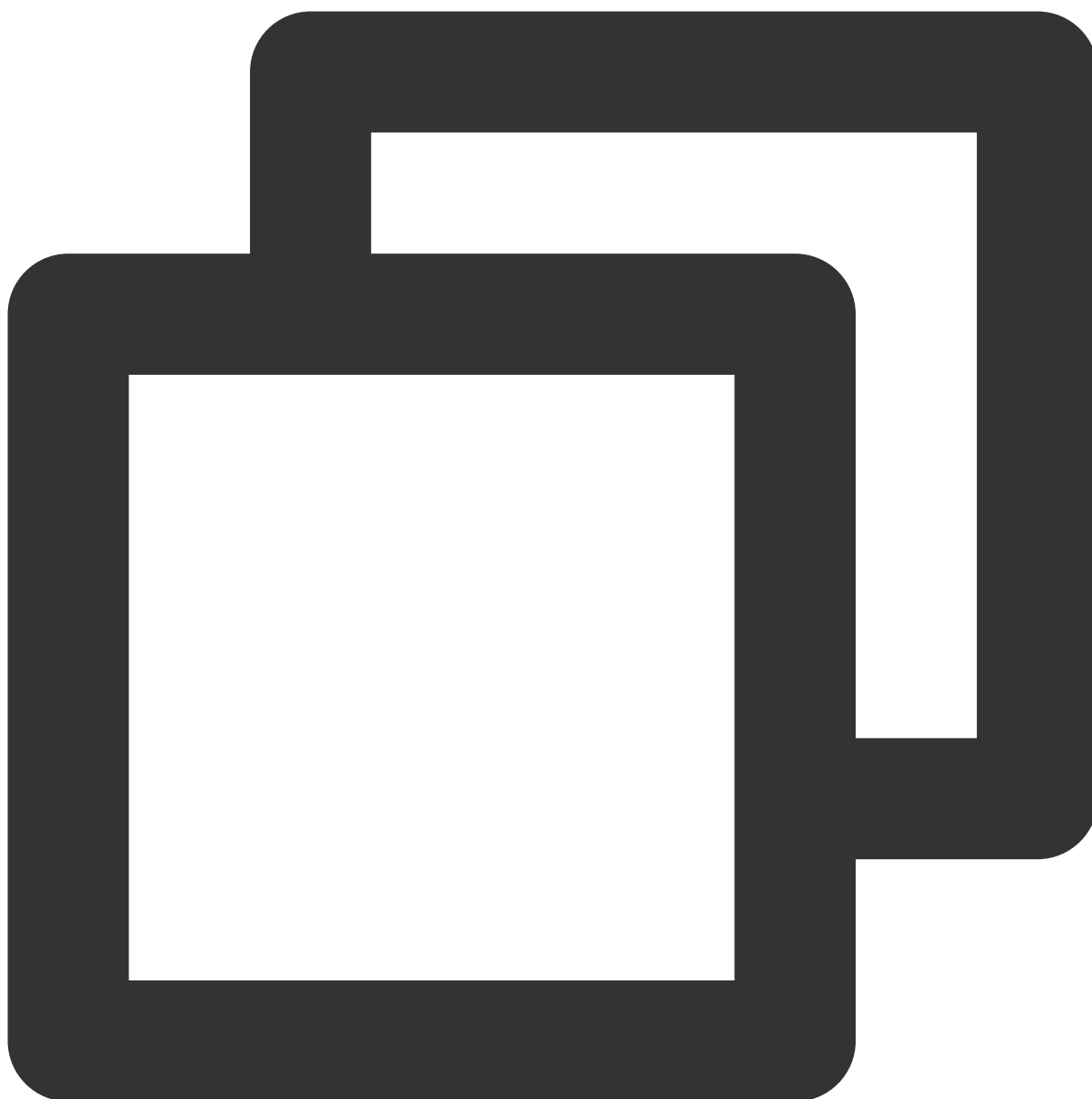
步骤2：安装 Node.js 依赖库

1. 执行以下命令为预处理器指定 OpenSSL 头文件路径。



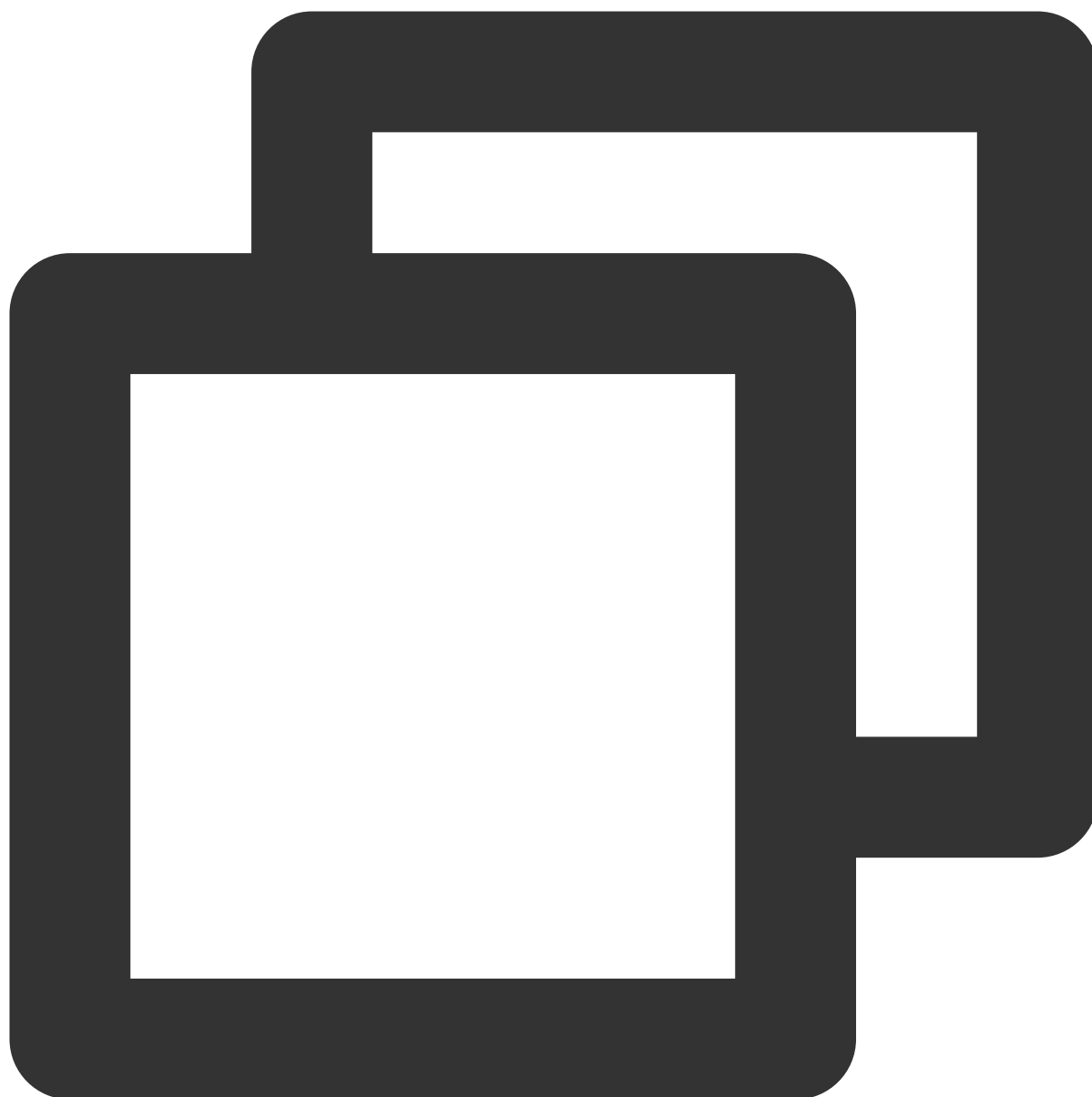
```
export CPPFLAGS=-I/usr/local/opt/openssl/include
```

2. 执行以下命令为连接器指定 OpenSSL 库路径。



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

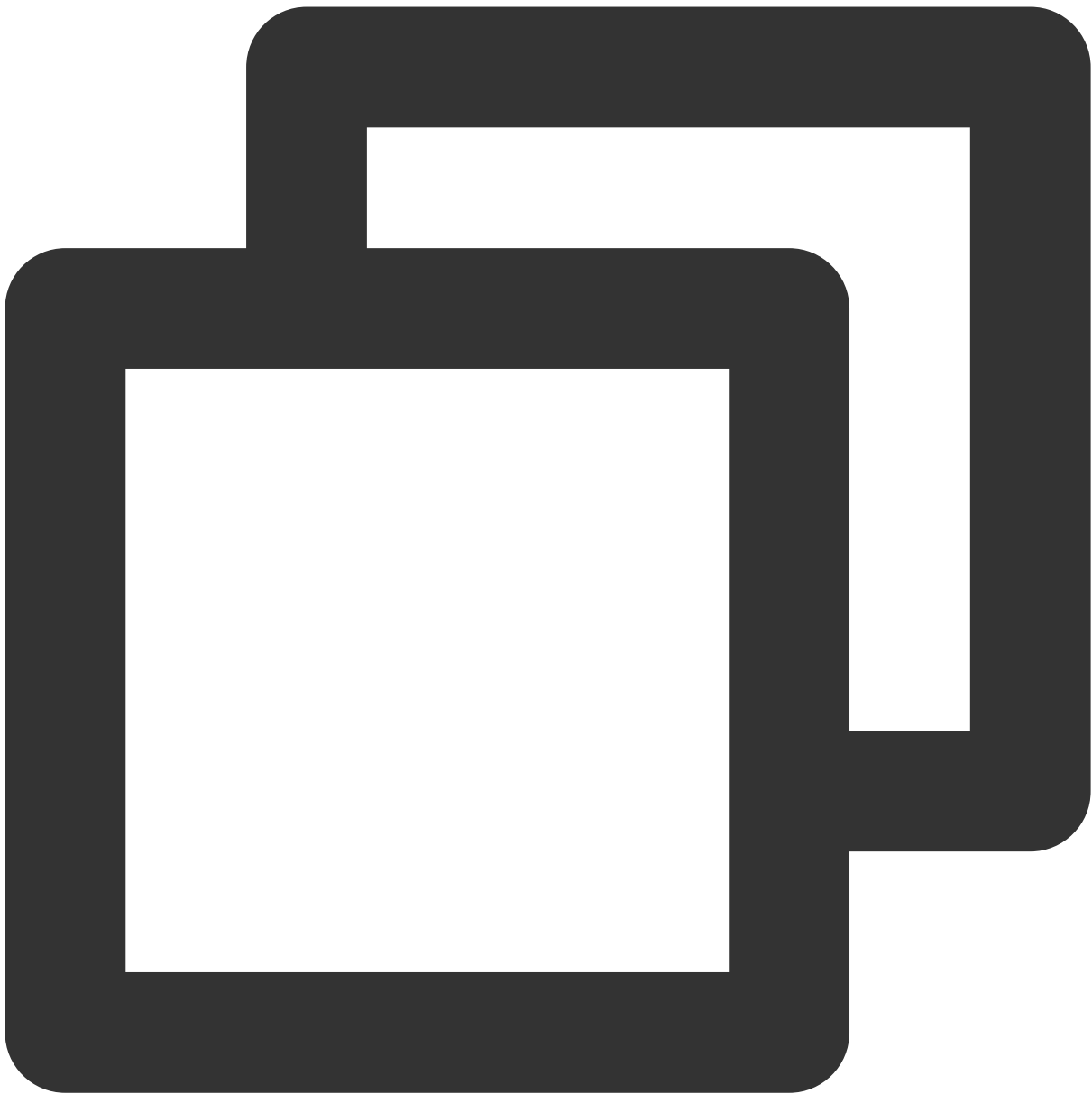
3. 执行以下命令安装 Node.js 依赖库。



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

步骤3：准备配置

创建消息队列 CKafka 版配置文件 `setting.js`。



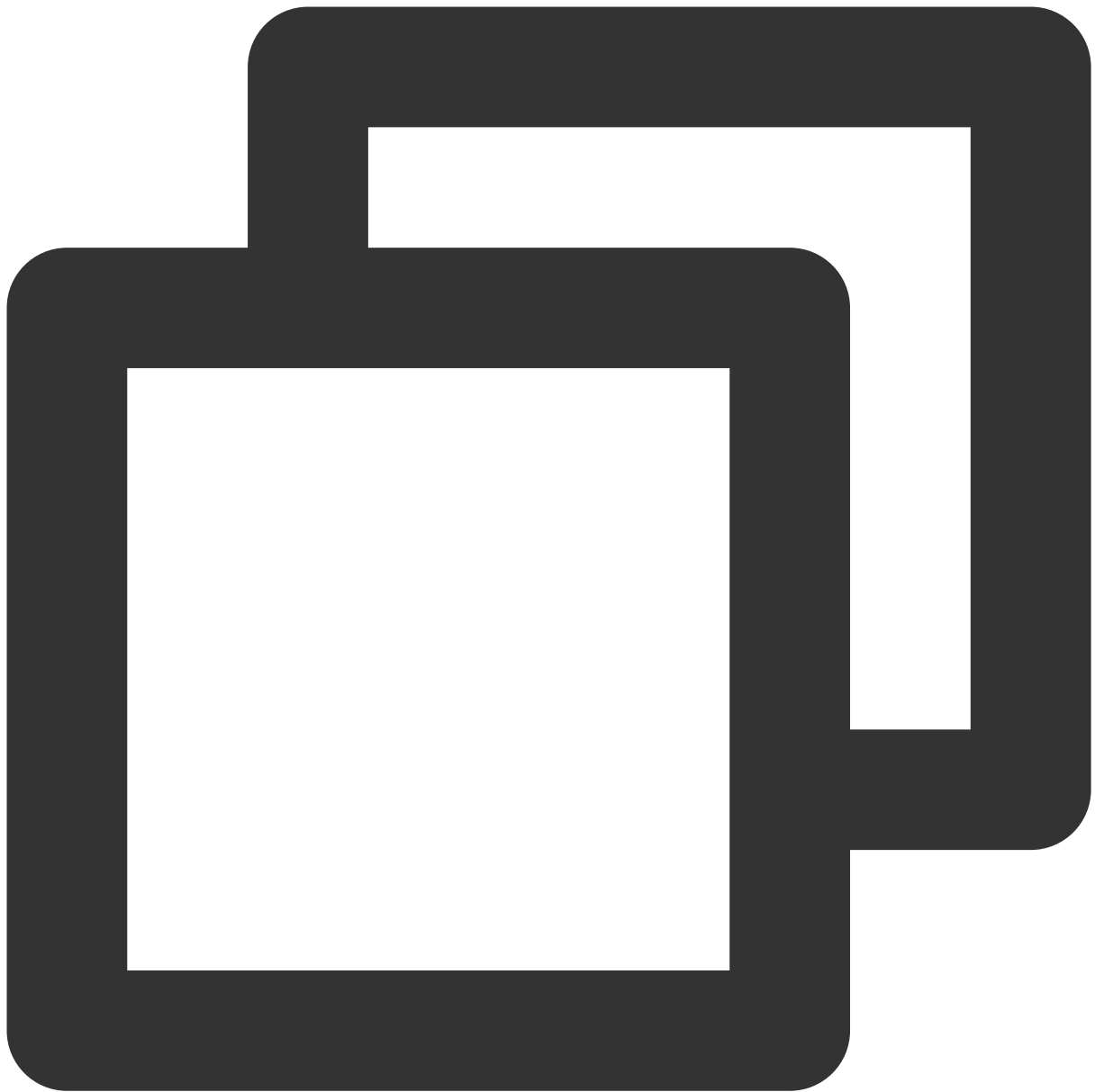
```
module.exports = {
  'bootstrap_servers': ["xxx.xx.xxx:xxxx"],
  'topic_name': 'xxx',
  'group_id': 'xxx'
}
```

