ALIBABA CLOUD

Alibaba Cloud

Message Queue for Apache Kafka

SDK reference

Document Version: 20201127

Alibaba Cloud

Legal disclaimer

Alibaba Cloud reminds you to carefully read and fully understand the terms and conditions of this legal disclaimer before you read or use this document. If you have read or used this document, it shall be deemed as your total acceptance of this legal disclaimer.

1. You shall download and obtain this document from the Alibaba Cloud website or other Alibaba Cloud-authorized channels, and use this document for your own legal business activities only. The content of this document is considered confidential information of Alibaba Cloud. You shall strictly abide by the confidentiality obligations. No part of this document shall be disclosed or provided to any third party for use without the prior written consent of Alibaba Cloud.

2. No part of this document shall be excerpted, translated, reproduced, transmitted, or disseminated by any organization, company or individual in any form or by any means without the prior written consent of Alibaba Cloud.

3. The content of this document may be changed because of product version upgrade, adjustment, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and an updated version of this document will be released through Alibaba Cloud-authorized channels from time to time. You should pay attention to the version changes of this document as they occur and download and obtain the most up-to-date version of this document from Alibaba Cloud-authorized channels.

4. This document serves only as a reference guide for your use of Alibaba Cloud products and services. Alibaba Cloud provides this document based on the "status quo", "being defective", and "existing functions" of its products and services. Alibaba Cloud makes every effort to provide relevant operational guidance based on existing technologies. However, Alibaba Cloud hereby makes a clear statement that it in no way guarantees the accuracy, integrity, applicability, and reliability of the content of this document, either explicitly or implicitly. Alibaba Cloud shall not take legal responsibility for any errors or lost profits incurred by any organization, company, or individual arising from download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, take responsibility for any indirect, consequential, punitive, contingent, special, or punitive damages, including lost profits arising from the use or trust in this document (even if Alibaba Cloud has been notified of the possibility of such a loss).

5. By law, all the contents in Alibaba Cloud documents, including but not limited to pictures, architecture design, page layout, and text description, are intellectual property of Alibaba Cloud and/or its affiliates. This intellectual property includes, but is not limited to, trademark rights, patent rights, copyrights, and trade secrets. No part of this document shall be used, modified, reproduced, publicly transmitted, changed, disseminated, distributed, or published without the prior written consent of Alibaba Cloud and/or its affiliates. The names owned by Alibaba Cloud shall not be used, published, or reproduced for marketing, advertising, promotion, or other purposes without the prior written consent of Alibaba Cloud. The names owned by Alibaba Cloud include, but are not limited to, "Alibaba Cloud", "Aliyun", "HiChina", and other brands of Alibaba Cloud and/or its affiliates, which appear separately or in combination, as well as the auxiliary signs and patterns of the preceding brands, or anything similar to the company names, trade names, trademarks, product or service names, domain names, patterns, logos, marks, signs, or special descriptions that third parties identify as Alibaba Cloud and/or its affiliates.

6. Please directly contact Alibaba Cloud for any errors of this document.

Document conventions

Style	Description	Example
Danger	A danger notice indicates a situation that will cause major system changes, faults, physical injuries, and other adverse results.	**Danger:** Resetting will result in the loss of user configuration data.
Warning	A warning notice indicates a situation that may cause major system changes, faults, physical injuries, and other adverse results.	**Warning:** Restarting will cause business interruption. About 10 minutes are required to restart an instance.
Notice	A caution notice indicates warning information, supplementary instructions, and other content that the user must understand.	**Notice:** If the weight is set to 0, the server no longer receives new requests.
Note	A note indicates supplemental instructions, best practices, tips, and other content.	**Note:** You can use Ctrl + A to select all files.
>	Closing angle brackets are used to indicate a multi-level menu cascade.	Click **Settings**> **Network**> **Set network type**.
Bold	Bold formatting is used for buttons , menus, page names, and other UI elements.	Click **OK**.
`Courier font`	Courier font is used for commands	Run the `cd /d C:/window` command to enter the Windows system folder.
Italic	Italic formatting is used for parameters and variables.	`bae log list --instanceid` *Instance_ID*
[] or [a|b]	This format is used for an optional value, where only one item can be selected.	`ipconfig [-all|-t]`
{} or {a|b}	This format is used for a required value, where only one item can be selected.	`switch {active|stand}`

Table of Contents

1.Overview

This topic introduces multi-language SDKs supported by Message Queue for Apache Kafka .

Java

- Demo
- Documentation

Python

- Demo
- Documentation

C++

- Demo
- Documentation

Go

- Demo
- Documentation

PHP

- Demo
- Documentation

Ruby

- Demo
- Documentation

Node.js

- Demo
- Documentation

2.SDK for Java

2.1. Overview

A Java client can connect to Message Queue for Apache Kafka through various endpoints and send and receive messages.

The following table lists the endpoints provided by Message Queue for Apache Kafka .

Item	Default endpoint	SASL endpoint
Network	VPC	VPC
Protocol	PLAINTEXT	SASL_PLAINTEXT
Port	9092	9094
SASL mechanism	N/A	• PLAIN: a simple username and password verification mechanism. Message Queue for Apache Kafka improves the PLAIN mechanism and allows you to dynamically add SASL users without restarting the instance. • SCRAM: a username and password verification mechanism, providing higher security than PLAIN. Message Queue for Apache Kafka uses SCRAM-SHA-256.
Demo	PLAINTEXT	• SASL_PLAINTEXT/PLAIN • SASL_PLAINTEXT/SCRAM
Topic	Use the default endpoint to send and subscribe to messages	• Use the PLAIN mechanism to send and subscribe to messages through an SASL endpoint • Use the SCRAM mechanism to send and subscribe to messages through an SASL endpoint

2.2. Use the default endpoint to send and subscribe to messages

This topic describes how to use SDK for Java to connect to the default endpoint of Message Queue for Apache Kafka to send and subscribe to messages in a virtual private cloud (VPC).

Prerequisites

- Install JDK 1.8 or later
- Install Maven 2.5 or later

Install Java dependencies

1. In the *pom.xml* And add the following dependencies.

```
<dependency> <groupId>org.apache.kafka <artifactId>kafka-clients</artifactId> <version>0.10.2.2< /version> </dependency> <dependency> <groupId>org.slf4j <artifactId>slf4j-log4j12</artifactId> <versi on>1.7.6</version> </dependency>
```

> **Note** We recommend that you keep the version of the provider and client consistent, that is, keep the client library version consistent with Message Queue for Apache Kafka The major version must be the same for all instances. You can go to the Message Queue for Apache Kafka The console's **Instance Details page** Page acquisition Message Queue for Apache Kafka The major version of the instance.

Prepare configurations

1. Create a Log4j configuration file *log4j.properties* .

```
# Licensed to the Apache Software Foundation (ASF) under one or more # contributor license agree ments. See the NOTICE file distributed with # this work for additional information regarding copyright ownership. # The ASF licenses this file to You under the Apache License, Version 2.0 # (the "License"); yo u may not use this file except in compliance with # the License. You may obtain a copy of the License at # # http://www.apache.org/licenses/LICENSE-2.0 # # Unless required by applicable law or agreed to in w riting, software # distributed under the License is distributed on an "AS IS" BASIS, # WITHOUT WARRAN TIES OR CONDITIONS OF ANY KIND, either express or implied. # See the License for the specific language governing permissions and # limitations under the License. log4j.rootLogger=INFO, STDOUT log4j.appe nder.STDOUT=org.apache.log4j.ConsoleAppender log4j.appender.STDOUT.layout=org.apache.log4j.Pa tternLayout log4j.appender.STDOUT.layout.ConversionPattern=[%d] %p %m (%c)%n
```

2. Create a Kafka configuration file.

```
## The endpoint. Set it to the default endpoint displayed on the Instance Details page in the console.
bootstrap.servers=xxxxxxxxxxxxxxxxxxxxx
## The topic, which is created in the Message Queue for Apache Kafka console.
topic=alikafka-topic-demo
## The consumer group, which is created in the Message Queue for Apache Kafka console.
group.id=CID-consumer-group-demo
```

3. Create a profile loader *JavaKafkaConfigurer.java*.

```
public class JavaKafkaConfigurer {
  private static Properties properties;
  public synchronized static Properties getKafkaProperties() {
    if (null ! = properties) {
      return properties;
    }
    // Obtain the content of the configuration file kafka.properties.
    Properties kafkaProperties = new Properties();
    try {
      kafkaProperties.load(KafkaProducerDemo.class.getClassLoader().getResourceAsStream("kafka.pr
operties"));
    } catch (Exception e) {
      // Exit the program if the file loading failed.
      e.printStackTrace();
    }
    properties = kafkaProperties;
    return kafkaProperties;
  }
}
```