参数	描述
bootstrap_servers	接入网络，在控制台的实例详情页面接入方式模块的网络列复制。

	接入方式? <table><tr><th>接入类型</th><th>接入方式</th><th>网络</th><th>操作</th></tr><tr><td>VPC网络</td><td>PLAINTEXT</td><td> vpc sub 10:7:9092 </td><td>删除 查看所</td></tr></table>	接入类型	接入方式	网络	操作	VPC网络	PLAINTEXT	vpc sub 10:7:9092	删除 查看所										
接入类型	接入方式	网络	操作																
VPC网络	PLAINTEXT	vpc sub 10:7:9092	删除 查看所																
topic_name	Topic 名称，您可以在控制台上 topic管理 页面复制。 基本信息topic管理Consumer Group监控事件中心HTTP接入ACL策略管理智能运维专业版 新建(12/450) <table><tr><th>ID/名称</th><th>监控</th><th>分区数(个)</th><th>副本数(个)</th><th>标签</th><th>备注</th></tr><tr><td> topic- 后 </td><td>山</td><td>100</td><td>3</td><td></td><td></td></tr><tr><td> topic- 后 </td><td>山</td><td>1</td><td>1</td><td></td><td></td></tr></table>	ID/名称	监控	分区数(个)	副本数(个)	标签	备注	topic- 后	山	100	3			topic- 后	山	1	1		
ID/名称	监控	分区数(个)	副本数(个)	标签	备注														
topic- 后	山	100	3																
topic- 后	山	1	1																
group_id	您可以自定义设置， Demo 运行成功后可以在 Consumer Group 页面看到该消费者。																		

步骤4：发送消息

1. 编写生产消息程序 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',
  // 设置入口服务, 请通过控制台获取对应的服务地址。
  'bootstrap.servers': config['bootstrap_servers'],
  'dr_cb': true,
  'dr_msg_cb': true,
```

```
// 请求发生错误时重试次数，建议将该值设置为大于0，失败重试最大程度保证消息不丢失
'retries': '0',
// 发送请求失败时到下一次重试请求之间的时间
'retry.backoff.ms': 100,
// producer 网络请求的超时时间。
'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);
  }
}
```



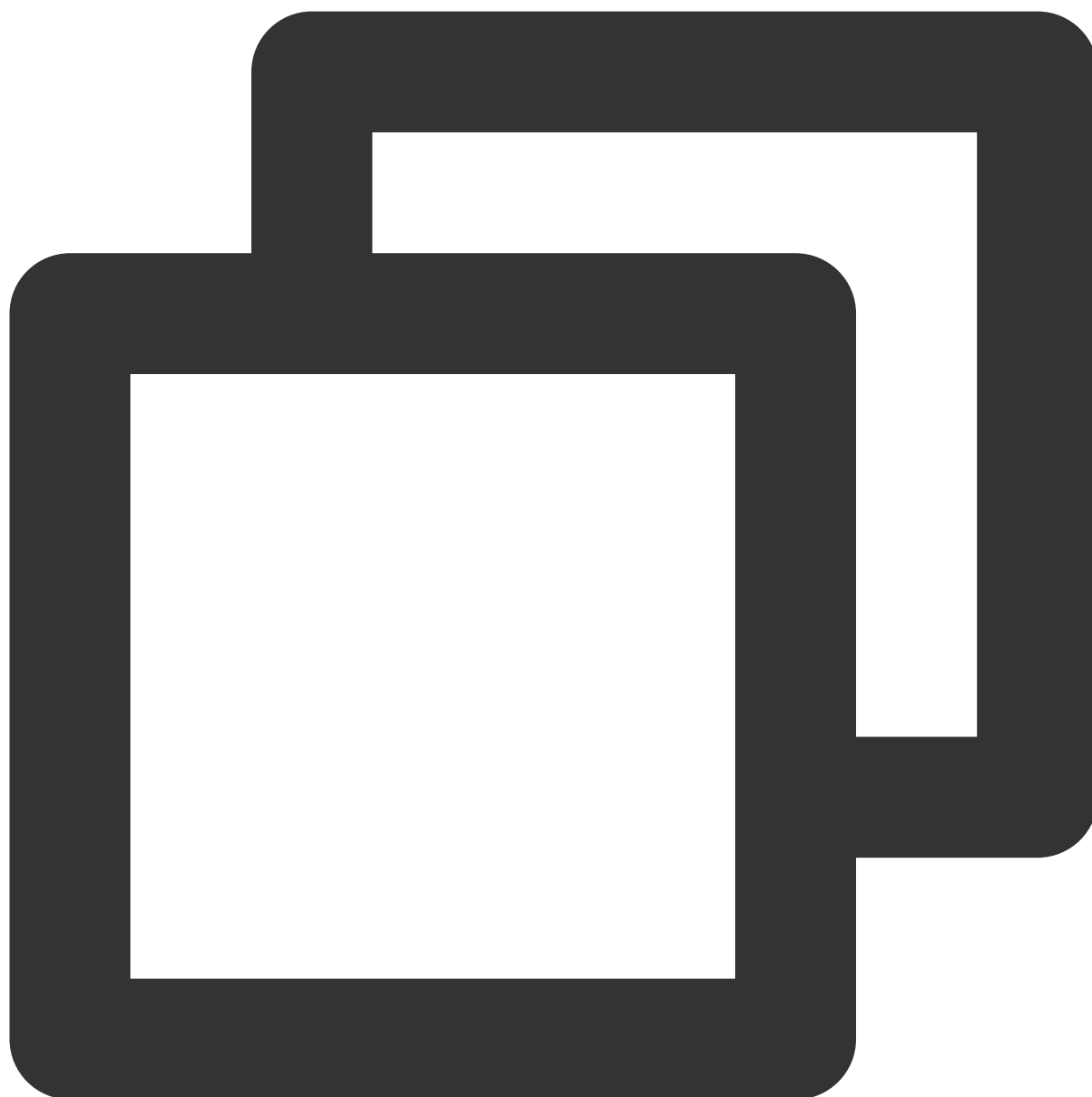
```
}

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. 执行以下命令发送消息。



```
node producer.js
```

3. 查看运行结果。

```
~/Demos/ckafka-demo/nodejskafkadem/sasl  kafka_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.alloc(), 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
delivery-report: producer ok
delivery-report: producer ok
delivery-report: producer ok
delivery-report: producer ok
```

4. 在 [CKafka 控制台](#) **topic** 管理页面，选择对应的 Topic，单击 **更多 > 消息查询**，查看刚刚发送的消息。

消息查询

消息查询会占用CKafka实例的带宽资源，建议您尽量缩小查询范围查询，不要频繁操作。

消息查询最多展示最近的20条消息。

实例

ckafka

Topic

ckafka-topic-demo

查询类型

按位点查询按起始时间查询

分区ID

0

时间

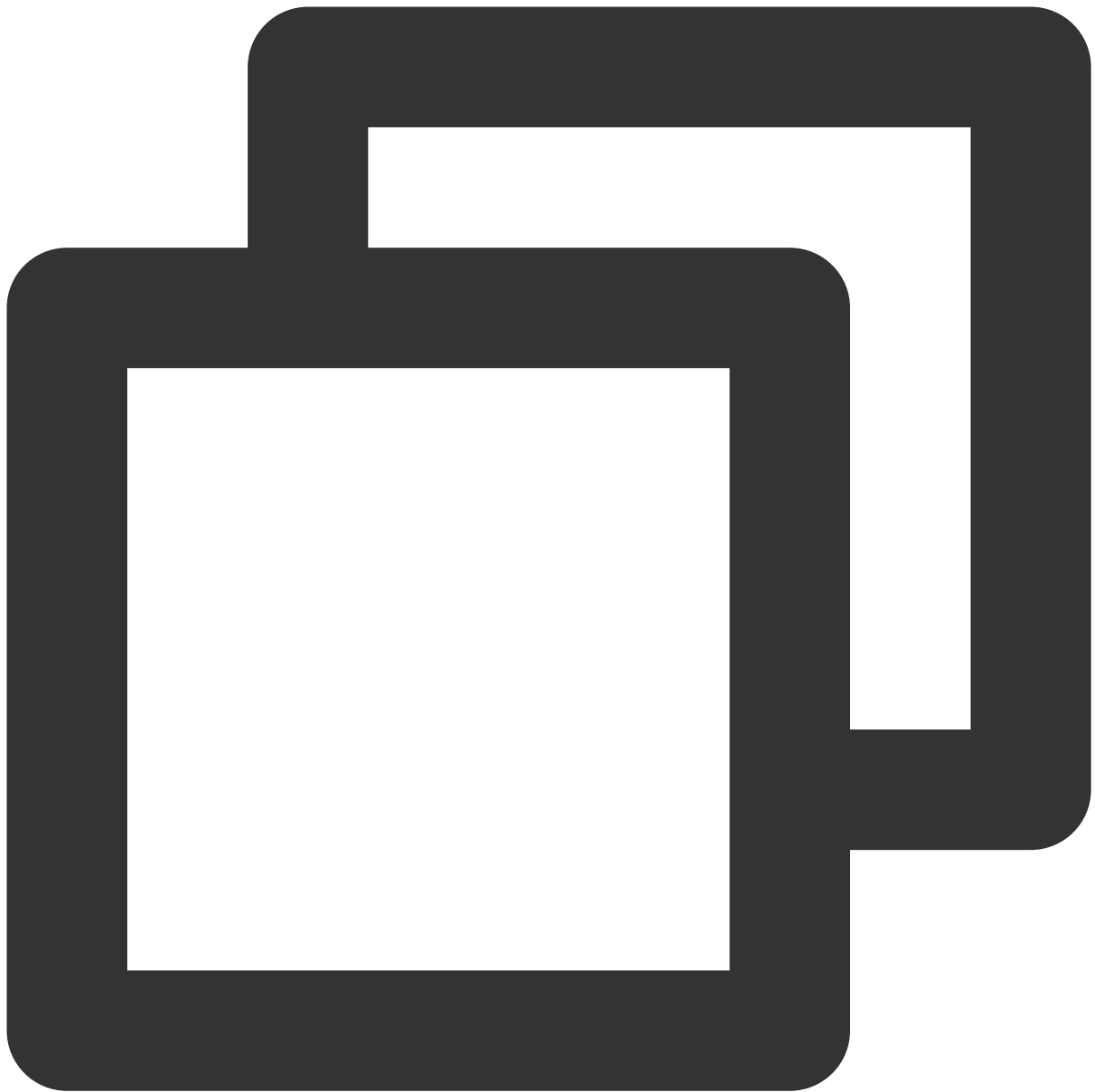
2021-05-07 17:31:41

查询

分区ID	位点	时间戳
0	205	2021-05-07 17:31:41.003
0	206	2021-05-07 17:31:41.058
0	207	2021-05-07 17:31:41.000

步骤5：订阅消息

1. 创建消费消息程序 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',
  // 设置入口服务，请通过控制台获取对应的服务地址。
  'bootstrap.servers': config['bootstrap_servers'],
  'group.id' : config['group_id'],
```

```
// 使用 Kafka 消费分组机制时，消费者超时时间。当 Broker 在该时间内没有收到消费者的心跳时
// 认为该消费者故障失败，Broker 发起重新 Rebalance 过程。
'session.timeout.ms': 10000,
// 客户端请求超时时间，如果超过这个时间没有收到应答，则请求超时失败
'metadadata.request.timeout.ms': 305000,
// 设置客户端内部重试间隔。
'reconnect.backoff.max.ms': 3000

});

consumer.connect();

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

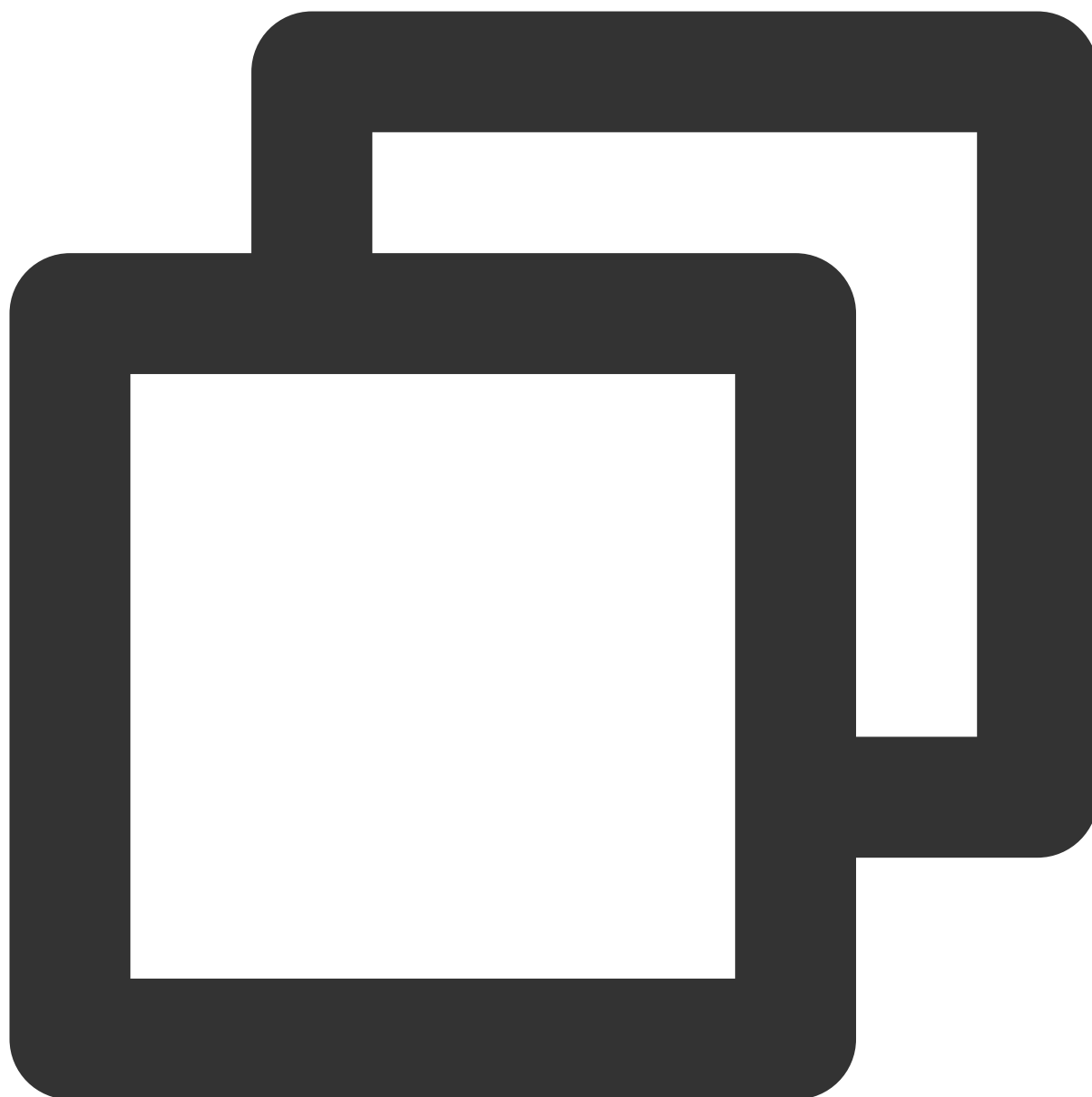
consumer.on('data', function(data) {
  console.log(data);
});

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. 执行以下命令消费消息。



```
node consumer.js
```

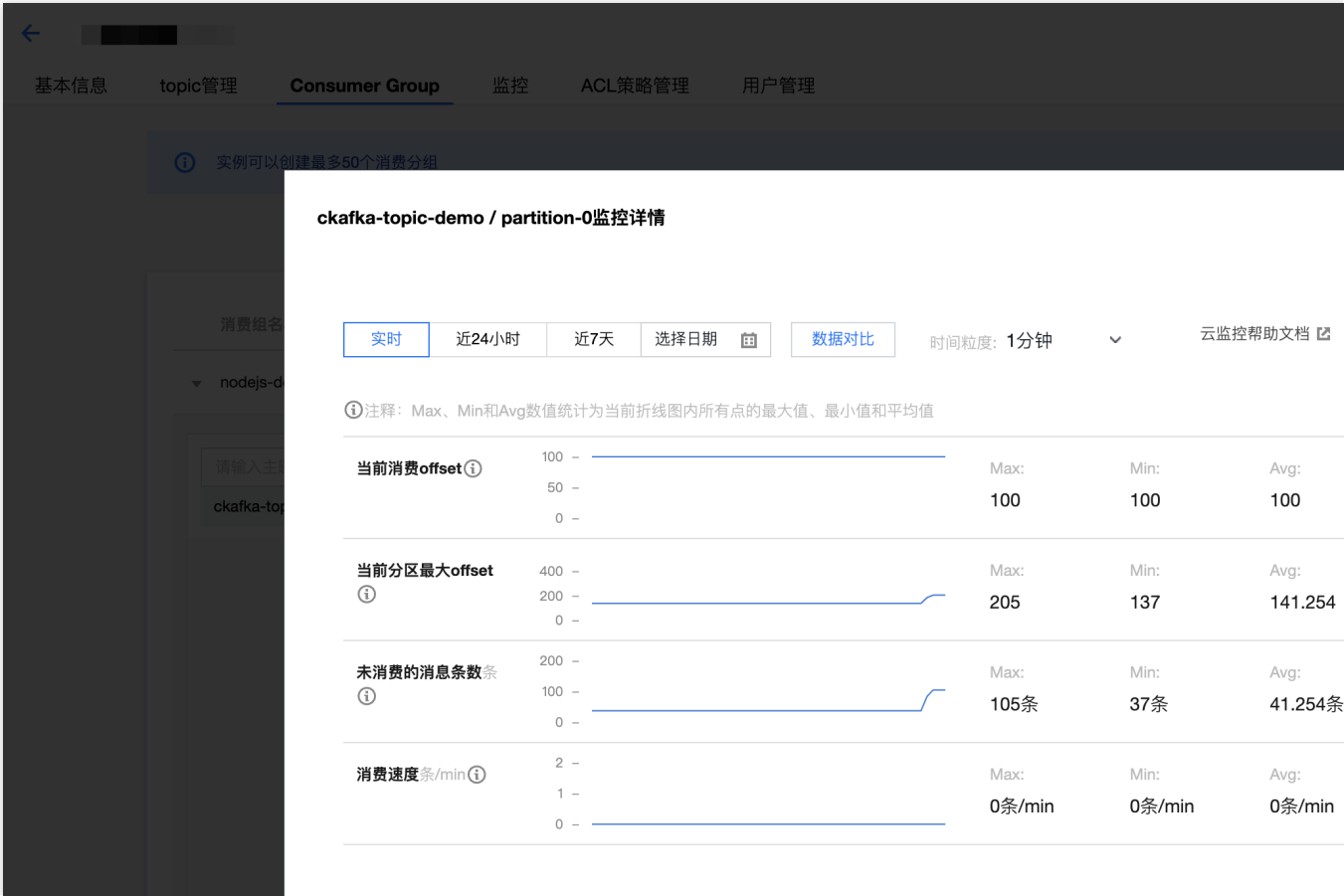
3. 查看运行结果。

```

MB2 ~/Demos/ckafka-demo/nodejskafkademo/sasl  ckafka_demo  node consumer.js
[ 'gzip', 'snappy', 'sasl', 'regex', 'lz4' ]
0.9.5
{ sasl_plain_username: 'ckafka-ckafkademo',
  sasl_plain_password: 'ckafkademo123',
  bootstrap_servers:
    [ 'ckafka-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. 在 [CKafka 控制台](#) Consumer Group 页面，选择对应的消费组，在主题名称输入 Topic 名称，单击[查询详情](#)，查看消费详情。



公网 SASL_PLAINTEXT 方式接入

最近更新时间：2024-05-31 14:40:49

操作场景

该任务以 Node.js 客户端为例，指导您使用公网 SASL_PLAINTEXT 方式接入消息队列 CKafka 版并收发消息。

前提条件

[安装 GCC](#)

[安装 Node.js](#)

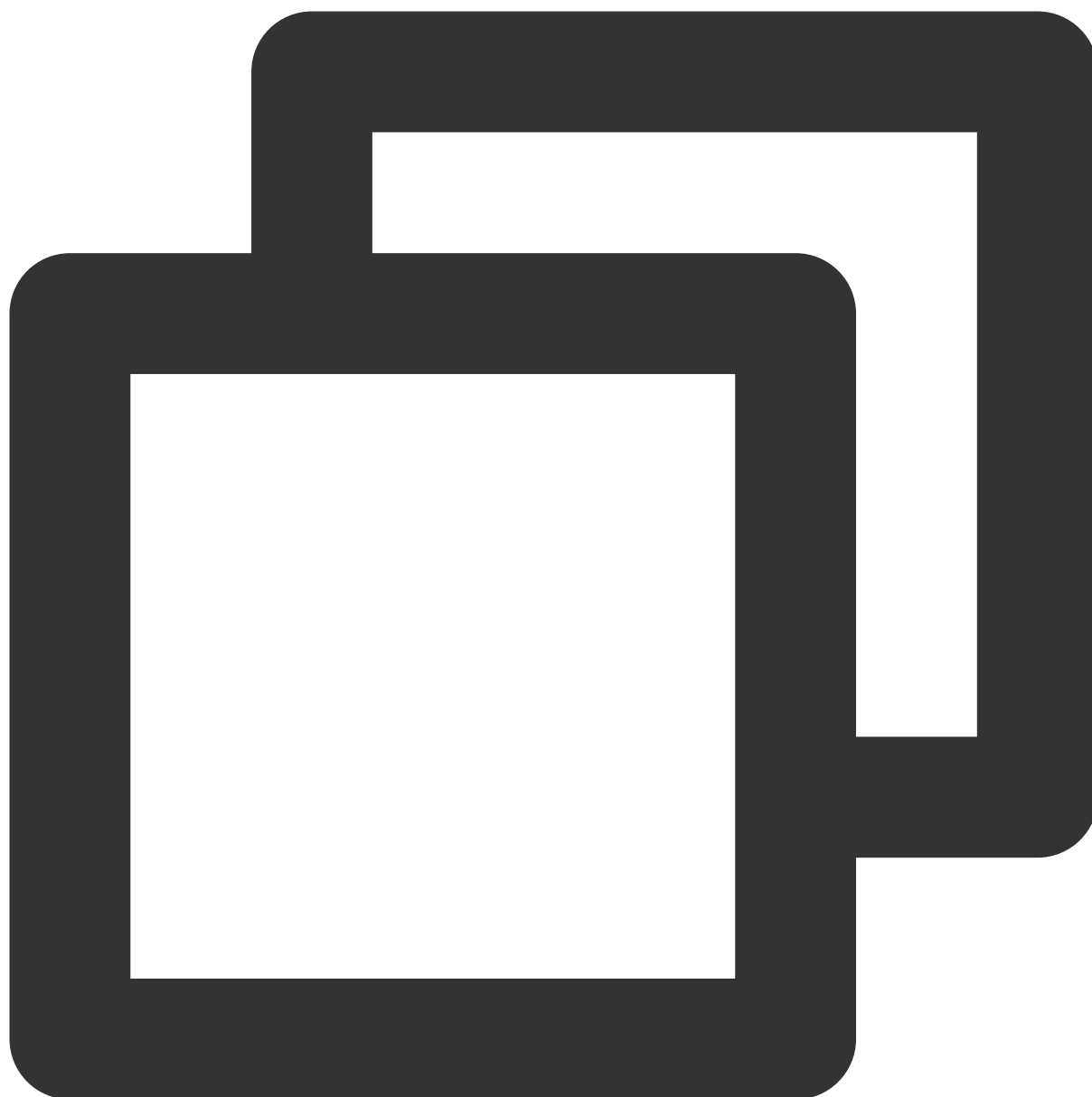
[配置 ACL 策略](#)

[下载 Demo](#)

操作步骤

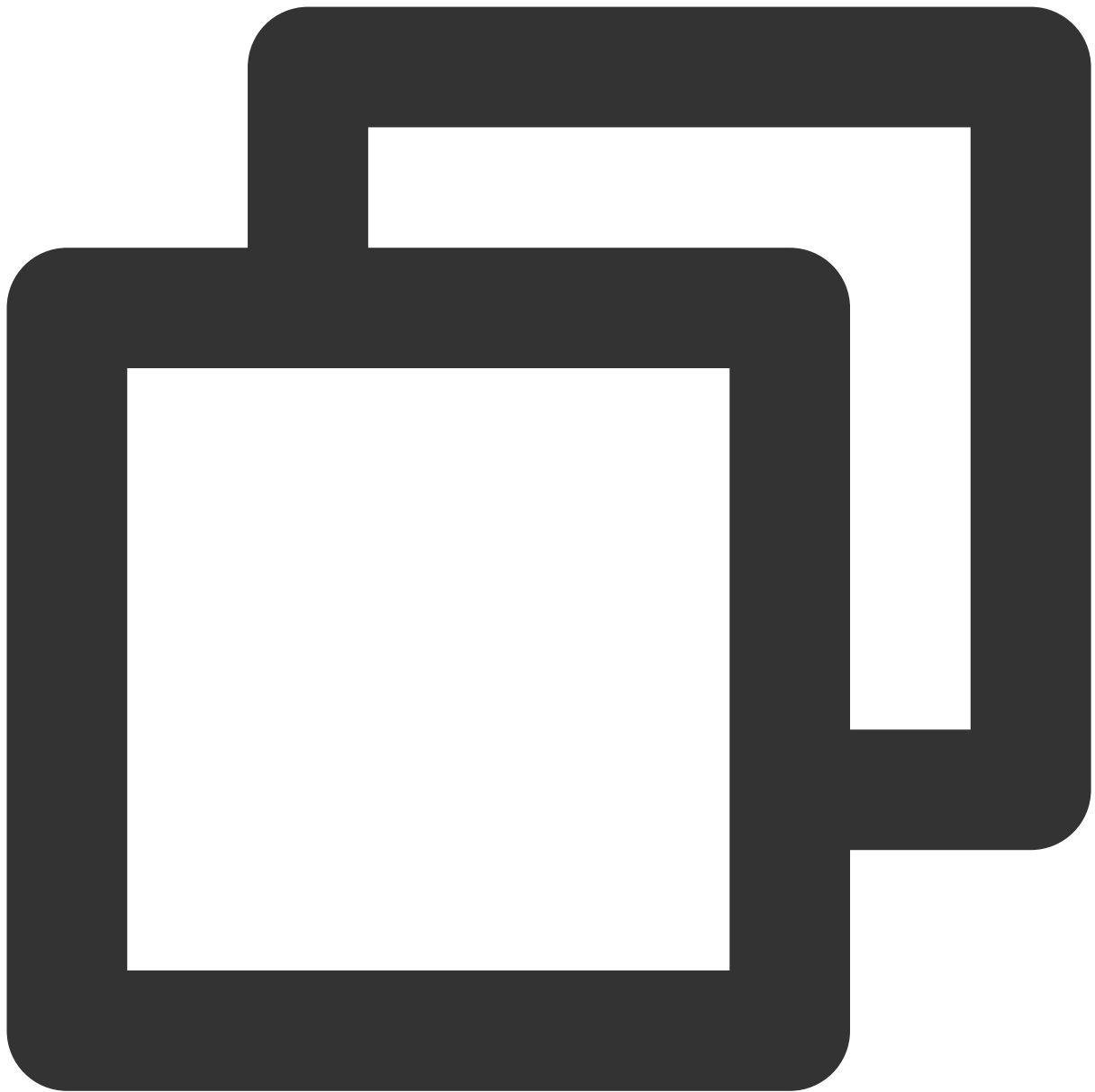
步骤1：安装 C++ 依赖库

1. 执行以下命令切换到 yum 源配置目录 `/etc/yum.repos.d/` 。



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

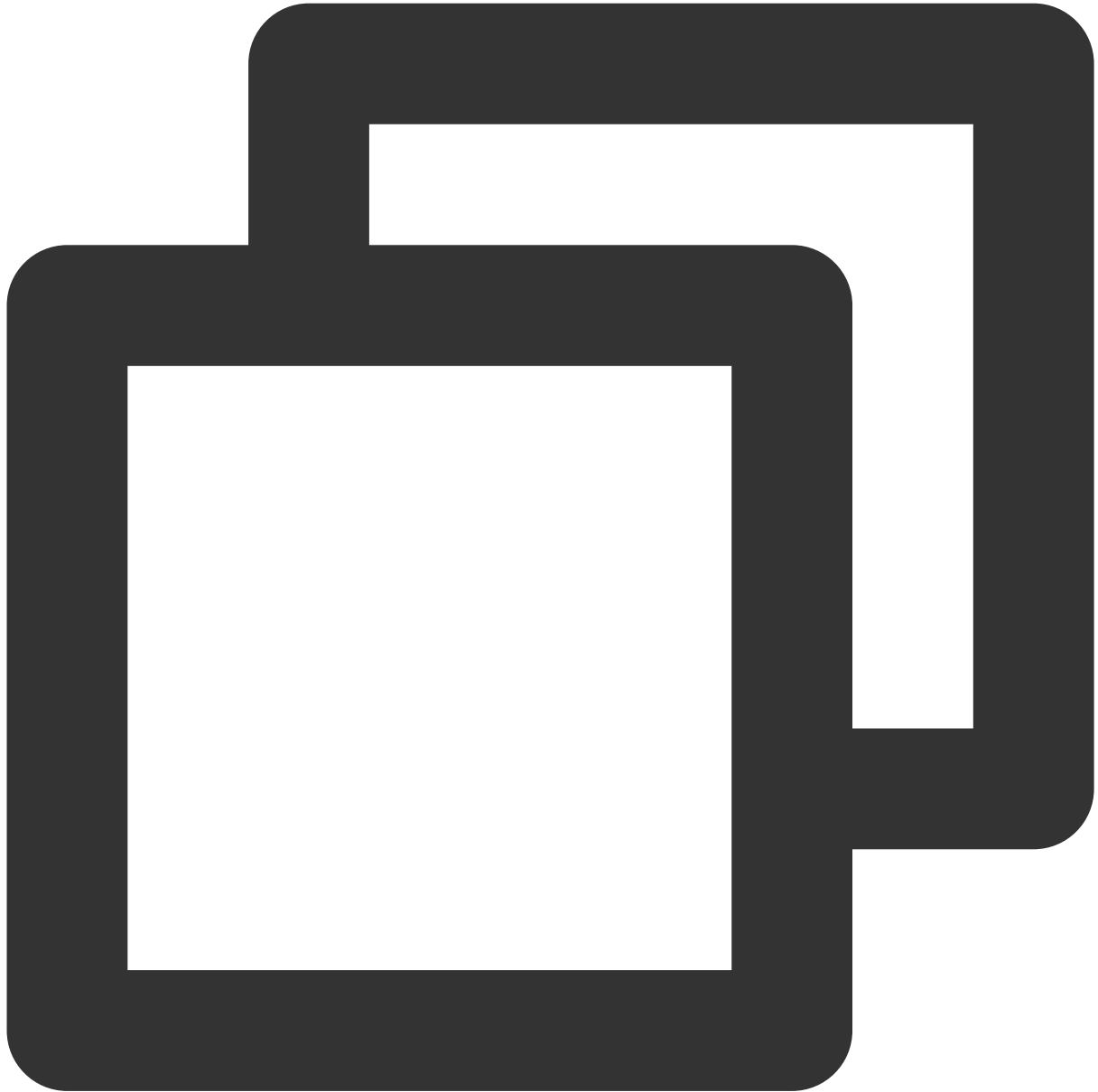
2. 创建 yum 源配置文件 `confluent.repo`。