Send messages

1. Create a message sender *KafkaProducerDemo.java*.

```
import java.util.ArrayList;
import java.util.List;
import java.util.Properties;
import java.util.concurrent.Future;
import java.util.concurrent.TimeUnit;
import org.apache.kafka.clients.CommonClientConfigs;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.clients.producer.RecordMetadata;
public class KafkaProducerDemo {
  public static void main(String args[]) {
    // Load kafka.properties.
    Properties kafkaProperties =  JavaKafkaConfigurer.getKafkaProperties();
    Properties props = new Properties();
    // Specify the endpoint. We recommend that you obtain the endpoint of the corresponding topic in
the console.
```

```
    props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, kafkaProperties.getProperty("bootstrap.servers"));
    // Specify the method for serializing messages of Message Queue for Apache Kafka.
    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");
    // Specify the maximum waiting time for a request.
    props.put(ProducerConfig.MAX_BLOCK_MS_CONFIG, 30 * 1000);
    // Set the number of retries in the client.
    props.put(ProducerConfig.RETRIES_CONFIG, 5);
    // Set the interval of retries in the client.
    props.put(ProducerConfig.RECONNECT_BACKOFF_MS_CONFIG, 3000);
    // Construct a producer object, which is thread-safe. One producer object can serve a process.
    // To improve performance, you can construct a few more objects. We recommend that you construct no more than five objects.
    KafkaProducer<String, String> producer = new KafkaProducer<String, String>(props);
    // Construct a Message Queue for Apache Kafka message.
    String topic = kafkaProperties.getProperty("topic"); // The topic of the message. Enter the topic you have created in the console.
    String value = "this is the message's value"; //Specify the content of the message.
    try {
      // You can speed up the process by obtaining Future objects in batches. Note that the batch size must not be too large.
      List<Future<RecordMetadata>> futures = new ArrayList<Future<RecordMetadata>>(128);
      for (int i =0; i < 100; i++) {
        // Send the message and obtain a Future object.
        ProducerRecord<String, String> kafkaMessage =  new ProducerRecord<String, String>(topic, value + ": " + i);
        Future<RecordMetadata> metadataFuture = producer.send(kafkaMessage);
        futures.add(metadataFuture);
      }
      producer.flush();
      for (Future<RecordMetadata> future: futures) {
        // Synchronize the obtained Future object.
        try {
          RecordMetadata recordMetadata = future.get();
          System.out.println("Produce ok:" + recordMetadata.toString());
        } catch (Throwable t) {
          t.printStackTrace();
        }
```

```
            }
        }
    } catch (Exception e) {
        // Catch the error for handling if the client fails to send data after a retry.
        System.out.println("error occurred");
        e.printStackTrace();
    }
  }
}
```

2. Compile and run the message sender *KafkaProducerDemo.java*.

Subscribe to messages

1. Use either of the following methods to subscribe to messages:
   - A single consumer subscribes to messages
     a. Create a single-consumer subscription program *KafkaConsumerDemo.java*.

```
import java.util.ArrayList;
import java.util.List;
import java.util.Properties;
import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.apache.kafka.clients.producer.ProducerConfig;
public class KafkaConsumerDemo {
  public static void main(String args[]) {
    // Load kafka.properties.
    Properties kafkaProperties =  JavaKafkaConfigurer.getKafkaProperties();
    Properties props = new Properties();
    // Specify the endpoint. We recommend that you obtain the endpoint of the corresponding t
opic in the console.
    props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, kafkaProperties.getProperty("b
ootstrap.servers"));
    // Specify the session timeout period. If the consumer does not return a heartbeat before the
session times out, the broker determines that the consumer is not alive. Then the broker remov
es the consumer from the consumer group and triggers rebalancing. The default value is 30s.
    props.put(ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG, 30000);
    // The maximum number of messages that can be polled at a time.
    // Do not set the number to an excessively large value. If excessive messages are polled but fa
il to be completely consumed before the next poll starts, load balancing is triggered, which caus
es freezing.
```

```
    props.put(ConsumerConfig.MAX_POLL_RECORDS_CONFIG, 30);
    // Specify the method for deserializing messages.
    props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, "org.apache.kafka.common
.serialization.StringDeserializer");
    props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, "org.apache.kafka.comm
on.serialization.StringDeserializer");
    // The consumer group of the current consumer instance. Enter the consumer group you hav
e created in the console.
    // The consumer instances that belong to the same consumer group. These instances consu
me messages in load balancing mode.
    props.put(ConsumerConfig.GROUP_ID_CONFIG, kafkaProperties.getProperty("group.id"));
    // Construct a message object to generate a consumer instance.
    KafkaConsumer<String, String> consumer = new org.apache.kafka.clients.consumer.KafkaC
onsumer<String, String>(props);
    // Set the topics to which the consumer group subscribes. One consumer group can subscrib
e to multiple topics.
    // We recommend that you set the topic to the same value for the consumer instances with t
he same GROUP_ID_CONFIG value.
    List<String> subscribedTopics =  new ArrayList<String>();
    // If you want to subscribe to multiple topics, add them here.
    // You must create the topics in the console before you add them.
    String topicStr = kafkaProperties.getProperty("topic");
    String[] topics = topicStr.split(",");
    for (String topic: topics) {
      subscribedTopics.add(topic.trim());
    }
    consumer.subscribe(subscribedTopics);
    // Cyclically consume messages.
    while (true){
      try {
        ConsumerRecords<String, String> records = consumer.poll(1000);
        // The messages must be completely consumed before the next polling cycle starts. The t
otal duration cannot exceed the timeout interval specified by SESSION_TIMEOUT_MS_CONFIG.
        // We recommend that you create a separate thread pool to consume messages and asyn
chronously return the results.
        for (ConsumerRecord<String, String> record : records) {
          System.out.println(String.format("Consume partition:%d offset:%d", record.partition(
), record.offset()));
        }
      } catch (Exception e) {
        try {
```

```
            Thread.sleep(1000);
          } catch (Throwable ignore) {
          }
          e.printStackTrace();
        }
      }
    }
}
```

b. Compile and run *KafkaConsumerDemo.java*.

- Multiple consumers subscribe to messages

a. Create a multi-consumer subscription program *KafkaMultiConsumerDemo.java*.

```
import java.util.ArrayList;
import java.util.List;
import java.util.Properties;
import java.util.concurrent.atomic.AtomicBoolean;
import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.common.errors.WakeupException;
/**
 * This tutorial demonstrates how to enable multiple consumers to simultaneously consume top
ics in one process.
 * Ensure that the number of global consumers does not exceed the total number of partitions of
the topics that the consumers subscribed to.
 */
public class KafkaMultiConsumerDemo {
  public static void main(String args[]) throws InterruptedException {
    // Load kafka.properties.
    Properties kafkaProperties = JavaKafkaConfigurer.getKafkaProperties();
    Properties props = new Properties();
    // Specify the endpoint. We recommend that you obtain the endpoint of the corresponding t
opic in the console.
    props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, kafkaProperties.getProperty("b
ootstrap.servers"));
    // Specify the session timeout period. If the consumer does not return a heartbeat before the
session times out, the broker determines that the consumer is not alive. Then the broker remov
es the consumer from the consumer group and triggers rebalancing. The default value is 30s.
    props.put(ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG, 30000);
```

```
        props.put(ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG, 30000);
        // The maximum number of messages that can be polled at a time.
        // Do not set the number to an excessively large value. If excessive messages are polled but fa
il to be completely consumed before the next poll starts, load balancing is triggered, which caus
es freezing.
        props.put(ConsumerConfig.MAX_POLL_RECORDS_CONFIG, 30);
        // Specify the method for deserializing messages.
        props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, "org.apache.kafka.common
.serialization.StringDeserializer");
        props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, "org.apache.kafka.comm
on.serialization.StringDeserializer");
        // The consumer group of the current consumer instance. Enter the consumer group you hav
e created in the console.
        // The consumer instances that belong to the same consumer group. These instances consu
me messages in load balancing mode.
        props.put(ConsumerConfig.GROUP_ID_CONFIG, kafkaProperties.getProperty("group.id"));
        int consumerNum = 2;
        Thread[] consumerThreads = new Thread[consumerNum];
        for (int i = 0; i < consumerNum; i++) {
            KafkaConsumer<String, String> consumer = new KafkaConsumer<String, String>(props);
            List<String> subscribedTopics = new ArrayList<String>();
            subscribedTopics.add(kafkaProperties.getProperty("topic"));
            consumer.subscribe(subscribedTopics);
            KafkaConsumerRunner kafkaConsumerRunner = new KafkaConsumerRunner(consumer);
            consumerThreads[i] = new Thread(kafkaConsumerRunner);
        }
        for (int i = 0; i < consumerNum; i++) {
            consumerThreads[i].start();
        }
        for (int i = 0; i < consumerNum; i++) {
            consumerThreads[i].join();
        }
    }
    static class KafkaConsumerRunner implements Runnable {
        private final AtomicBoolean closed = new AtomicBoolean(false);
        private final KafkaConsumer consumer;
        KafkaConsumerRunner(KafkaConsumer consumer) {
            this.consumer = consumer;
        }
        @Override
        public void run() {
```

```
        try {
          while (! closed.get()) {
            try {
              ConsumerRecords<String, String> records = consumer.poll(1000);
              // The messages must be completely consumed before the next polling cycle starts. T
he total duration cannot exceed the timeout interval specified by SESSION_TIMEOUT_MS_CONF
IG.
              for (ConsumerRecord<String, String> record : records) {
                System.out.println(String.format("Thread:%s Consume partition:%d offset:%d", T
hread.currentThread().getName(), record.partition(), record.offset()));
              }
            } catch (Exception e) {
              try {
                Thread.sleep(1000);
              } catch (Throwable ignore) {
              }
              e.printStackTrace();
            }
          }
        } catch (WakeupException e) {
          // Ignore exceptions if it is disabled.
          if (! closed.get()) {
            throw e;
          }
        } finally {
          consumer.close();
        }
      }
      // The shutdown hook that can be called by another thread.
      public void shutdown() {
        closed.set(true);
        consumer.wakeup();
      }
    }
}
```

b. Compile and run *KafkaMultiConsumerDemo.java*.

2.3. Use the PLAIN mechanism to send and subscribe to messages through an SASL endpoint

This topic describes how a Java client connects to Message Queue for Apache Kafka through an SASL endpoint in a virtual private cloud (VPC) and uses the PLAIN mechanism to send and subscribe to messages.

Prerequisites

- Install JDK 1.8 or later
- Install Maven 2.5 or later
- Authorize SASL users

Install Java dependencies

1. In the *pom.xml* And add the following dependencies.

```
<dependency> <groupId>org.apache.kafka <artifactId>kafka-clients</artifactId> <version>0.10.2.2</version> </dependency> <dependency> <groupId>org.slf4j <artifactId>slf4j-log4j12</artifactId> <version>1.7.6</version> </dependency>
```

> **Note** We recommend that you keep the version of the provider and client consistent, that is, keep the client library version consistent with Message Queue for Apache Kafka The major version must be the same for all instances. You can go to the Message Queue for Apache Kafka The console's **Instance Details page** Page acquisition Message Queue for Apache Kafka The major version of the instance.

Prepare configurations

1. Create a Log4j configuration file *log4j.properties* .

```
# Licensed to the Apache Software Foundation (ASF) under one or more # contributor license agreements. See the NOTICE file distributed with # this work for additional information regarding copyright ownership. # The ASF licenses this file to You under the Apache License, Version 2.0 # (the "License"); you may not use this file except in compliance with # the License. You may obtain a copy of the License at # # http://www.apache.org/licenses/LICENSE-2.0 # # Unless required by applicable law or agreed to in writing, software # distributed under the License is distributed on an "AS IS" BASIS, # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. # See the License for the specific language governing permissions and # limitations under the License. log4j.rootLogger=INFO, STDOUT log4j.appender.STDOUT=org.apache.log4j.ConsoleAppender log4j.appender.STDOUT.layout=org.apache.log4j.PatternLayout log4j.appender.STDOUT.layout.ConversionPattern=[%d] %p %m (%c)%n
```

2. Create a JAAS configuration file *kafka_client_jaas_plain.conf*.

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

3. Create a Kafka configuration file *kafka.properties*.

```
## The SASL endpoint, which is obtained from the Message Queue for Apache Kafka console.
bootstrap.servers=xxxx
## The topic, which is created in the Message Queue for Apache Kafka console.
topic=xxxx
## The consumer group, which is created in the Message Queue for Apache Kafka console.
group.id=xxxx
## The JAAS configuration file.
java.security.auth.login.config.plain=/xxxx/kafka_client_jaas_plain.conf
```

4. Create a profile loader *JavaKafkaConfigurer.java*.

```
import java.util.Properties;
public class JavaKafkaConfigurer {
  private static Properties properties;
  public static void configureSaslPlain() {
    // If you have set the path by using methods such as -D, do not set it again.
    if (null == System.getProperty("java.security.auth.login.config")) {
      // Replace XXX with your own path.
      // Ensure that the path is readable by the file system. Do not compress the file into a JAR package.
      System.setProperty("java.security.auth.login.config", getKafkaProperties().getProperty("java.sec
urity.auth.login.config.plain"));
    }
  }
  public synchronized static Properties getKafkaProperties() {
    if (null ! = properties) {
      return properties;
    }
    // Obtain the content of the configuration file kafka.properties.
    Properties kafkaProperties = new Properties();
    try {
      kafkaProperties.load(KafkaProducerDemo.class.getClassLoader().getResourceAsStream("kafka.p
roperties"));
    } catch (Exception e) {
      // Exit the program if the file loading failed.
      e.printStackTrace();
    }
    properties = kafkaProperties;
    return kafkaProperties;
  }
}
```

Send messages

1. Create a message sender *KafkaProducerDemo.java*.

```
import java.util.ArrayList;
import java.util.List;
import java.util.Properties;
import java.util.concurrent.Future;
import org.apache.kafka.clients.CommonClientConfigs;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.clients.producer.ProducerRecord;
```

```
import org.apache.kafka.clients.producer.RecordMetadata;
import org.apache.kafka.common.config.SaslConfigs;
import org.apache.kafka.common.config.SslConfigs;
public class KafkaProducerDemo {
  public static void main(String args[]) {
    // Set the path of the JAAS configuration file.
    JavaKafkaConfigurer.configureSaslPlain();
    // Load kafka.properties.
    Properties kafkaProperties =  JavaKafkaConfigurer.getKafkaProperties();
    Properties props = new Properties();
    // Specify the endpoint. We recommend that you obtain the endpoint of the corresponding topic in the console.
    props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, kafkaProperties.getProperty("bootstrap.servers"));
    // Specify the access protocol.
    props.put(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG, "SASL_PLAINTEXT");
    // Specify the PLAIN mechanism.
    props.put(SaslConfigs.SASL_MECHANISM, "PLAIN");
    // Specify the method for serializing messages of Message Queue for Apache Kafka.
    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");
    // Specify the maximum waiting time for a request.
    props.put(ProducerConfig.MAX_BLOCK_MS_CONFIG, 30 * 1000);
    // Set the number of retries in the client.
    props.put(ProducerConfig.RETRIES_CONFIG, 5);
    // Set the interval of retries in the client.
    props.put(ProducerConfig.RECONNECT_BACKOFF_MS_CONFIG, 3000);
    // Construct a producer object, which is thread-safe. One producer object can serve a process.
    // To improve performance, you can construct a few more objects. We recommend that you construct no more than five objects.
    KafkaProducer<String, String> producer = new KafkaProducer<String, String>(props);
    // Construct a Message Queue for Apache Kafka message.
    String topic = kafkaProperties.getProperty("topic"); // The topic of the message. Enter the topic you have created in the console.
    String value = "this is the message's value"; //Specify the content of the message.
    try {
      // You can speed up the process by obtaining Future objects in batches. Note that the batch size must not be too large.
      List<Future<RecordMetadata>> futures = new ArrayList<Future<RecordMetadata>>(128);
```

```
        List<Future<RecordMetadata>> futures = new ArrayList<Future<RecordMetadata>>(128);
        for (int i =0; i < 100; i++) {
          // Send the message and obtain a Future object.
          ProducerRecord<String, String> kafkaMessage =  new ProducerRecord<String, String>(topic, val
ue + ": " + i);
          Future<RecordMetadata> metadataFuture = producer.send(kafkaMessage);
          futures.add(metadataFuture);
        }
        producer.flush();
        for (Future<RecordMetadata> future: futures) {
          // Synchronize the obtained Future object.
          try {
            RecordMetadata recordMetadata = future.get();
            System.out.println("Produce ok:" + recordMetadata.toString());
          } catch (Throwable t) {
            t.printStackTrace();
          }
        }
      } catch (Exception e) {
        // Catch the error for handling if the client fails to send data after a retry.
        System.out.println("error occurred");
        e.printStackTrace();
      }
    }
}
```

2. Compile and run *KafkaProducerDemo.java*.

Subscribe to messages

1. Use either of the following methods to subscribe to messages:
   - A single consumer subscribes to messages
     a. Create a single-consumer subscription program *KafkaConsumerDemo.java*.

```
import java.util.ArrayList;
import java.util.List;
import java.util.Properties;
import org.apache.kafka.clients.CommonClientConfigs;
import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.common.config.SaslConfigs;
```

```
import org.apache.kafka.common.config.SaslConfigs;
public class KafkaConsumerDemo {
  public static void main(String args[]) {
    // Set the path of the JAAS configuration file.
    JavaKafkaConfigurer.configureSaslPlain();
    // Load kafka.properties.
    Properties kafkaProperties =  JavaKafkaConfigurer.getKafkaProperties();
    Properties props = new Properties();
    // Specify the endpoint. We recommend that you obtain the endpoint of the corresponding t
opic in the console.
    props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, kafkaProperties.getProperty("b
ootstrap.servers"));
    // Specify the access protocol.
    props.put(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG, "SASL_PLAINTEXT");
    // Specify the PLAIN mechanism.
    props.put(SaslConfigs.SASL_MECHANISM, "PLAIN");
    // Specify the session timeout period. If the consumer does not return a heartbeat before the
session times out, the broker determines that the consumer is not alive. Then the broker remov
es the consumer from the consumer group and triggers rebalancing. The default value is 30s.
    props.put(ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG, 30000);
    // The maximum number of messages that can be polled at a time.
    // Do not set the number to an excessively large value. If excessive messages are polled but fa
il to be completely consumed before the next poll starts, load balancing is triggered, which caus
es freezing.
    props.put(ConsumerConfig.MAX_POLL_RECORDS_CONFIG, 30);
    // Specify the method for deserializing messages.
    props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, "org.apache.kafka.common
.serialization.StringDeserializer");
    props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, "org.apache.kafka.comm
on.serialization.StringDeserializer");
    // The consumer group of the current consumer instance. Enter the consumer group you hav
e created in the console.
    // The consumer instances that belong to the same consumer group. These instances consu
me messages in load balancing mode.
    props.put(ConsumerConfig.GROUP_ID_CONFIG, kafkaProperties.getProperty("group.id"));
    // Construct a message object to generate a consumer instance.
    KafkaConsumer<String, String> consumer = new org.apache.kafka.clients.consumer.KafkaC
onsumer<String, String>(props);
    // Set the topics to which the consumer group subscribes. One consumer group can subscrib
e to multiple topics.
    // We recommend that you set the topic to the same value for the consumer instances with t
```

```
he same GROUP_ID_CONFIG value.
    List<String> subscribedTopics =  new ArrayList<String>();
    // If you want to subscribe to multiple topics, add them here.
    // You must create the topics in the console before you add them.
    String topicStr = kafkaProperties.getProperty("topic");
    String[] topics = topicStr.split(",");
    for (String topic: topics) {
      subscribedTopics.add(topic.trim());
    }
    consumer.subscribe(subscribedTopics);
    // Cyclically consume messages.
    while (true){
      try {
        ConsumerRecords<String, String> records = consumer.poll(1000);
        // The messages must be completely consumed before the next polling cycle starts. The t
otal duration cannot exceed the timeout interval specified by SESSION_TIMEOUT_MS_CONFIG.
        // We recommend that you create a separate thread pool to consume messages and asyn
chronously return the results.
        for (ConsumerRecord<String, String> record : records) {
          System.out.println(String.format("Consume partition:%d offset:%d", record.partition(
), record.offset()));
        }
      } catch (Exception e) {
        try {
          Thread.sleep(1000);
        } catch (Throwable ignore) {
        }
        e.printStackTrace();
      }
    }
  }
}
```

b. Compile and run *KafkaConsumerDemo.java*.