```
[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
```

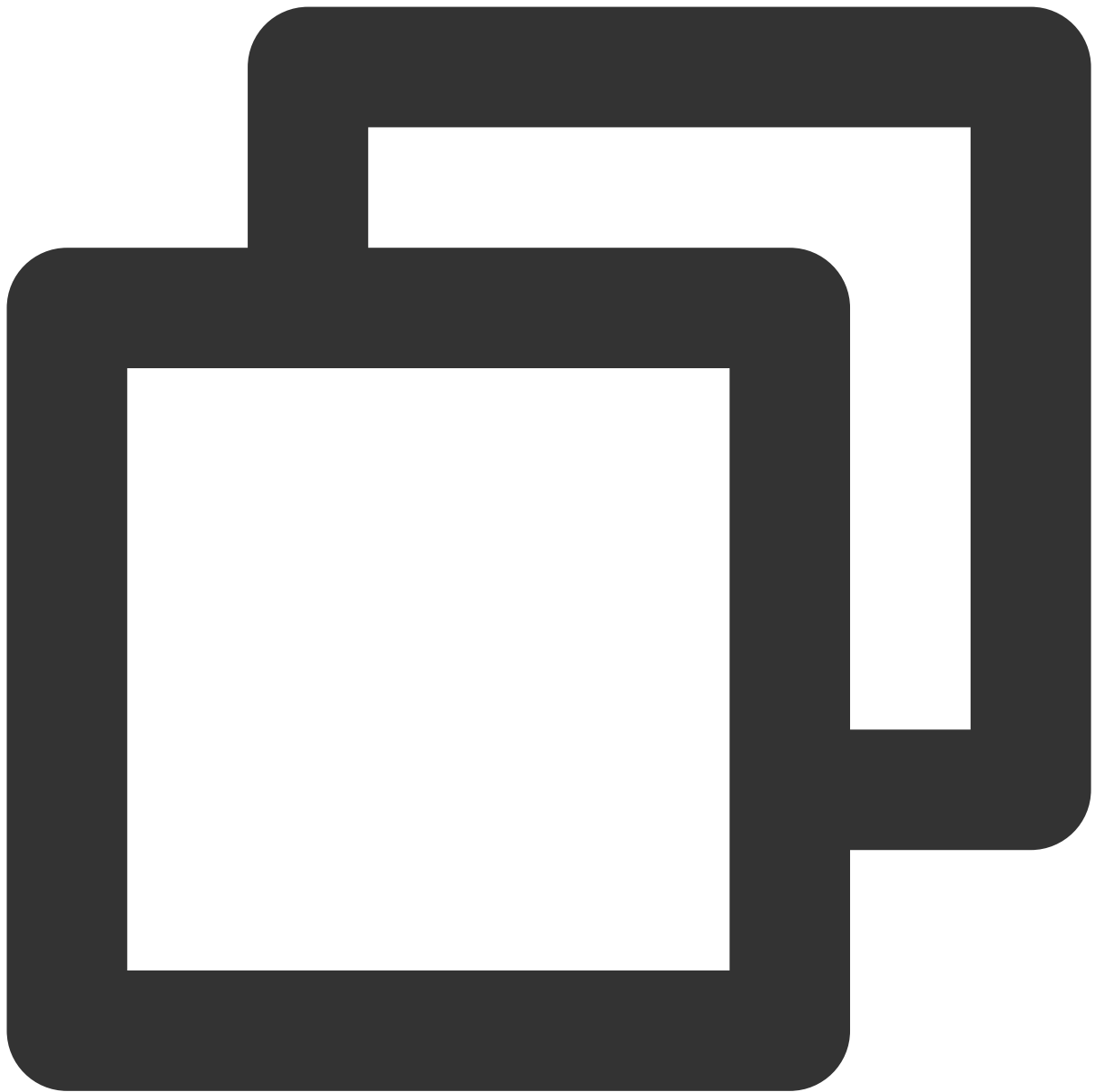
3. 执行以下命令安装 C++ 依赖库。