- Multiple consumers subscribe to messages

a. Create a multi-consumer subscription program *KafkaMultiConsumerDemo.java*.

```
import java.util.ArrayList;
import java.util.List;
import java.util.Properties;
import java.util.concurrent.atomic.AtomicBoolean;
import org.apache.kafka.clients.CommonClientConfigs;
```

```
import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.common.config.SaslConfigs;
import org.apache.kafka.common.errors.WakeupException;
/**
 * This tutorial demonstrates how to enable multiple consumers to simultaneously consume top
ics in one process.
 * Ensure that the number of global consumers does not exceed the total number of partitions of
the topics that the consumers subscribed to.
 */
public class KafkaMultiConsumerDemo {
  public static void main(String args[]) throws InterruptedException {
    // Set the path of the JAAS configuration file.
    JavaKafkaConfigurer.configureSaslPlain();
    // Load kafka.properties.
    Properties kafkaProperties = JavaKafkaConfigurer.getKafkaProperties();
    Properties props = new Properties();
    // Specify the endpoint. We recommend that you obtain the endpoint of the corresponding t
opic in the console.
    props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, kafkaProperties.getProperty("b
ootstrap.servers"));
    // Specify the access protocol.
    props.put(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG, "SASL_PLAINTEXT");
    // Specify the PLAIN mechanism.
    props.put(SaslConfigs.SASL_MECHANISM, "PLAIN");
    // Specify the session timeout period. If the consumer does not return a heartbeat before the
session times out, the broker determines that the consumer is not alive. Then the broker remov
es the consumer from the consumer group and triggers rebalancing. The default value is 30s.
    props.put(ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG, 30000);
    // The maximum number of messages that can be polled at a time.
    // Do not set the number to an excessively large value. If excessive messages are polled but fa
il to be completely consumed before the next poll starts, load balancing is triggered, which caus
es freezing.
    props.put(ConsumerConfig.MAX_POLL_RECORDS_CONFIG, 30);
    // Specify the method for deserializing messages.
    props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, "org.apache.kafka.common
.serialization.StringDeserializer");
```

```
    props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, "org.apache.kafka.comm
on.serialization.StringDeserializer");
    // The consumer group of the current consumer instance. Enter the consumer group you hav
e created in the console.
    // The consumer instances that belong to the same consumer group. These instances consu
me messages in load balancing mode.
    props.put(ConsumerConfig.GROUP_ID_CONFIG, kafkaProperties.getProperty("group.id"));
    int consumerNum = 2;
    Thread[] consumerThreads = new Thread[consumerNum];
    for (int i = 0; i < consumerNum; i++) {
      KafkaConsumer<String, String> consumer = new KafkaConsumer<String, String>(props);
      List<String> subscribedTopics = new ArrayList<String>();
      subscribedTopics.add(kafkaProperties.getProperty("topic"));
      consumer.subscribe(subscribedTopics);
      KafkaConsumerRunner kafkaConsumerRunner = new KafkaConsumerRunner(consumer);
      consumerThreads[i] = new Thread(kafkaConsumerRunner);
    }
    for (int i = 0; i < consumerNum; i++) {
      consumerThreads[i].start();
    }
    for (int i = 0; i < consumerNum; i++) {
      consumerThreads[i].join();
    }
  }
  static class KafkaConsumerRunner implements Runnable {
    private final AtomicBoolean closed = new AtomicBoolean(false);
    private final KafkaConsumer consumer;
    KafkaConsumerRunner(KafkaConsumer consumer) {
      this.consumer = consumer;
    }
    @Override
    public void run() {
      try {
        while (! closed.get()) {
          try {
            ConsumerRecords<String, String> records = consumer.poll(1000);
            // The messages must be completely consumed before the next polling cycle starts. T
he total duration cannot exceed the timeout interval specified by SESSION_TIMEOUT_MS_CONF
IG.
            for (ConsumerRecord<String, String> record : records) {
              System.out.println(String.format("Thread:%s Consume partition:%d offset:%d", T
```

```
hread.currentThread().getName(), record.partition(), record.offset()));
                }
              } catch (Exception e) {
                try {
                  Thread.sleep(1000);
                } catch (Throwable ignore) {
                }
                e.printStackTrace();
              }
            }
          } catch (WakeupException e) {
            // Ignore exceptions if it is disabled.
            if (! closed.get()) {
              throw e;
            }
          } finally {
            consumer.close();
          }
        }
        // The shutdown hook that can be called by another thread.
        public void shutdown() {
          closed.set(true);
          consumer.wakeup();
        }
      }
    }
```

b. Compile and run *KafkaMultiConsumerDemo.java*.

2.4. Use the SCRAM mechanism to send and subscribe to messages through an SASL endpoint

This topic describes how a Java client connects to Message Queue for Apache Kafka through an SASL endpoint in a virtual private cloud (VPC) and uses the SCRAM mechanism to send and subscribe to messages.

Prerequisites

- Install JDK 1.8 or later
- Install Maven 2.5 or later
- Authorize SASL users

Install Java dependencies

1. In the *pom.xml* And add the following dependencies.

```
<dependency> <groupId>org.apache.kafka <artifactId>kafka-clients</artifactId> <version>0.10.2.2< /version> </dependency> <dependency> <groupId>org.slf4j <artifactId>slf4j-log4j12</artifactId> <versi on>1.7.6</version> </dependency>
```

> **Note** We recommend that you keep the version of the provider and client consistent, that is, keep the client library version consistent with Message Queue for Apache Kafka The major version must be the same for all instances. You can go to the Message Queue for Apache Kafka The console's **Instance Details page** Page acquisition Message Queue for Apache Kafka The major version of the instance.

Prepare configurations

1. Create a Log4j configuration file *log4j.properties* .

```
# Licensed to the Apache Software Foundation (ASF) under one or more # contributor license agree ments. See the NOTICE file distributed with # this work for additional information regarding copyright ownership. # The ASF licenses this file to You under the Apache License, Version 2.0 # (the "License"); yo u may not use this file except in compliance with # the License. You may obtain a copy of the License at # # http://www.apache.org/licenses/LICENSE-2.0 # # Unless required by applicable law or agreed to in w riting, software # distributed under the License is distributed on an "AS IS" BASIS, # WITHOUT WARRAN TIES OR CONDITIONS OF ANY KIND, either express or implied. # See the License for the specific language governing permissions and # limitations under the License. log4j.rootLogger=INFO, STDOUT log4j.appe nder.STDOUT=org.apache.log4j.ConsoleAppender log4j.appender.STDOUT.layout=org.apache.log4j.Pa tternLayout log4j.appender.STDOUT.layout.ConversionPattern=[%d] %p %m (%c)%n
```

2. Create a JAAS configuration file *kafka_client_jaas_scram.conf*.

```
KafkaClient {
 org.apache.kafka.common.security.scram.ScramLoginModule required
 username="xxxx"
 password="xxxx";
};
```

3. Create a Kafka configuration file *kafka.properties*.

```
## The SASL endpoint, which is obtained from the Message Queue for Apache Kafka console.
bootstrap.servers=xxxx
## The topic, which is created in the Message Queue for Apache Kafka console.
topic=xxxx
## The consumer group, which is created in the Message Queue for Apache Kafka console.
group.id=xxxx
## The JAAS configuration file.
java.security.auth.login.config.scram=/xxxx/kafka_client_jaas_scram.conf
```

4. Create a profile loader *JavaKafkaConfigurer.java*.

```
import java.util.Properties;
public class JavaKafkaConfigurer {
  private static Properties properties;
  public static void configureSaslScram() {
    // If you have set the path by using methods such as -D, do not set it again.
    if (null == System.getProperty("java.security.auth.login.config")) {
      // Replace XXX with your own path.
      // Ensure that the path is readable by the file system. Do not compress the file into a JAR package.
      System.setProperty("java.security.auth.login.config", getKafkaProperties().getProperty("java.sec
urity.auth.login.config.scram"));
    }
  }
  public synchronized static Properties getKafkaProperties() {
    if (null ! = properties) {
      return properties;
    }
    // Obtain the content of the configuration file kafka.properties.
    Properties kafkaProperties = new Properties();
    try {
      kafkaProperties.load(KafkaProducerDemo.class.getClassLoader().getResourceAsStream("kafka.p
roperties"));
    } catch (Exception e) {
      // Exit the program if the file loading failed.
      e.printStackTrace();
    }
    properties = kafkaProperties;
    return kafkaProperties;
  }
}
```

Send messages

1. Create a message sender *KafkaProducerDemo.java*.

```
import java.util.ArrayList;
import java.util.List;
import java.util.Properties;
import java.util.concurrent.Future;
import org.apache.kafka.clients.CommonClientConfigs;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.clients.producer.RecordMetadata;
import org.apache.kafka.common.config.SaslConfigs;
import org.apache.kafka.common.config.SslConfigs;
public class KafkaProducerDemo {
  public static void main(String args[]) {
    // Set the path of the JAAS configuration file.
    JavaKafkaConfigurer.configureSaslScram();
    // Load kafka.properties.
    Properties kafkaProperties =  JavaKafkaConfigurer.getKafkaProperties();
    Properties props = new Properties();
    // Specify the endpoint. We recommend that you obtain the endpoint of the corresponding topic in the console.
    props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, kafkaProperties.getProperty("bootstrap.servers"));
    // Specify the access protocol.
    props.put(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG, "SASL_PLAINTEXT");
    // Specify the SCRAM mechanism.
    props.put(SaslConfigs.SASL_MECHANISM, "SCRAM-SHA-256");
    // Specify the method for serializing messages of Message Queue for Apache Kafka.
    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");
    // Specify the maximum waiting time for a request.
    props.put(ProducerConfig.MAX_BLOCK_MS_CONFIG, 30 * 1000);
    // Set the number of retries in the client.
    props.put(ProducerConfig.RETRIES_CONFIG, 5);
    // Set the interval of retries in the client.
    props.put(ProducerConfig.RECONNECT_BACKOFF_MS_CONFIG, 3000);
    // Construct a producer object, which is thread-safe. One producer object can serve a process.
```

```
    // To improve performance, you can construct a few more objects. We recommend that you construct no more than five objects.
    KafkaProducer<String, String> producer = new KafkaProducer<String, String>(props);
    // Construct a Message Queue for Apache Kafka message.
    String topic = kafkaProperties.getProperty("topic"); // The topic of the message. Enter the topic you have created in the console.
    String value = "this is the message's value"; //Specify the content of the message.
    try {
      // You can speed up the process by obtaining Future objects in batches. Note that the batch size must not be too large.
      List<Future<RecordMetadata>> futures = new ArrayList<Future<RecordMetadata>>(128);
      for (int i =0; i < 100; i++) {
        // Send the message and obtain a Future object.
        ProducerRecord<String, String> kafkaMessage =  new ProducerRecord<String, String>(topic, value + ": " + i);
        Future<RecordMetadata> metadataFuture = producer.send(kafkaMessage);
        futures.add(metadataFuture);
      }
      producer.flush();
      for (Future<RecordMetadata> future: futures) {
        // Synchronize the obtained Future object.
        try {
          RecordMetadata recordMetadata = future.get();
          System.out.println("Produce ok:" + recordMetadata.toString());
        } catch (Throwable t) {
          t.printStackTrace();
        }
      }
    } catch (Exception e) {
      // Catch the error for handling if the client fails to send data after a retry.
      System.out.println("error occurred");
      e.printStackTrace();
    }
  }
}
```