```
yum install librdkafka-devel
```

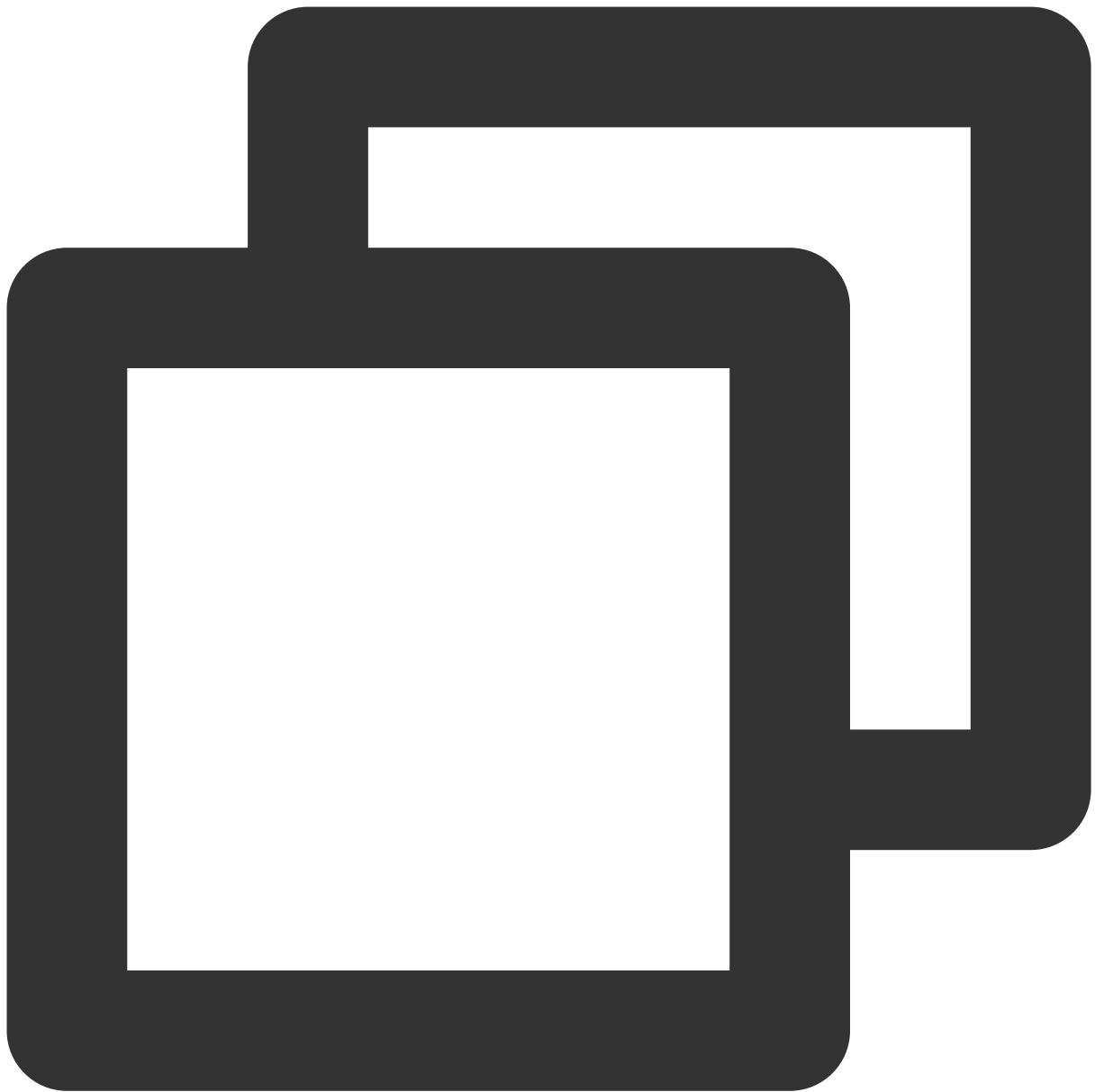
步骤2：安装 Node.js 依赖库

1. 执行以下命令为预处理器指定 OpenSSL 头文件路径。



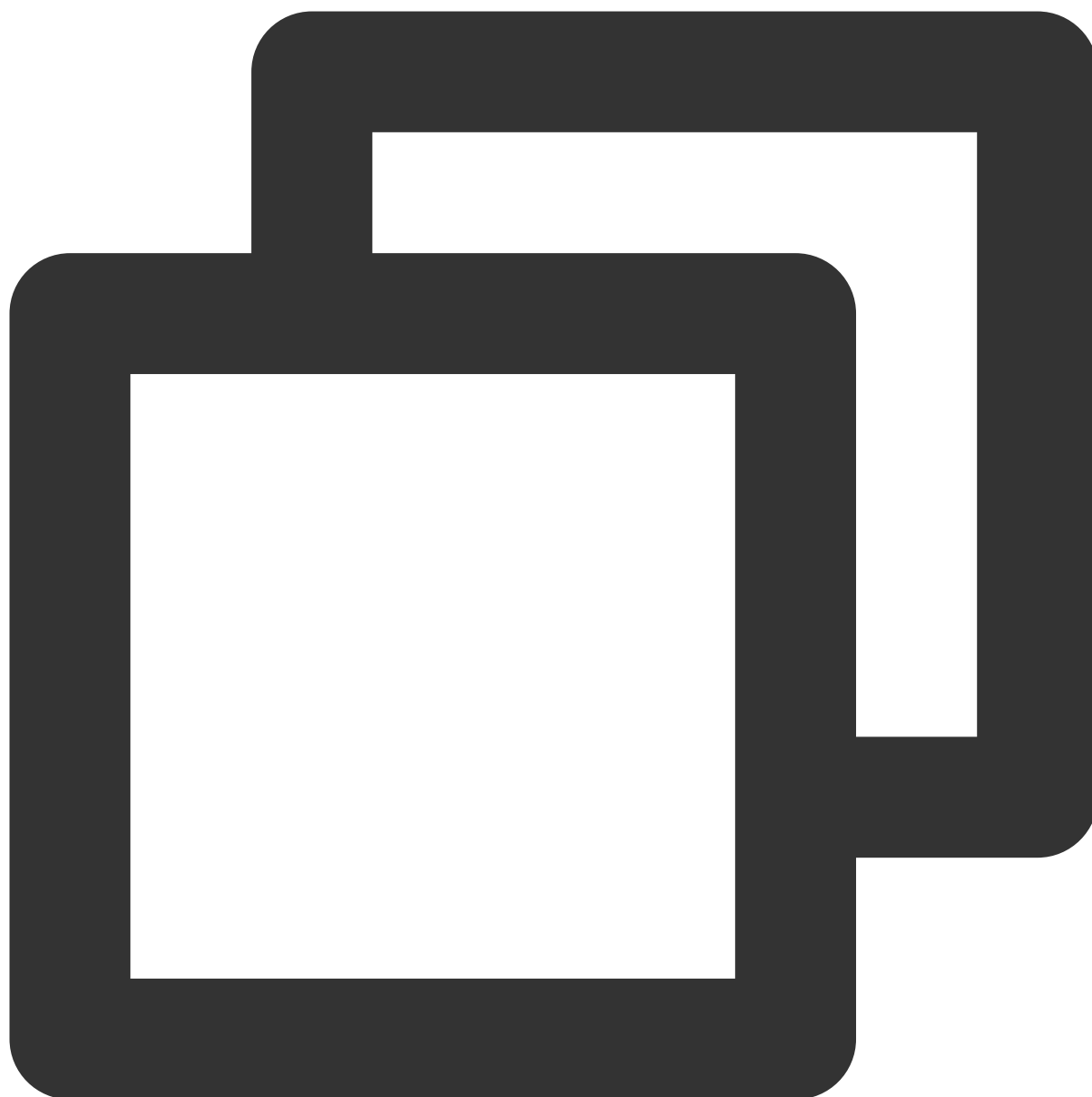
```
export CPPFLAGS=-I/usr/local/opt/openssl/include
```

2. 执行以下命令为连接器指定 OpenSSL 库路径。



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

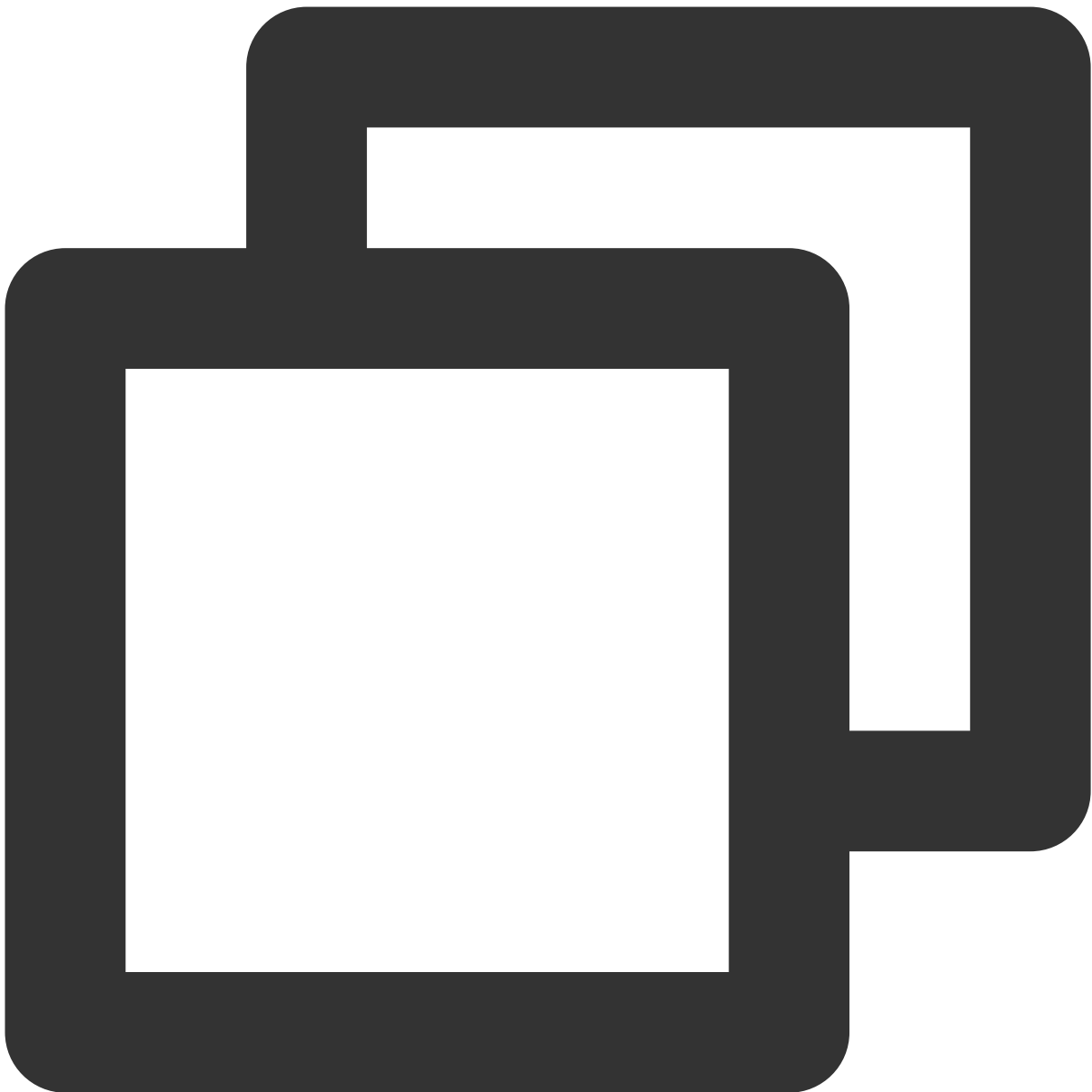
3. 执行以下命令安装 Node.js 依赖库。



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

步骤3：准备配置

创建消息队列 CKafka 版配置文件 `setting.js`。



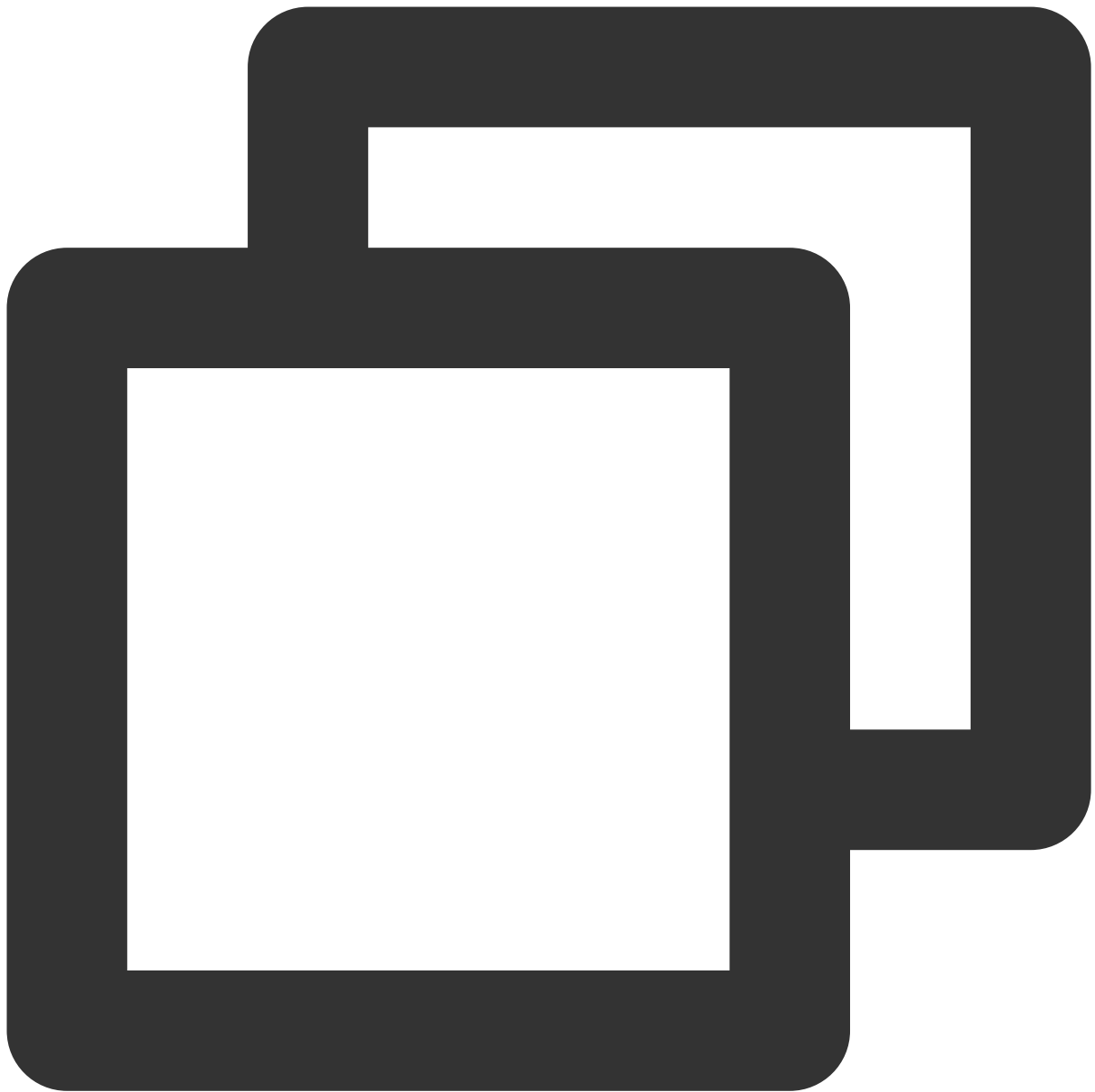
```
module.exports = {
  'sasl_plain_username': 'ckafka-xxxxxxx#ckafkadem',
  'sasl_plain_password': 'ckafkadem123',
  'bootstrap_servers': ["xxx.ckafka.tencentcloudmq.com:6018"],
  'topic_name': 'xxx',
  'group_id': 'xxx'
}
```

参数	描述

sasl_plain_username	用户名，格式为 实例 ID + # + 用户名。实例 ID 在 CKafka 控制台 的实例详情页面获取，用户在 ACL 策略管理 下的 用户管理 创建用户时设置。																		
sasl_plain_password	用户密码，在 CKafka 控制台实例详情页面 ACL 策略管理 下的 用户管理 创建用户时设置。																		
bootstrap_servers	SASL 接入点，在 CKafka 控制台的实例详情页面的基本信息 > 接入方式获取。 接入方式? 添加路由策略 <table><tr><th>接入类型</th><th>接入方式</th><th>网络</th><th>操作</th></tr><tr><td>公网域名接入</td><td>SASL_PLAINTEXT</td><td>ckafka-l4xppqrz...</td><td>删除</td></tr></table>	接入类型	接入方式	网络	操作	公网域名接入	SASL_PLAINTEXT	ckafka-l4xppqrz...	删除										
接入类型	接入方式	网络	操作																
公网域名接入	SASL_PLAINTEXT	ckafka-l4xppqrz...	删除																
topic_name	Topic名称，在 CKafka 控制台实例详情页面的 topic 管理创建和获取。 基本信息topic管理Consumer Group监控事件中心HTTP接入ACL策略管理智能运维专业版 新建(12/450) <table><tr><th>ID/名称</th><th>监控</th><th>分区数(个)</th><th>副本数(个)</th><th>标签</th><th>备注</th></tr><tr><td>topic- [img] 后</td><td>[img]</td><td>100</td><td>3</td><td></td><td></td></tr><tr><td>topic- [img] 后</td><td>[img]</td><td>1</td><td>1</td><td></td><td></td></tr></table>	ID/名称	监控	分区数(个)	副本数(个)	标签	备注	topic- [img] 后	[img]	100	3			topic- [img] 后	[img]	1	1		
ID/名称	监控	分区数(个)	副本数(个)	标签	备注														
topic- [img] 后	[img]	100	3																
topic- [img] 后	[img]	1	1																
group_id	消费者的组 ID，根据业务需求自定义																		

步骤4：发送消息

- 1. 编写生产消息程序 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);
});

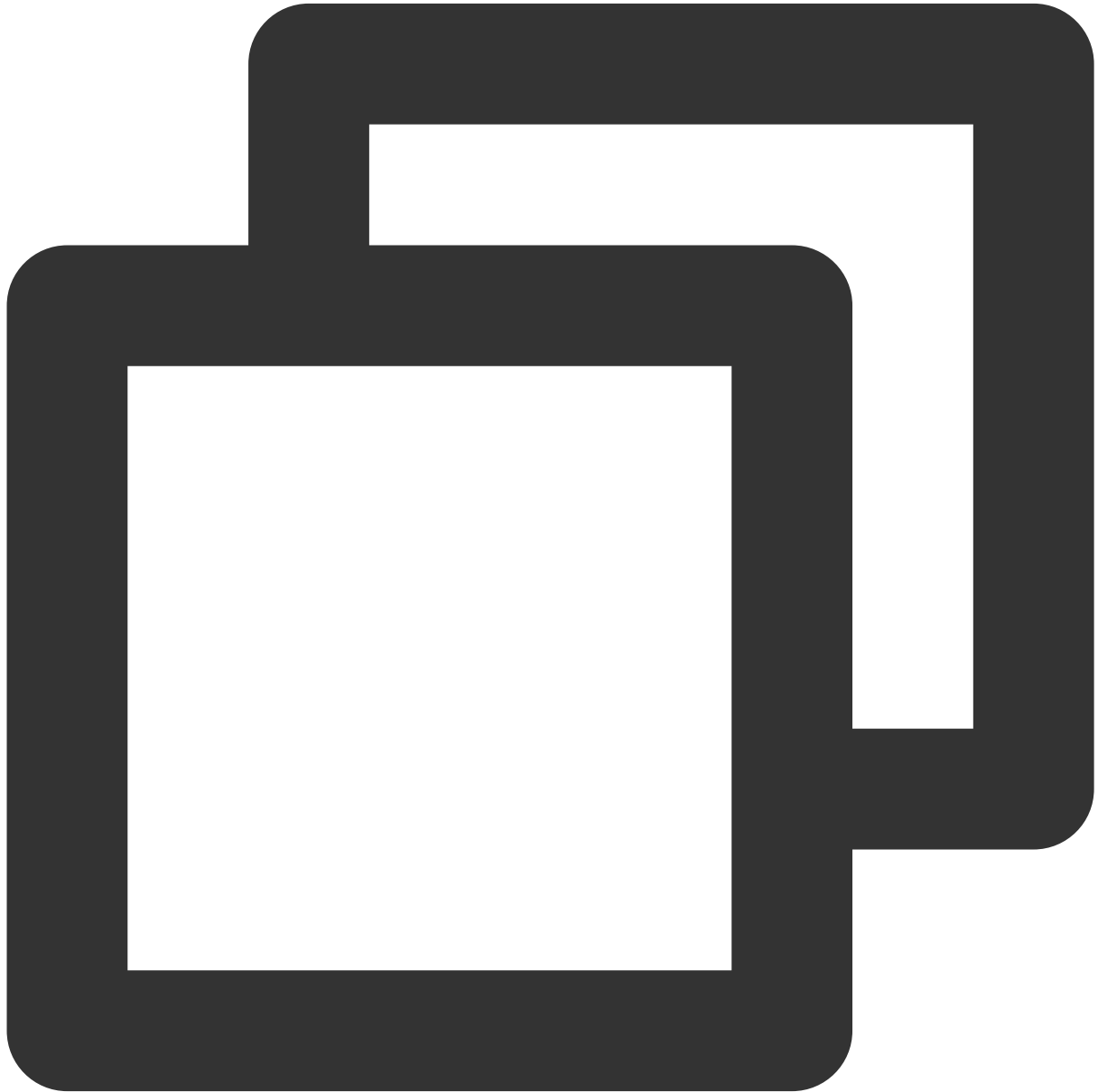
producer.on("error", function(error) {
  console.log("error:" + error);
});

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. 执行以下命令发送消息。