2. Compile and run *KafkaProducerDemo.java*.

Subscribe to messages

1. Use either of the following methods to subscribe to messages:
   - A single consumer subscribes to messages

a. Create a single-consumer subscription program *KafkaConsumerDemo.java*.

```
import java.util.ArrayList;
import java.util.List;
import java.util.Properties;
import org.apache.kafka.clients.CommonClientConfigs;
import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.common.config.SaslConfigs;
public class KafkaConsumerDemo {
  public static void main(String args[]) {
    // Set the path of the JAAS configuration file.
    JavaKafkaConfigurer.configureSaslScram();
    // Load kafka.properties.
    Properties kafkaProperties =  JavaKafkaConfigurer.getKafkaProperties();
    Properties props = new Properties();
    // Specify the endpoint. We recommend that you obtain the endpoint of the corresponding t
opic in the console.
    props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, kafkaProperties.getProperty("b
ootstrap.servers"));
    // Specify the access protocol.
    props.put(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG, "SASL_PLAINTEXT");
    // Specify the SCRAM mechanism.
    props.put(SaslConfigs.SASL_MECHANISM, "SCRAM-SHA-256");
    // Specify the session timeout period. If the consumer does not return a heartbeat before the
session times out, the broker determines that the consumer is not alive. Then the broker remov
es the consumer from the consumer group and triggers rebalancing. The default value is 30s.
    props.put(ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG, 30000);
    // The maximum number of messages that can be polled at a time.
    // Do not set the number to an excessively large value. If excessive messages are polled but fa
il to be completely consumed before the next poll starts, load balancing is triggered, which caus
es freezing.
    props.put(ConsumerConfig.MAX_POLL_RECORDS_CONFIG, 30);
    // Specify the method for deserializing messages.
    props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, "org.apache.kafka.common
.serialization.StringDeserializer");
    props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, "org.apache.kafka.comm
on.serialization.StringDeserializer");
```

```
    // The consumer group of the current consumer instance. Enter the consumer group you hav
e created in the console.
    // The consumer instances that belong to the same consumer group. These instances consu
me messages in load balancing mode.
    props.put(ConsumerConfig.GROUP_ID_CONFIG, kafkaProperties.getProperty("group.id"));
    // Construct a message object to generate a consumer instance.
    KafkaConsumer<String, String> consumer = new org.apache.kafka.clients.consumer.KafkaC
onsumer<String, String>(props);
    // Set the topics to which the consumer group subscribes. One consumer group can subscrib
e to multiple topics.
    // We recommend that you set the topic to the same value for the consumer instances with t
he same GROUP_ID_CONFIG value.
    List<String> subscribedTopics =  new ArrayList<String>();
    // If you want to subscribe to multiple topics, add them here.
    // You must create the topics in the console before you add them.
    String topicStr = kafkaProperties.getProperty("topic");
    String[] topics = topicStr.split(",");
    for (String topic: topics) {
      subscribedTopics.add(topic.trim());
    }
    consumer.subscribe(subscribedTopics);
    // Cyclically consume messages.
    while (true){
      try {
        ConsumerRecords<String, String> records = consumer.poll(1000);
        // The messages must be completely consumed before the next polling cycle starts. The t
otal duration cannot exceed the timeout interval specified by SESSION_TIMEOUT_MS_CONFIG.
        // We recommend that you create a separate thread pool to consume messages and asyn
chronously return the results.
        for (ConsumerRecord<String, String> record : records) {
          System.out.println(String.format("Consume partition:%d offset:%d", record.partition(
), record.offset()));
        }
      } catch (Exception e) {
        try {
          Thread.sleep(1000);
        } catch (Throwable ignore) {
        }
        e.printStackTrace();
      }
    }
```

```
  }
}
```

b. Compile and run *KafkaConsumerDemo.java*.

- Multiple consumers subscribe to messages

a. Create a multi-consumer subscription program *KafkaMultiConsumerDemo.java*.

```
import java.util.ArrayList;
import java.util.List;
import java.util.Properties;
import java.util.concurrent.atomic.AtomicBoolean;
import org.apache.kafka.clients.CommonClientConfigs;
import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.common.config.SaslConfigs;
import org.apache.kafka.common.errors.WakeupException;
/**
 * This tutorial demonstrates how to enable multiple consumers to simultaneously consume top
ics in one process.
 * Ensure that the number of global consumers does not exceed the total number of partitions of
the topics that the consumers subscribed to.
 */
public class KafkaMultiConsumerDemo {
  public static void main(String args[]) throws InterruptedException {
    // Set the path of the JAAS configuration file.
    JavaKafkaConfigurer.configureSaslScram();
    // Load kafka.properties.
    Properties kafkaProperties = JavaKafkaConfigurer.getKafkaProperties();
    Properties props = new Properties();
    // Specify the endpoint. We recommend that you obtain the endpoint of the corresponding t
opic in the console.
    props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, kafkaProperties.getProperty("b
ootstrap.servers"));
    // Specify the access protocol.
    props.put(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG, "SASL_PLAINTEXT");
    // Specify the SCRAM mechanism.
    props.put(SaslConfigs.SASL_MECHANISM, "SCRAM-SHA-256");
    // Specify the session timeout period. If the consumer does not return a heartbeat before the
session times out, the broker determines that the consumer is not alive. Then the broker remov
```

```
session times out, the broker determines that the consumer is not alive. Then the broker removes the consumer from the consumer group and triggers rebalancing. The default value is 30s.
    props.put(ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG, 30000);
    // The maximum number of messages that can be polled at a time.
    // Do not set the number to an excessively large value. If excessive messages are polled but fail to be completely consumed before the next poll starts, load balancing is triggered, which causes freezing.
    props.put(ConsumerConfig.MAX_POLL_RECORDS_CONFIG, 30);
    // Specify the method for deserializing messages.
    props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, "org.apache.kafka.common.serialization.StringDeserializer");
    props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, "org.apache.kafka.common.serialization.StringDeserializer");
    // The consumer group of the current consumer instance. Enter the consumer group you have created in the console.
    // The consumer instances that belong to the same consumer group. These instances consume messages in load balancing mode.
    props.put(ConsumerConfig.GROUP_ID_CONFIG, kafkaProperties.getProperty("group.id"));
    int consumerNum = 2;
    Thread[] consumerThreads = new Thread[consumerNum];
    for (int i = 0; i < consumerNum; i++) {
      KafkaConsumer<String, String> consumer = new KafkaConsumer<String, String>(props);
      List<String> subscribedTopics = new ArrayList<String>();
      subscribedTopics.add(kafkaProperties.getProperty("topic"));
      consumer.subscribe(subscribedTopics);
      KafkaConsumerRunner kafkaConsumerRunner = new KafkaConsumerRunner(consumer);
      consumerThreads[i] = new Thread(kafkaConsumerRunner);
    }
    for (int i = 0; i < consumerNum; i++) {
      consumerThreads[i].start();
    }
    for (int i = 0; i < consumerNum; i++) {
      consumerThreads[i].join();
    }
  }
  static class KafkaConsumerRunner implements Runnable {
    private final AtomicBoolean closed = new AtomicBoolean(false);
    private final KafkaConsumer consumer;
    KafkaConsumerRunner(KafkaConsumer consumer) {
      this.consumer = consumer;
    }
```

```
    @Override
    public void run() {
      try {
        while (! closed.get()) {
          try {
            ConsumerRecords<String, String> records = consumer.poll(1000);
            // The messages must be completely consumed before the next polling cycle starts. T
he total duration cannot exceed the timeout interval specified by SESSION_TIMEOUT_MS_CONF
IG.
            for (ConsumerRecord<String, String> record : records) {
              System.out.println(String.format("Thread:%s Consume partition:%d offset:%d", T
hread.currentThread().getName(), record.partition(), record.offset()));
            }
          } catch (Exception e) {
            try {
              Thread.sleep(1000);
            } catch (Throwable ignore) {
            }
            e.printStackTrace();
          }
        }
      } catch (WakeupException e) {
        // Ignore exceptions if it is disabled.
        if (! closed.get()) {
          throw e;
        }
      } finally {
        consumer.close();
      }
    }
    // The shutdown hook that can be called by another thread.
    public void shutdown() {
      closed.set(true);
      consumer.wakeup();
    }
  }
}
```

b. Compile and run *KafkaMultiConsumerDemo.java*.

3.SDK for Python

3.1. Overview

A Python client can connect to Message Queue for Apache Kafka and send and subscribe to messages over various endpoints.