```
node producer.js
```

3. 查看运行结果。

```
~/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
Buffer.allocUnsafe(), Buffer.allocUnsafeSlow(), 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
delivery-report: producer ok
delivery-report: producer ok
delivery-report: producer ok
delivery-report: producer ok
```

4. 在 [CKafka 控制台](#) **topic 管理** 页面，选择对应的 Topic，单击 **更多 > 消息查询**，查看刚发送的消息。

消息查询

消息查询会占用CKafka实例的带宽资源，建议您尽量缩小查询范围查询，不要频繁操作。
消息查询最多展示最近的20条消息。

实例

ckafka

Topic

ckafka-topic-demo

查询类型

按位点查询

按起始时间查询

分区ID

0

时间

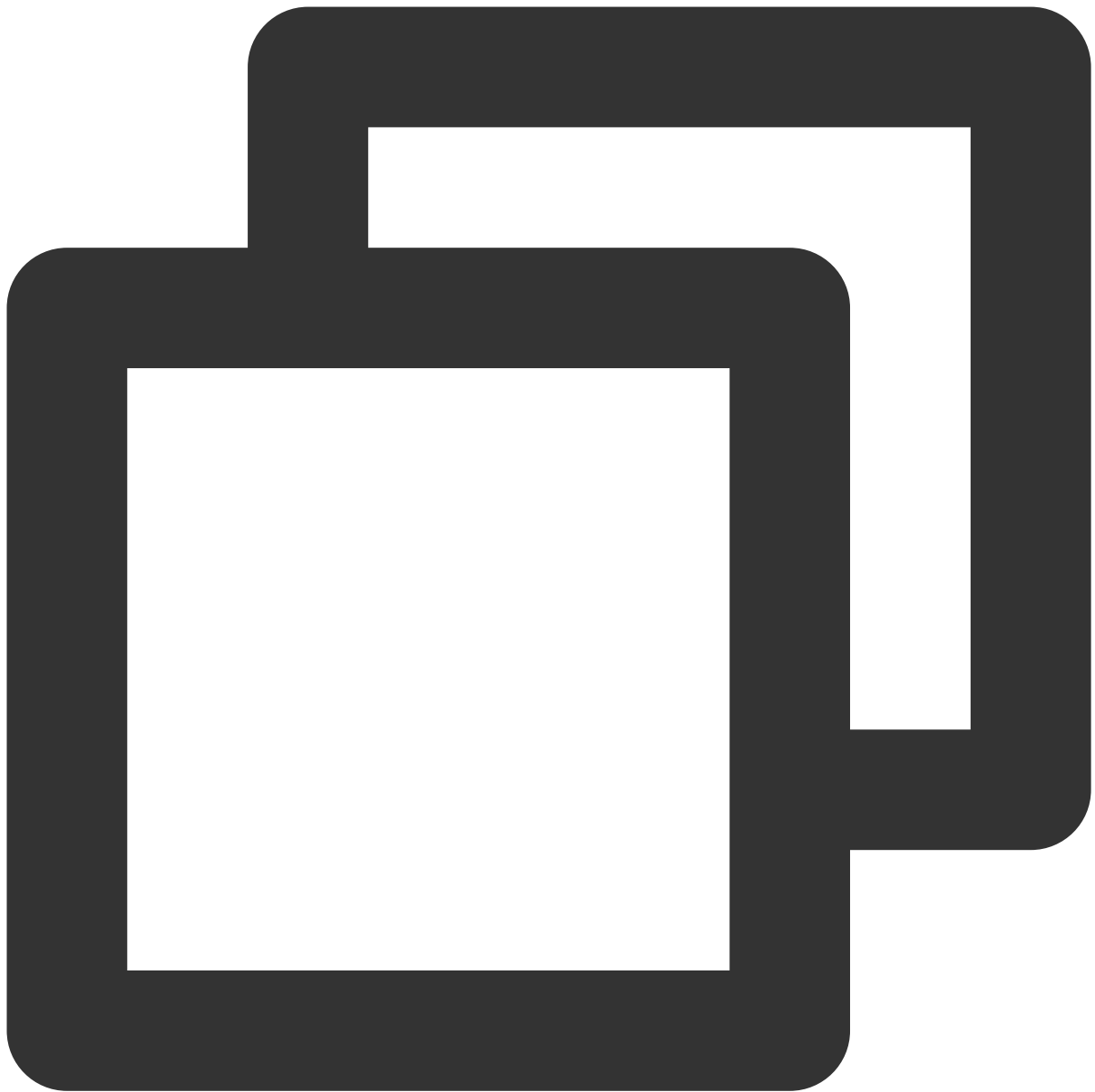
2021-05-07 17:31:41

查询

分区ID	位点	时间戳
0	205	2021-05-07 17:31:41.000
0	206	2021-05-07 17:31:41.058
0	207	2021-05-07 17:31:41.000

步骤5：订阅消息

1. 创建消费消息程序 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)

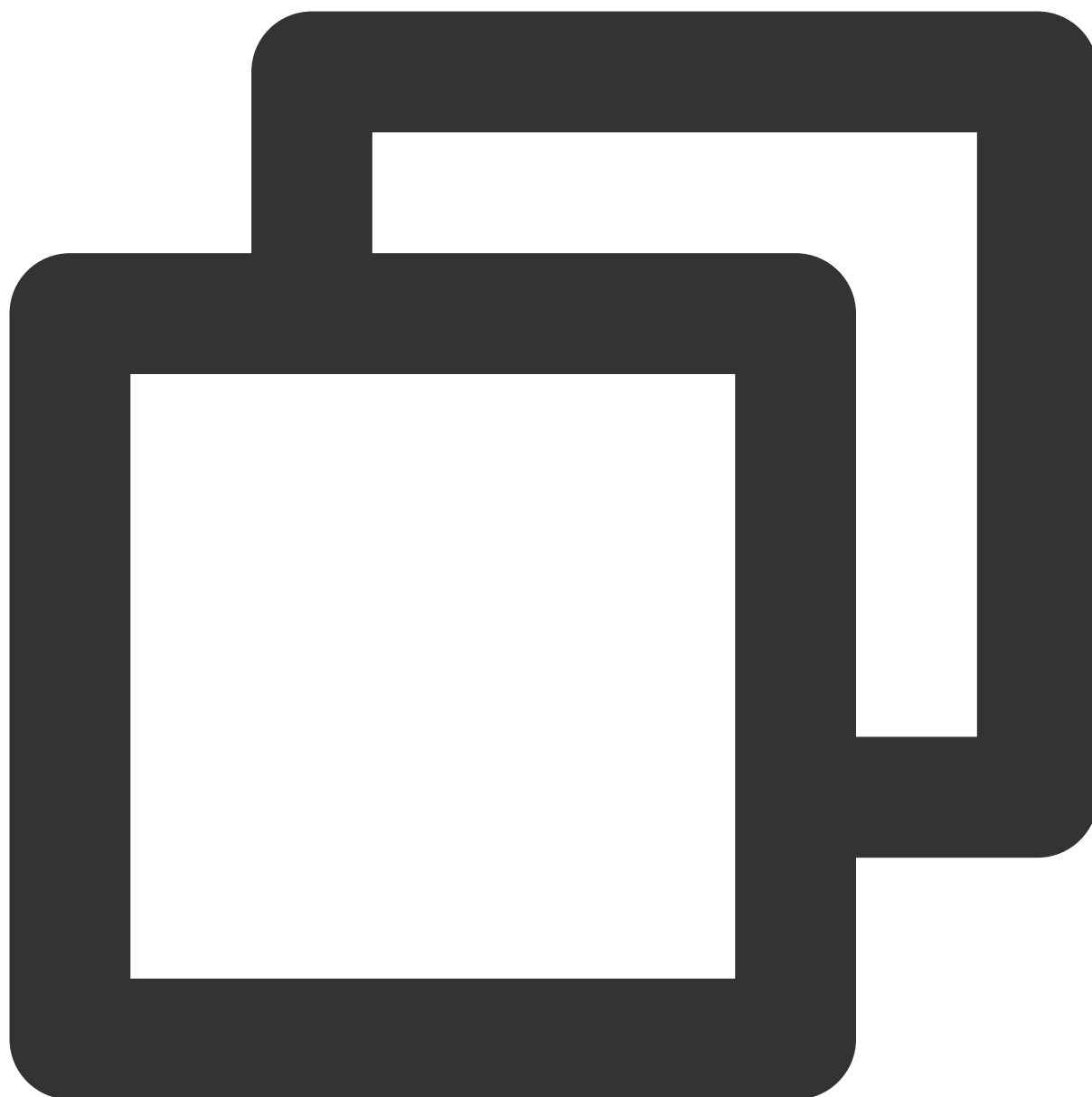
var consumer = new Kafka.KafkaConsumer({
  'api.version.request': 'true',
  'bootstrap.servers': config['bootstrap_servers'],
  'security.protocol' : 'SASL_PLAINTEXT',
  'sasl.mechanisms' : 'PLAIN',
  'message.max.bytes': 32000,
  'fetch.message.max.bytes': 32000,
  'max.partition.fetch.bytes': 32000,
  '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. 执行以下命令消费消息。



```
node consumer.js
```

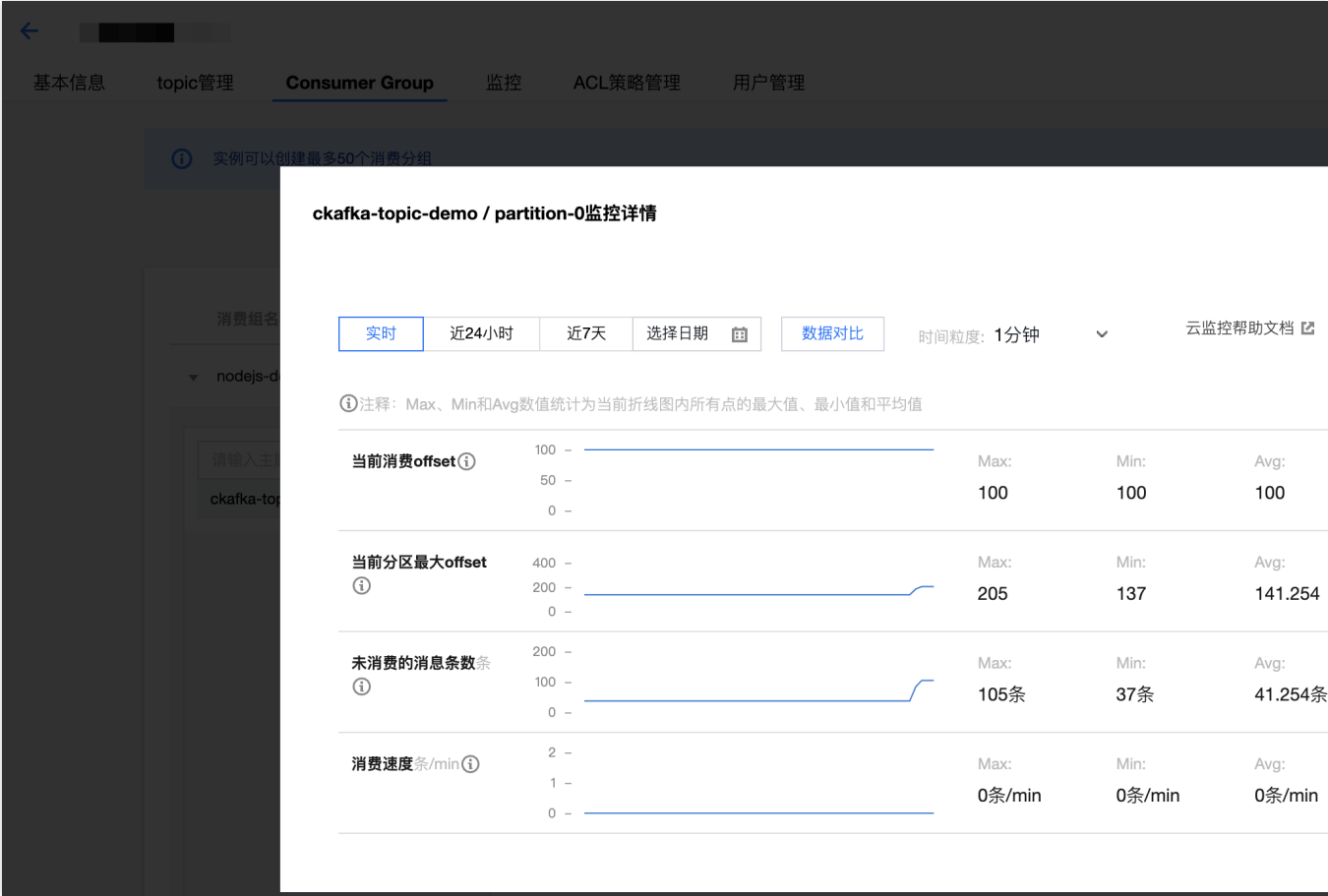
3. 查看运行结果。

```

MB2 ~ /Demos/ckafka-demo/nodejskafkademo/sasl c kafka_demo node consumer.js
[ 'gzip', 'snappy', 'sasl', 'regex', 'lz4' ]
0.9.5
{ sasl_plain_username: 'ckafka-ckafka',
  sasl_plain_password: 'ckafkademo123',
  bootstrap_servers:
    [ 'ckafka-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. 在 [CKafka 控制台](#) Consumer Group 页面，选择对应的消费组名称，在主题名称输入 Topic 名称，单击[查询详情](#)，查看消费详情。



连接器相关 SDK

数据上报 SDK

最近更新时间：2024-01-09 15:00:32

操作场景

本文档以 Java 语言为例，介绍客户端通过集成 Java 版本的数据上报 SDK 快捷地将数据上报到 Datahub 中的操作方法。

操作步骤

步骤1：创建 HTTP 接入点

参见 [创建 HTTP 上报接入点](#) 在 Datahub 控制台创建一个 HTTP 接入点，获取到标识上报 EndPoint 的DatahubId。

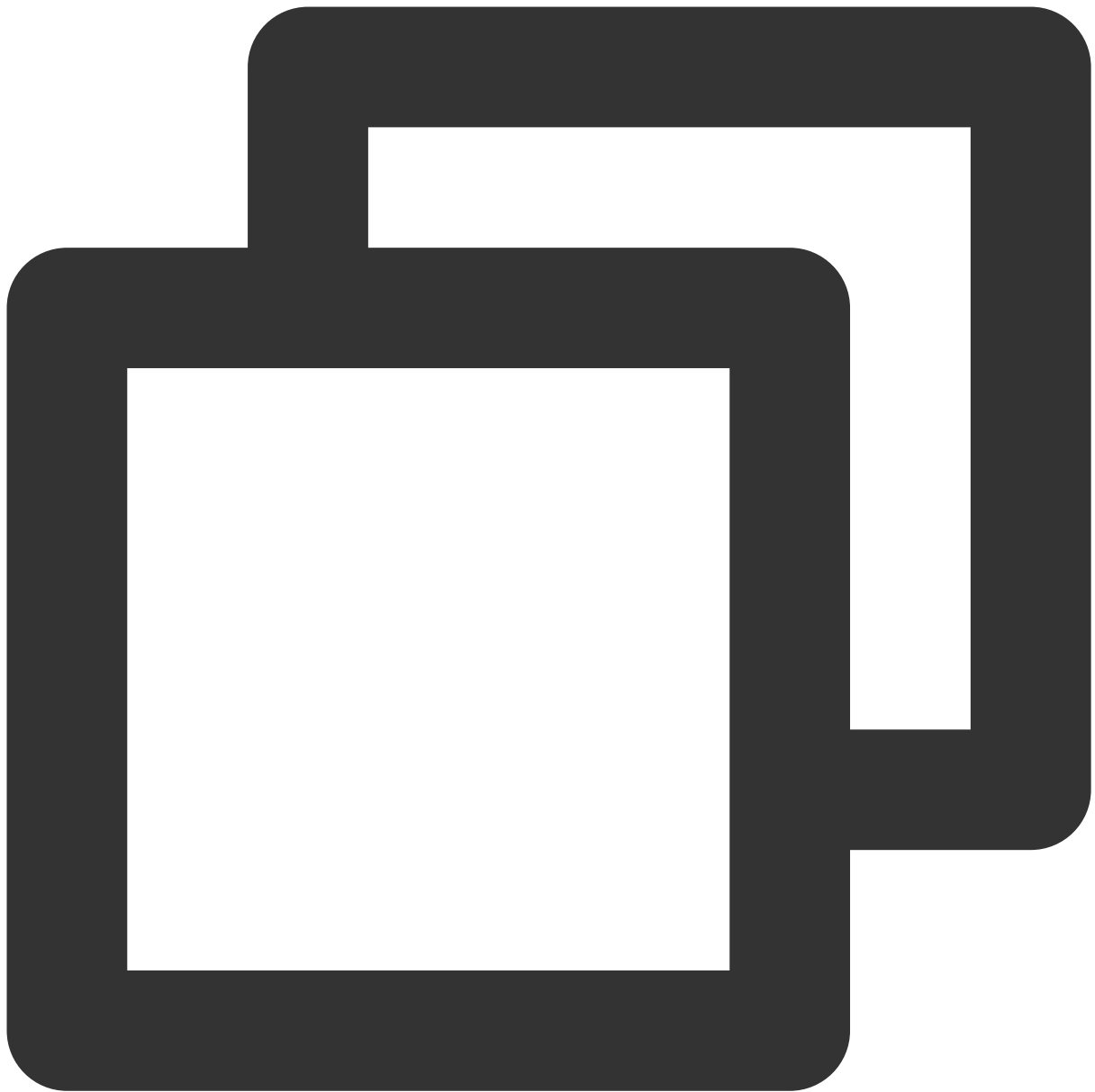
步骤2：引入 Java SDK

在 Java 项目通过 Maven、Gradle 等方式引入数据上报 SDK。

步骤3：数据上报

引入 SDK 后，可以通过调用 SDK 的 SendMessage 接口单条/批量上报数据，整体分为四步：

1. 实例化认证对象。
2. 实例化 Client 对象。
3. SendMessage 请求上报数据。
4. 处理返回结果。



```
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{
```

```
// 实例化一个认证对象, 入参需要传入腾讯云账户secretId, secretKey, 此处还需注意密钥
// 密钥可前往https://console.tencentcloud.com/cam/capi网站进行获取
Credential cred = new Credential("SecretId", "SecretKey");
// 实例化一个http选项, 可选的, 没有特殊需求可以跳过
HttpProfile httpProfile = new HttpProfile();
httpProfile.setEndpoint("ckafka.tencentcloudapi.com");
// 实例化一个client选项, 可选的, 没有特殊需求可以跳过
ClientProfile clientProfile = new ClientProfile();
clientProfile.setHttpProfile(httpProfile);
// 实例化要请求产品的client对象, clientProfile是可选的
CkafkaClient client = new CkafkaClient(cred, "ap-beijing", clientProfile);
// 实例化一个请求对象, 每个接口都会对应一个request对象
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);

// 返回的resp是一个SendMessageResponse的实例, 与请求对象对应
SendMessageResponse resp = client.SendMessage(req);
// 输出json格式的字符串回包
System.out.println(SendMessageResponse.toJsonString(resp));
} catch (TencentCloudSDKException e) {
    System.out.println(e.toString());
}
}
```