The following table lists the endpoints provided by Message Queue for Apache Kafka .

Item	Default endpoint	Simple Authentication and Security Layer (SASL) endpoint
Network	VPC	VPC
Protocol	PLAINTEXT	SASL_PLAINTEXT
Port	9092	9094
SASL mechanism	N/A	• PLAIN: a simple username and password verification mechanism. Message Queue for Apache Kafka improves the PLAIN mechanism and allows you to dynamically add SASL users without restarting the instance. • SCRAM: a username and password verification mechanism, providing higher security than PLAIN. Message Queue for Apache Kafka uses SCRAM-SHA-256.
Demo	PLAINTEXT	None
Documentation	Use the default endpoint to send and subscribe to messages	None

3.2. Use the default endpoint to send and subscribe to messages

This topic describes how a Python client uses SDK for Python to connect to the default endpoint of Message Queue for Apache Kafka and send and subscribe to messages in a virtual private cloud (VPC).

Prerequisites

- Install Python

  > **Note** Python 2.7, 3.4, 3.5, 3.6, and 3.7 are supported.

- Install pip

Install the Python library

1. Run the following command to install the Python library:

```
pip install kafka-python
```

Prepare configurations

1. Create a Kafka configuration file *setting.py*.

```
kafka_setting = {
  'bootstrap_servers': ["XXX", "XXX", "XXX"],
  'topic_name': 'XXX',
  'consumer_id': 'XXX'
}
```

Parameter	Description
bootstrap_servers	The default endpoint of the Message Queue for Apache Kafka instance. You can obtain the default endpoint in the **Basic Information** section of the **Instance Details** page in the Message Queue for Apache Kafka console.
topic_name	The name of the topic. You can obtain the name of the topic on the **Topics** page in the Message Queue for Apache Kafka console.
consumer_id	The name of the consumer group. You can obtain the name of the consumer group on the **Consumer Groups** page in the Message Queue for Apache Kafka console.

Send messages

1. Create a message sender *aliyun_kafka_producer.py*.

```
#! /usr/bin/env python
# encoding: utf-8
import socket
from kafka import KafkaProducer
from kafka.errors import KafkaError
import setting
conf = setting.kafka_setting
print conf
producer = KafkaProducer(bootstrap_servers=conf['bootstrap_servers'],
                         api_version = (0,10),
                         retries=5)
partitions = producer.partitions_for(conf['topic_name'])
print 'partition in topic: %s' % partitions
try:
  future = producer.send(conf['topic_name'], 'hello aliyun-kafka!')
  future.get()
  print 'send message succeed.'
except KafkaError, e:
  print 'send message failed.'
  print e
```

2. Run the following command to send messages:

```
python aliyun_kafka_producer.py
```

Subscribe to messages

1. Create a subscription program *aliyun_kafka_consumer.py*.

```
#! /usr/bin/env python
# encoding: utf-8
import socket
from kafka import KafkaConsumer
from kafka.errors import KafkaError
import setting
conf = setting.kafka_setting
consumer = KafkaConsumer(bootstrap_servers=conf['bootstrap_servers'],
                         group_id=conf['consumer_id'],
                         api_version = (0,10,2),
                         session_timeout_ms=25000,
                         max_poll_records=100,
                         fetch_max_bytes=1 * 1024 * 1024)
print 'consumer start to consuming...'
consumer.subscribe((conf['topic_name'], ))
for message in consumer:
  print message.topic, message.offset, message.key, message.value, message.value, message.partition
```

2. Run the following command to consume messages:

```
python aliyun_kafka_consumer.py
```

4.SDK for C++

4.1. Overview

A C++ client can connect to Message Queue for Apache Kafka and send and subscribe to messages over various endpoints.

The following table lists the endpoints provided by Message Queue for Apache Kafka .

Item	Default endpoint	Simple Authentication and Security Layer (SASL) endpoint
Network	VPC	VPC
Protocol	PLAINTEXT	SASL_PLAINTEXT
Port	9092	9094
SASL mechanism	N/A	• PLAIN: a simple username and password verification mechanism. Message Queue for Apache Kafka improves the PLAIN mechanism and allows you to dynamically add SASL users without restarting the instance. • SCRAM: a username and password verification mechanism, providing higher security than PLAIN. Message Queue for Apache Kafka uses SCRAM-SHA-256.
Demo	PLAINTEXT	None
Documentation	Use the default endpoint to send and subscribe to messages	None

4.2. Use the default endpoint to send and subscribe to messages

This topic describes how a C++ client uses SDK for C++ to connect to the default endpoint of Message Queue for Apache Kafka and send and subscribe to messages in a virtual private cloud (VPC).

Prerequisites

GNU Compiler Collection (GCC) is installed. For more information, see Install GCC .

Install the C++ library

1. Run the following command to switch to the yum source configuration Directory: */etc/yum.repos.d/*.

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

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

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

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

```
yum install librdkafka-devel
```

Send messages

1. Create a message sender *kafka_producer.c*.

```
/*
 * 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>
/* Typical include path would be <librdkafka/rdkafka.h>, but this program
 * is builtin from within the librdkafka source tree and thus differs. */
#include "librdkafka/rdkafka.h"
static int run = 1;
/**
 * @brief Signal termination of program
 */
static void stop (int sig) {
    run = 0;
    fclose(stdin); /* abort fgets() */
}
/**
 * @brief Message delivery report callback.
 *
 * This callback is called exactly once per message, indicating if
 * the message was succesfully delivered
 * (rkmessage->err == RD_KAFKA_RESP_ERR_NO_ERROR) or permanently
 * failed delivery (rkmessage->err ! = RD_KAFKA_RESP_ERR_NO_ERROR).
 *
 * The callback is triggered from rd_kafka_poll() and executes on
 * the application's thread.
 */
static void dr_msg_cb (rd_kafka_t *rk,
                const rd_kafka_message_t *rkmessage, void *opaque) {
    if (rkmessage->err)
        fprintf(stderr, "%% Message delivery failed: %s\n",
            rd_kafka_err2str(rkmessage->err));
    else
        fprintf(stderr,
            "%% Message delivered (%zd bytes, "
```

```
                %%Message delivered (%zd bytes,
                "partition %"PRId32")\n",
                rkmessage->len, rkmessage->partition);
        /* The rkmessage is destroyed automatically by librdkafka */
}
int main (int argc, char **argv) {
        rd_kafka_t *rk;         /* Producer instance handle */
        rd_kafka_topic_t *rkt;  /* Topic object */
        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 */
        /*
         * Argument validation
         */
        if (argc ! = 3) {
                fprintf(stderr, "%% Usage: %s <broker> <topic>\n", argv[0]);
                return 1;
        }
        brokers = argv[1];
        topic   = argv[2];
        /*
         * 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 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. */
        rd_kafka_conf_set_dr_msg_cb(conf, dr_msg_cb);
        /*
```

```
 * Create producer instance.
 *
 * NOTE: rd_kafka_new() takes ownership of the conf object
 *     and the application must not reference it again after
 *     this call.
 */
rk = rd_kafka_new(RD_KAFKA_PRODUCER, conf, errstr, sizeof(errstr));
if (! rk) {
    fprintf(stderr,
        "%% Failed to create new producer: %s\n", errstr);
    return 1;
}
/* Create topic object that will be reused for each message
 * produced.
 *
 * Both the producer instance (rd_kafka_t) and topic objects (topic_t)
 * are long-lived objects that should be reused as much as possible.
 */
rkt = rd_kafka_topic_new(rk, topic, NULL);
if (! rkt) {
    fprintf(stderr, "%% Failed to create topic object: %s\n",
        rd_kafka_err2str(rd_kafka_last_error()));
    rd_kafka_destroy(rk);
    return 1;
}
/* Signal handler for clean shutdown */
signal(SIGINT, stop);
fprintf(stderr,
    "%% Type some text and hit enter to produce message\n"
    "%% Or just hit enter to only serve delivery reports\n"
    "%% Press Ctrl-C or Ctrl-D to exit\n");
while (run && fgets(buf, sizeof(buf), stdin)) {
    size_t len = strlen(buf);
    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:
    if (rd_kafka_produce(
            /* Topic object */
            rkt,
            /* Use builtin partitioner to select partition*/
            RD_KAFKA_PARTITION_UA,
            /* Make a copy of the payload. */
            RD_KAFKA_MSG_F_COPY,
            /* Message payload (value) and length */
            buf, len,
            /* Optional key and its length */
            NULL, 0,
            /* Message opaque, provided in
             * delivery report callback as
             * msg_opaque. */
            NULL) == -1) {
        /**
         * Failed to *enqueue* message for producing.
         */
        fprintf(stderr,
            "%% Failed to produce to topic %s: %s\n",
            rd_kafka_topic_name(rkt),
            rd_kafka_err2str(rd_kafka_last_error()));
        /* Poll to handle delivery reports */
        if (rd_kafka_last_error() ==
          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
```

```
                    delivery report callback to be called.
                    *
                    * The internal queue is limited by the
                    * configuration property
                    * queue.buffering.max.messages */
                    rd_kafka_poll(rk, 1000/*block for max 1000ms*/);
                    goto retry;
                }
            } else {
                fprintf(stderr, "%% Enqueued message (%zd bytes) "
                    "for topic %s\n",
                    len, rd_kafka_topic_name(rkt));
            }
            /* A producer application should continually serve
             * the delivery report queue by calling rd_kafka_poll()
             * at frequent intervals.
             * Either put the poll call in your main loop, or in a
             * dedicated thread, or call it after every
             * rd_kafka_produce() call.
             * Just make sure that rd_kafka_poll() is still called
             * during periods where you are not producing any messages
             * to make sure previously produced messages have their
             * delivery report callback served (and any other callbacks
             * you register). */
            rd_kafka_poll(rk, 0/*non-blocking*/);
        }
        /* Wait for final messages to be delivered or fail.
         * rd_kafka_flush() is an abstraction over rd_kafka_poll() which
         * waits for all messages to be delivered. */
        fprintf(stderr, "%% Flushing final messages..\n");
        rd_kafka_flush(rk, 10*1000 /* wait for max 10 seconds */);
        /* Destroy topic object */
        rd_kafka_topic_destroy(rkt);
        /* Destroy the producer instance */
        rd_kafka_destroy(rk);
        return 0;
    }
```

2. Run the following command to compile *kafka_producer.c*:

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

3. Run the following command to send messages:

```
./kafka_producer <bootstrap_servers> <topic>
```

Parameter	Description
bootstrap_servers	The default endpoint of the Message Queue for Apache Kafka instance. You can obtain the default endpoint in the **Basic Information** section of the **Instance Details** page in the Message Queue for Apache Kafka console.
topic	The name of the topic. You can obtain the name of the topic on the **Topics** page in the Message Queue for Apache Kafka console.