步骤4：消息查询

数据发送成功后, 可以在消息查询页面, 查看数据是否发送成功, 详情请参见 [消息查询](#)。

源码 DEMO

Java

Python

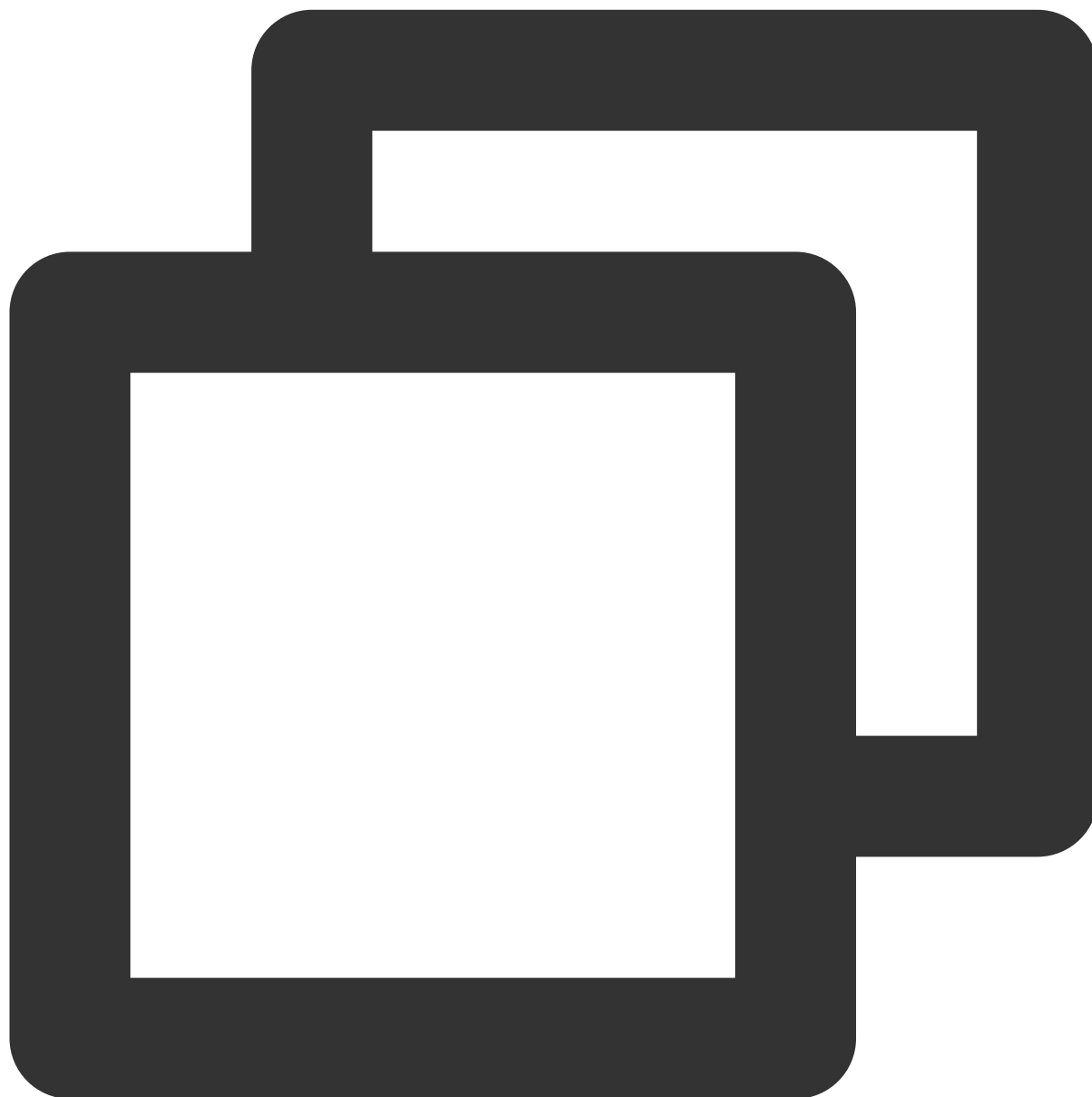
Node.JS

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.*;

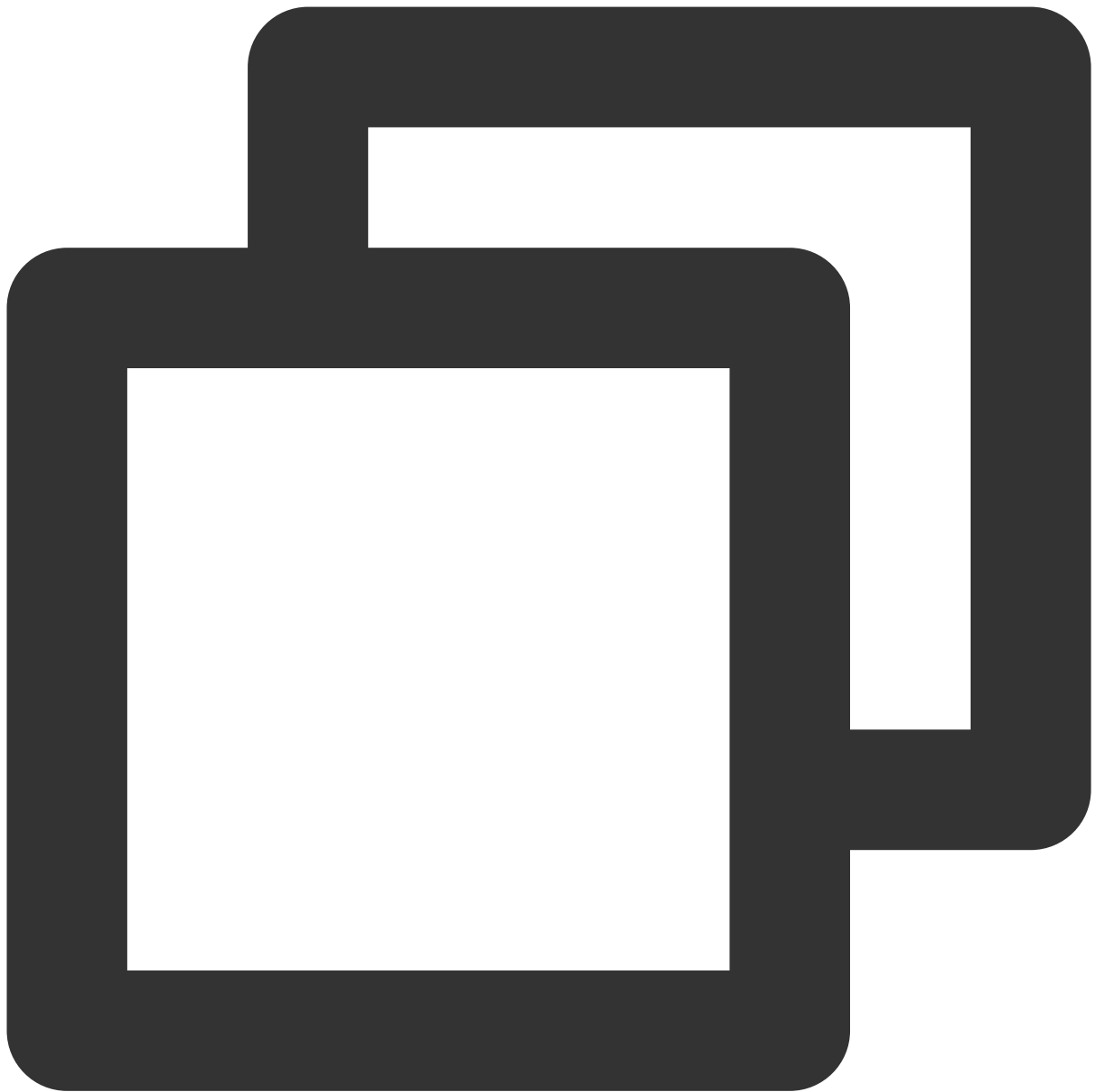
public class SendMessage
{
    public static void main(String [] args) {
        try{
            // 实例化一个认证对象，入参需要传入腾讯云账户secretId, secretKey,此处还需注意密钥
            // 密钥可前往https://console.tencentcloud.com/cam/capi网站进行获取
            Credential cred = new Credential("SecretId", "SecretKey");
            // 实例化一个http选项，可选的，没有特殊需求可以跳过
            HttpProfile httpProfile = new HttpProfile();
            httpProfile.setEndpoint("ckafka.tencentcloudapi.com");
            // 实例化一个client选项，可选的，没有特殊需求可以跳过
            ClientProfile clientProfile = new ClientProfile();
            clientProfile.setHttpProfile(httpProfile);
            // 实例化要请求产品的client对象,clientProfile是可选的
            CkafkaClient client = new CkafkaClient(cred, "ap-beijing", clientProfile);
            // 实例化一个请求对象,每个接口都会对应一个request对象
            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);

            // 返回的resp是一个SendMessageResponse的实例，与请求对象对应
            SendMessageResponse resp = client.SendMessage(req);
            // 输出json格式的字符串回包
            System.out.println(SendMessageResponse.toJsonString(resp));
        } catch (TencentCloudSDKException e) {
            System.out.println(e.toString());
        }
    }
}
```



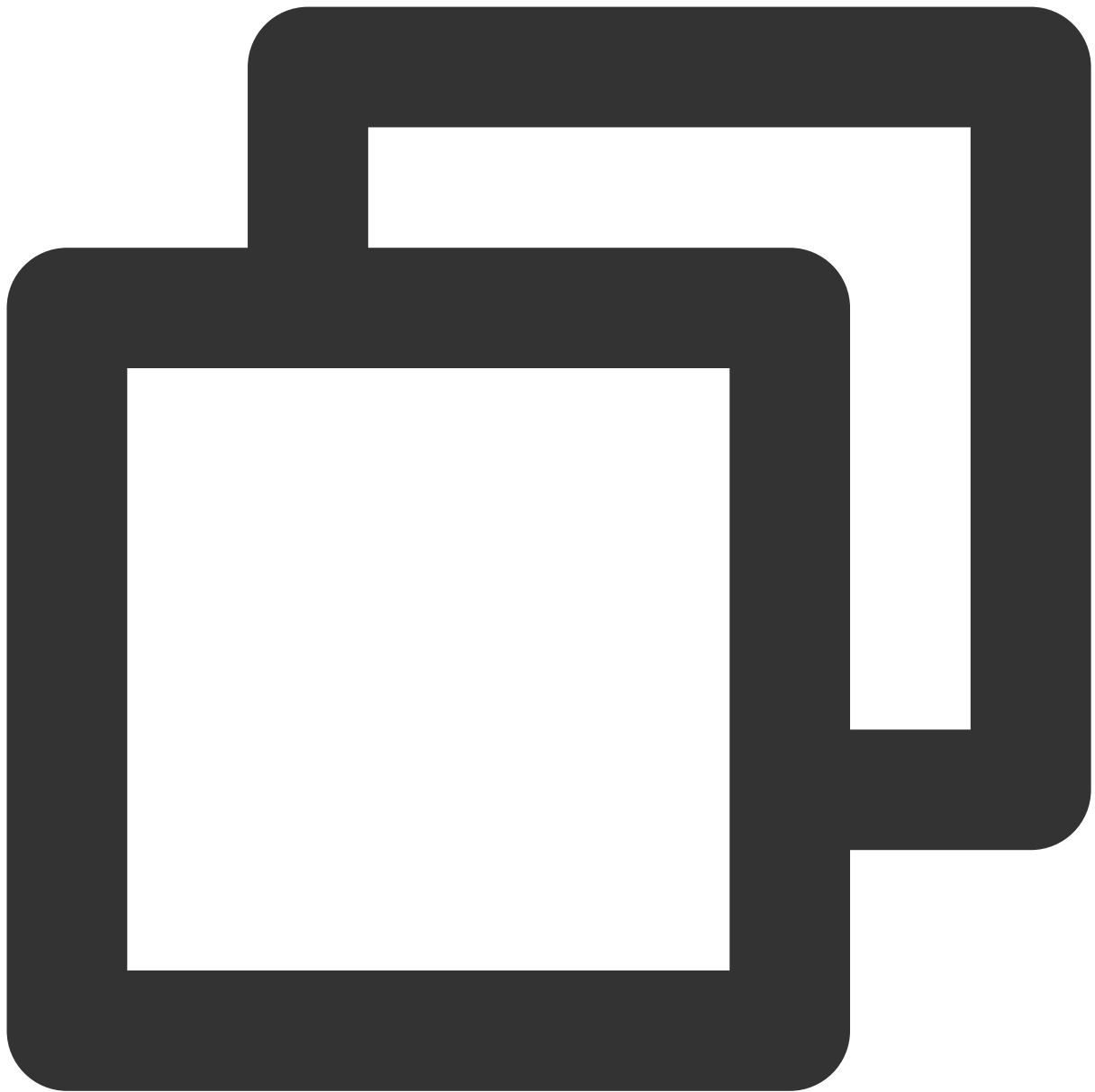
```
import json
from tencentcloud.common import credential
from tencentcloud.common.profile.client_profile import ClientProfile
from tencentcloud.common.profile.http_profile import HttpProfile
from tencentcloud.common.exception.tencent_cloud_sdk_exception import TencentCloudS
from tencentcloud.ckafka.v20190819 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-r6gknngy3",
    "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");

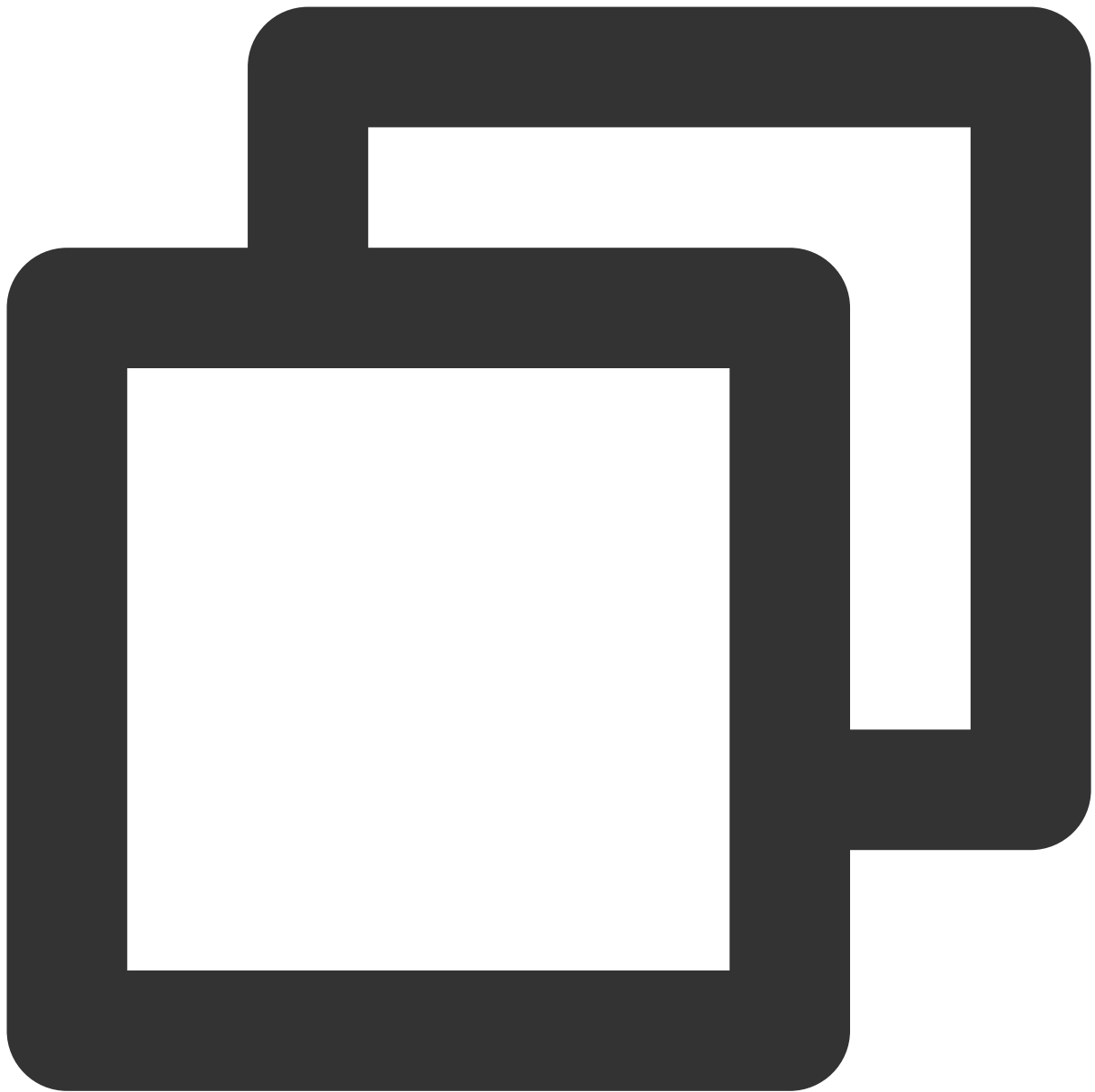
const CkafkaClient = tencentcloud.ckafka.v20190819.Client;

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");

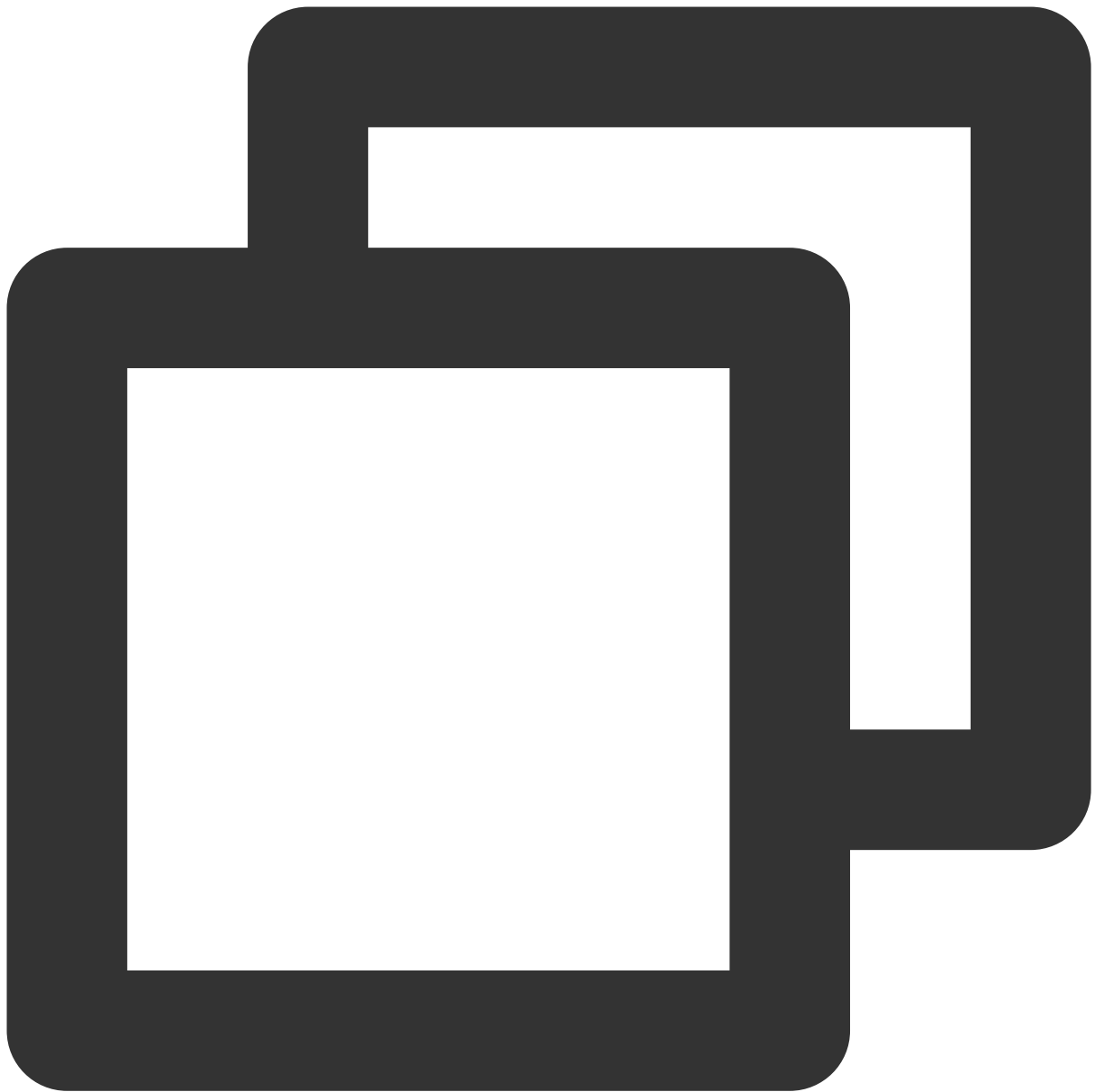
$clientProfile = new ClientProfile();
$clientProfile->setHttpProfile($httpProfile);
$client = new CkafkaClient($cred, "ap-beijing", $clientProfile);

$req = new SendMessageRequest();

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

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

print_r($res->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() {

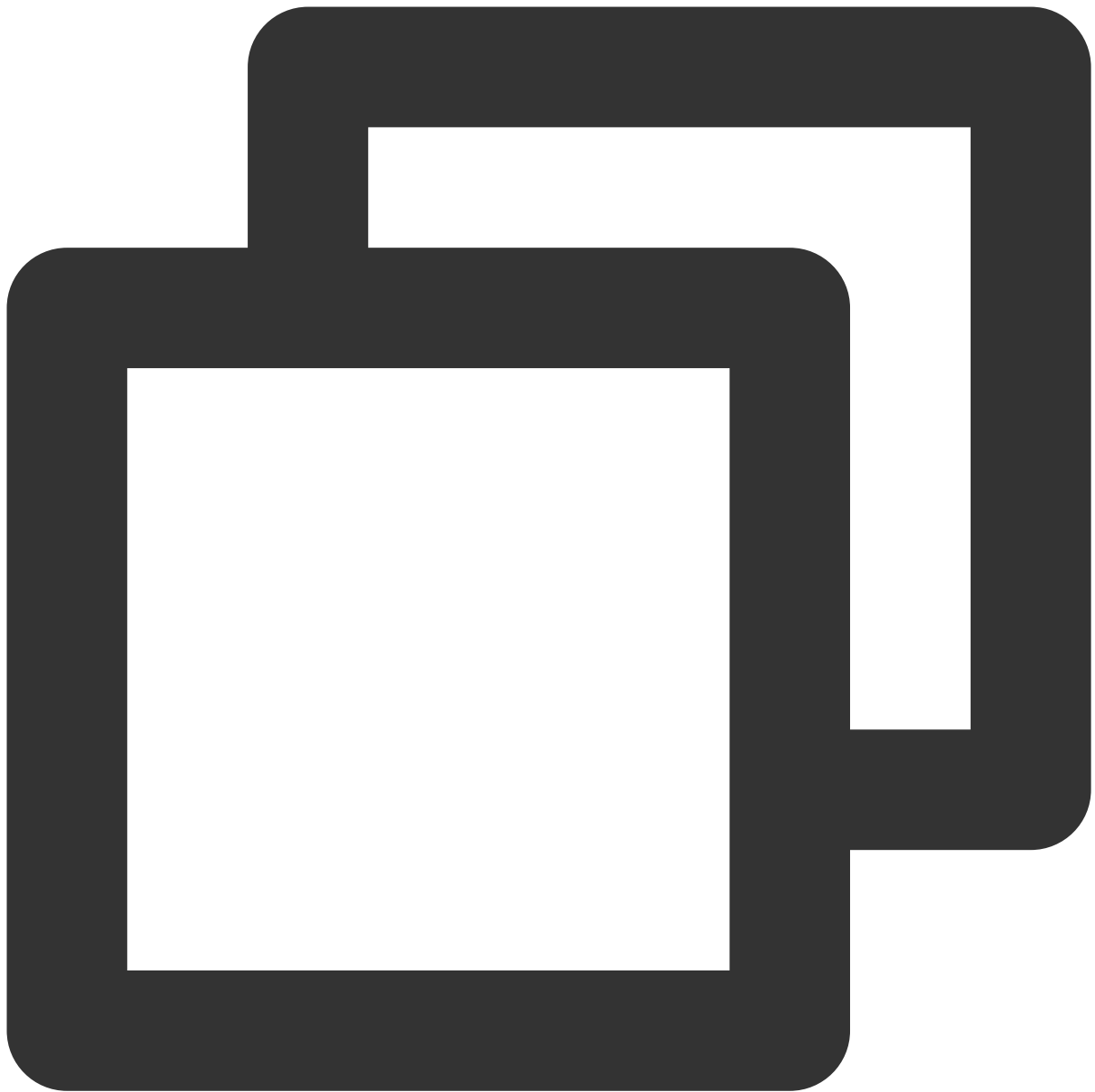
    credential := common.NewCredential(
        "SecretId",
        "SecretKey",
    )

    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
    {

```

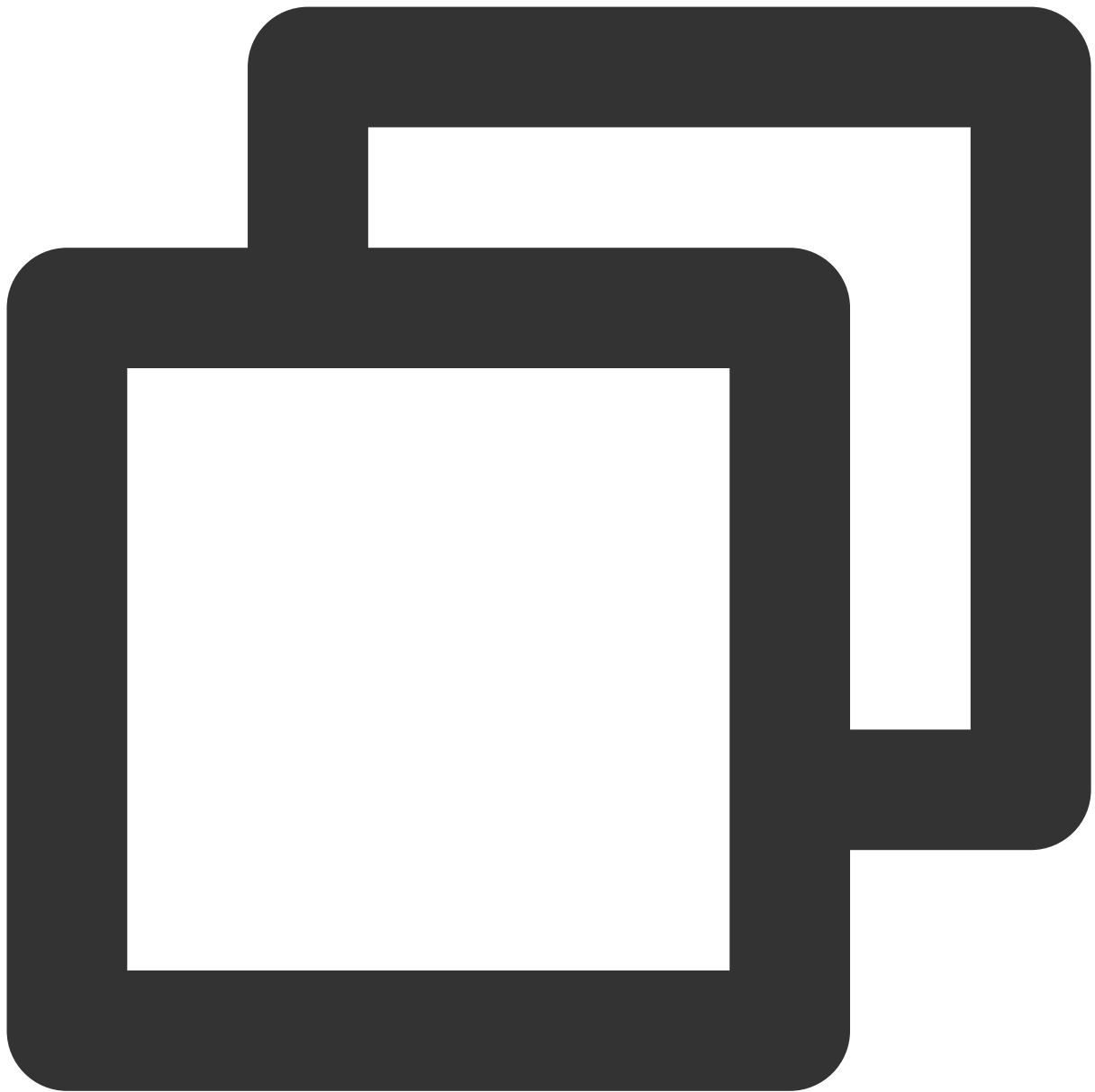
```
static void Main(string[] args)
{
    try
    {
        Credential cred = new Credential {
            SecretId = "SecretId",
            SecretKey = "SecretKey"
        };

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

        CkafkaClient client = new CkafkaClient(cred, "ap-beijing", clientProfile);
        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;
}
```