Subscribe to messages

1. Create a subscription program *kafka_consumer.c*.

```
/*
 * librdkafka - Apache Kafka C library
 *
 * Copyright (c) 2015, 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.
```

```
 */
/**
 * Apache Kafka high level consumer example program
 * using the Kafka driver from librdkafka
 * (https://github.com/edenhill/librdkafka)
 */
#include <ctype.h>
#include <signal.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <syslog.h>
#include <sys/time.h>
#include <errno.h>
#include <getopt.h>
/* Typical include path would be <librdkafka/rdkafka.h>, but this program
 * is builtin from within the librdkafka source tree and thus differs. */
#include "librdkafka/rdkafka.h"  /* for Kafka driver */
static int run = 1;
static rd_kafka_t *rk;
static int exit_eof = 0;
static int wait_eof = 0;  /* number of partitions awaiting EOF */
static int quiet = 0;
static    enum {
  OUTPUT_HEXDUMP,
  OUTPUT_RAW,
} output = OUTPUT_HEXDUMP;
static void stop (int sig) {
    if (! run)
        exit(1);
  run = 0;
  fclose(stdin); /* abort fgets() */
}
static void hexdump (FILE *fp, const char *name, const void *ptr, size_t len) {
  const char *p = (const char *)ptr;
  unsigned int of = 0;
  if (name)
    fprintf(fp, "%s hexdump (%zd bytes):\n", name, len);
  for (of = 0 ; of < len ; of += 16) {
    char hexen[16*3+1];
```

```
      char charen[16+1];
      int hof = 0;
      int cof = 0;
      int i;
      for (i = of ; i < (int)of + 16 && i < (int)len ; i++) {
        hof += sprintf(hexen+hof, "%02x ", p[i] & 0xff);
        cof += sprintf(charen+cof, "%c",
              isprint((int)p[i]) ? p[i] : '.');
      }
      fprintf(fp, "%08x: %-48s %-16s\n",
        of, hexen, charen);
    }
}
/**
 * Kafka logger callback (optional)
 */
static void logger (const rd_kafka_t *rk, int level,
        const char *fac, const char *buf) {
  struct timeval tv;
  gettimeofday(&tv, NULL);
  fprintf(stdout, "%u.%03u RDKAFKA-%i-%s: %s: %s\n",
    (int)tv.tv_sec, (int)(tv.tv_usec / 1000),
    level, fac, rd_kafka_name(rk), buf);
}
/**
 * Handle and print a consumed message.
 * Internally crafted messages are also used to propagate state from
 * librdkafka to the application. The application needs to check
 * the `rkmessage->err` field for this purpose.
 */
static void msg_consume (rd_kafka_message_t *rkmessage) {
  if (rkmessage->err) {
    if (rkmessage->err == RD_KAFKA_RESP_ERR__PARTITION_EOF) {
      fprintf(stderr,
        "%% Consumer reached end of %s [%"PRId32"] "
          "message queue at offset %"PRId64"\n",
          rd_kafka_topic_name(rkmessage->rkt),
          rkmessage->partition, rkmessage->offset);
      if (exit_eof && --wait_eof == 0) {
            fprintf(stderr,
              "%% All partition(s) reached EOF: "
```

```
                    "exiting\n");
        run = 0;
            }
    return;
  }
        if (rkmessage->rkt)
            fprintf(stderr, "%% Consume error for "
                "topic \"%s\" [%"PRId32"] "
                "offset %"PRId64": %s\n",
                rd_kafka_topic_name(rkmessage->rkt),
                rkmessage->partition,
                rkmessage->offset,
                rd_kafka_message_errstr(rkmessage));
        else
            fprintf(stderr, "%% Consumer error: %s: %s\n",
                rd_kafka_err2str(rkmessage->err),
                rd_kafka_message_errstr(rkmessage));
        if (rkmessage->err == RD_KAFKA_RESP_ERR__UNKNOWN_PARTITION ||
          rkmessage->err == RD_KAFKA_RESP_ERR__UNKNOWN_TOPIC)
            run = 0;
  return;
}
if (! quiet)
  fprintf(stdout, "%% Message (topic %s [%"PRId32"], "
            "offset %"PRId64", %zd bytes):\n",
            rd_kafka_topic_name(rkmessage->rkt),
            rkmessage->partition,
    rkmessage->offset, rkmessage->len);
if (rkmessage->key_len) {
  if (output == OUTPUT_HEXDUMP)
    hexdump(stdout, "Message Key",
      rkmessage->key, rkmessage->key_len);
  else
    printf("Key: %.*s\n",
        (int)rkmessage->key_len, (char *)rkmessage->key);
}
if (output == OUTPUT_HEXDUMP)
  hexdump(stdout, "Message Payload",
    rkmessage->payload, rkmessage->len);
else
  printf("%.*s\n"
```

```
        (int)rkmessage->len, (char *)rkmessage->payload);
}
static void print_partition_list (FILE *fp,
                      const rd_kafka_topic_partition_list_t
                      *partitions) {
    int i;
    for (i = 0 ; i < partitions->cnt ; i++) {
        fprintf(stderr, "%s %s [%"PRId32"] offset %"PRId64,
            i > 0 ? ",":"",
            partitions->elems[i].topic,
            partitions->elems[i].partition,
      partitions->elems[i].offset);
    }
    fprintf(stderr, "\n");
}
static void rebalance_cb (rd_kafka_t *rk,
              rd_kafka_resp_err_t err,
        rd_kafka_topic_partition_list_t *partitions,
              void *opaque) {
  fprintf(stderr, "%% Consumer group rebalanced: ");
  switch (err)
  {
  case RD_KAFKA_RESP_ERR__ASSIGN_PARTITIONS:
    fprintf(stderr, "assigned:\n");
    print_partition_list(stderr, partitions);
    rd_kafka_assign(rk, partitions);
    wait_eof += partitions->cnt;
    break;
  case RD_KAFKA_RESP_ERR__REVOKE_PARTITIONS:
    fprintf(stderr, "revoked:\n");
    print_partition_list(stderr, partitions);
    rd_kafka_assign(rk, NULL);
    wait_eof = 0;
    break;
  default:
    fprintf(stderr, "failed: %s\n",
            rd_kafka_err2str(err));
        rd_kafka_assign(rk, NULL);
    break;
  }
```

```
}
static int describe_groups (rd_kafka_t *rk, const char *group) {
    rd_kafka_resp_err_t err;
    const struct rd_kafka_group_list *grplist;
    int i;
    err = rd_kafka_list_groups(rk, group, &grplist, 10000);
    if (err) {
        fprintf(stderr, "%% Failed to acquire group list: %s\n",
            rd_kafka_err2str(err));
        return -1;
    }
    for (i = 0 ; i < grplist->group_cnt ; i++) {
        const struct rd_kafka_group_info *gi = &grplist->groups[i];
        int j;
        printf("Group \"%s\" in state %s on broker %d (%s:%d)\n",
            gi->group, gi->state,
            gi->broker.id, gi->broker.host, gi->broker.port);
        if (gi->err)
            printf(" Error: %s\n", rd_kafka_err2str(gi->err));
        printf(" Protocol type \"%s\", protocol \"%s\", "
            "with %d member(s):\n",
            gi->protocol_type, gi->protocol, gi->member_cnt);
        for (j = 0 ; j < gi->member_cnt ; j++) {
            const struct rd_kafka_group_member_info *mi;
            mi = &gi->members[j];
            printf("  \"%s\", client id \"%s\" on host %s\n",
                mi->member_id, mi->client_id, mi->client_host);
            printf("    metadata: %d bytes\n",
                mi->member_metadata_size);
            printf("    assignment: %d bytes\n",
                mi->member_assignment_size);
        }
        printf("\n");
    }
    if (group && ! grplist->group_cnt)
        fprintf(stderr, "%% No matching group (%s)\n", group);
    rd_kafka_group_list_destroy(grplist);
    return 0;
}
static void sig_usr1 (int sig) {
  rd_kafka_dump(stdout, rk);
```

```
}
int main (int argc, char **argv) {
    char mode = 'C';
  char *brokers = "localhost:9092";
  int opt;
  rd_kafka_conf_t *conf;
  rd_kafka_topic_conf_t *topic_conf;
  char errstr[512];
  const char *debug = NULL;
  int do_conf_dump = 0;
  char tmp[16];
    rd_kafka_resp_err_t err;
    char *group = NULL;
    rd_kafka_topic_partition_list_t *topics;
    int is_subscription;
    int i;
  quiet = ! isatty(STDIN_FILENO);
  /* Kafka configuration */
  conf = rd_kafka_conf_new();
    /* Set logger */
    rd_kafka_conf_set_log_cb(conf, logger);
  /* Quick termination */
  snprintf(tmp, sizeof(tmp), "%i", SIGIO);
  rd_kafka_conf_set(conf, "internal.termination.signal", tmp, NULL, 0);
  /* Topic configuration */
  topic_conf = rd_kafka_topic_conf_new();
  while ((opt = getopt(argc, argv, "g:b:qd:eX:ADO")) ! = -1) {
    switch (opt) {
    case 'b':
      brokers = optarg;
      break;
        case 'g':
            group = optarg;
            break;
    case 'e':
      exit_eof = 1;
      break;
    case 'd':
      debug = optarg;
      break;
    case 'q':
```

```
case 'q':
  quiet = 1;
  break;
case 'A':
  output = OUTPUT_RAW;
  break;
case 'X':
{
  char *name, *val;
  rd_kafka_conf_res_t res;
  if (! strcmp(optarg, "list") ||
    ! strcmp(optarg, "help")) {
    rd_kafka_conf_properties_show(stdout);
    exit(0);
  }
  if (! strcmp(optarg, "dump")) {
    do_conf_dump = 1;
    continue;
  }
  name = optarg;
  if (!( val = strchr(name, '='))) {
    fprintf(stderr, "%% Expected "
      "-X property=value, not %s\n", name);
    exit(1);
  }
  *val = '\0';
  val++;
  res = RD_KAFKA_CONF_UNKNOWN;
  /* Try "topic." prefixed properties on topic
   * conf first, and then fall through to global if
   * it didnt match a topic configuration property. */
  if (! strncmp(name, "topic.", strlen("topic.")))
    res = rd_kafka_topic_conf_set(topic_conf,
            name+
            strlen("topic."),
            val,
            errstr,
            sizeof(errstr));
  if (res == RD_KAFKA_CONF_UNKNOWN)
    res = rd_kafka_conf_set(conf, name, val,
          errstr, sizeof(errstr));
```

```
      if (res ! = RD_KAFKA_CONF_OK) {
        fprintf(stderr, "%% %s\n", errstr);
        exit(1);
      }
    }
    break;
        case 'D':
        case 'O':
            mode = opt;
            break;
    default:
      goto usage;
    }
}
if (do_conf_dump) {
    const char **arr;
    size_t cnt;
    int pass;
    for (pass = 0 ; pass < 2 ; pass++) {
      if (pass == 0) {
        arr = rd_kafka_conf_dump(conf, &cnt);
        printf("# Global config\n");
      } else {
        printf("# Topic config\n");
        arr = rd_kafka_topic_conf_dump(topic_conf,
                    &cnt);
      }
      for (i = 0 ; i < (int)cnt ; i += 2)
        printf("%s = %s\n",
            arr[i], arr[i+1]);
      printf("\n");
      rd_kafka_conf_dump_free(arr, cnt);
    }
    exit(0);
}
if (strchr("OC", mode) && optind == argc) {
usage:
    fprintf(stderr,
      "Usage: %s [options] <topic[:part]> <topic[:part]>..\n"
      "\n"
      "librdkafka version %s (0x%08x)\n"
```

```
      "\n"
      " Options:\n"
            "  -g <group>     Consumer group (%s)\n"
      "  -b <brokers>   Broker address (%s)\n"
      "  -e           Exit consumer when last message\n"
      "               in partition has been received.\n"
            "  -D           Describe group.\n"
            "  -O           Get commmitted offset(s)\n"
      "  -d [facs..]    Enable debugging contexts:\n"
      "               %s\n"
      "  -q           Be quiet\n"
      "  -A           Raw payload output (consumer)\n"
      "  -X <prop=name> Set arbitrary librdkafka "
      "configuration property\n"
      "           Properties prefixed with \"topic.\" "
      "will be set on topic object.\n"
      "           Use '-X list' to see the full list\n"
      "           of supported properties.\n"
      "\n"
            "For balanced consumer groups use the 'topic1 topic2..'"
            " format\n"
            "and for static assignment use "
            "'topic1:part1 topic1:part2 topic2:part1..'\n"
      "\n",
      argv[0],
      rd_kafka_version_str(), rd_kafka_version(),
            group, brokers,
      RD_KAFKA_DEBUG_CONTEXTS);
    exit(1);
}
signal(SIGINT, stop);
signal(SIGUSR1, sig_usr1);
if (debug &&
    rd_kafka_conf_set(conf, "debug", debug, errstr, sizeof(errstr)) ! =
    RD_KAFKA_CONF_OK) {
    fprintf(stderr, "%% Debug configuration failed: %s: %s\n",
      errstr, debug);
    exit(1);
}
    /*
     * Client/Consumer group
```

```
 * Client/Consumer group
 */
if (strchr("CO", mode)) {
    /* Consumer groups require a group id */
    if (! group)
        group = "rdkafka_consumer_example";
    if (rd_kafka_conf_set(conf, "group.id", group,
                errstr, sizeof(errstr)) !=
      RD_KAFKA_CONF_OK) {
        fprintf(stderr, "%% %s\n", errstr);
        exit(1);
    }
    /* Consumer groups always use broker based offset storage */
    if (rd_kafka_topic_conf_set(topic_conf, "offset.store.method",
                    "broker",
                    errstr, sizeof(errstr)) !=
      RD_KAFKA_CONF_OK) {
        fprintf(stderr, "%% %s\n", errstr);
        exit(1);
    }
    /* Set default topic config for pattern-matched topics. */
    rd_kafka_conf_set_default_topic_conf(conf, topic_conf);
    /* Callback called on partition assignment changes */
    rd_kafka_conf_set_rebalance_cb(conf, rebalance_cb);
    rd_kafka_conf_set(conf, "enable.partition.eof", "true",
              NULL, 0);
}
/* Create Kafka handle */
if (!( rk = rd_kafka_new(RD_KAFKA_CONSUMER, conf,
              errstr, sizeof(errstr)))) {
    fprintf(stderr,
        "%% Failed to create new consumer: %s\n",
        errstr);
    exit(1);
}
/* Add brokers */
if (rd_kafka_brokers_add(rk, brokers) == 0) {
    fprintf(stderr, "%% No valid brokers specified\n");
    exit(1);
}
if (mode == 'D') {
```

```
        int r;
        /* Describe groups */
        r = describe_groups(rk, group);
        rd_kafka_destroy(rk);
        exit(r == -1 ? 1 : 0);
    }
    /* Redirect rd_kafka_poll() to consumer_poll() */
    rd_kafka_poll_set_consumer(rk);
    topics = rd_kafka_topic_partition_list_new(argc - optind);
    is_subscription = 1;
    for (i = optind ; i < argc ; i++) {
        /* Parse "topic[:part] */
        char *topic = argv[i];
        char *t;
        int32_t partition = -1;
        if ((t = strstr(topic, ":"))) {
            *t = '\0';
            partition = atoi(t+1);
            is_subscription = 0; /* is assignment */
            wait_eof++;
        }
        rd_kafka_topic_partition_list_add(topics, topic, partition);
    }
    if (mode == 'O') {
        /* Offset query */
        err = rd_kafka_committed(rk, topics, 5000);
        if (err) {
            fprintf(stderr, "%% Failed to fetch offsets: %s\n",
                rd_kafka_err2str(err));
            exit(1);
        }
        for (i = 0 ; i < topics->cnt ; i++) {
            rd_kafka_topic_partition_t *p = &topics->elems[i];
            printf("Topic \"%s\" partition %"PRId32,
                p->topic, p->partition);
            if (p->err)
                printf(" error %s",
                    rd_kafka_err2str(p->err));
            else {
                printf(" offset %"PRId64"",
                    p->offset);
```

```
                if (p->metadata_size)
                    printf(" (%d bytes of metadata)",
                        (int)p->metadata_size);
            }
            printf("\n");
        }
        goto done;
    }
    if (is_subscription) {
        fprintf(stderr, "%% Subscribing to %d topics\n", topics->cnt);
        if ((err = rd_kafka_subscribe(rk, topics))) {
            fprintf(stderr,
                "%% Failed to start consuming topics: %s\n",
                rd_kafka_err2str(err));
            exit(1);
        }
    } else {
        fprintf(stderr, "%% Assigning %d partitions\n", topics->cnt);
        if ((err = rd_kafka_assign(rk, topics))) {
            fprintf(stderr,
                "%% Failed to assign partitions: %s\n",
                rd_kafka_err2str(err));
        }
    }
    while (run) {
        rd_kafka_message_t *rkmessage;
        rkmessage = rd_kafka_consumer_poll(rk, 1000);
        if (rkmessage) {
            msg_consume(rkmessage);
            rd_kafka_message_destroy(rkmessage);
        }
    }
done:
    err = rd_kafka_consumer_close(rk);
    if (err)
        fprintf(stderr, "%% Failed to close consumer: %s\n",
            rd_kafka_err2str(err));
    else
        fprintf(stderr, "%% Consumer closed\n");
    rd_kafka_topic_partition_list_destroy(topics);
```

```
    /* Destroy handle */
    rd_kafka_destroy(rk);
  /* Let background threads clean up and terminate cleanly. */
  run = 5;
  while (run-- > 0 && rd_kafka_wait_destroyed(1000) == -1)
    printf("Waiting for librdkafka to decommission\n");
  if (run <= 0)
    rd_kafka_dump(stdout, rk);
  return 0;
}
```

2. Run the following command to compile *kafka_consumer.c*:

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

3. Run the following command to consume messages:

```
./kafka_consumer -g <group> -b <bootstrap_servers> <topic>
```

Parameter	Description
group	The name of the consumer group. You can obtain the name of the consumer group on the **Consumer Groups** page in the Message Queue for Apache Kafka console.
bootstrap_servers	The default endpoint of the Message Queue for Apache Kafka instance. You can obtain the default endpoint in the **Basic Information** section of the **Instance Details** page in the Message Queue for Apache Kafka console.
topic	The name of the topic. You can obtain the name of the topic on the **Topics** page in the Message Queue for Apache Kafka console.

5.SDK for Go

5.1. Overview

A Go client can connect to Message Queue for Apache Kafka and send and subscribe to messages over various endpoints.
The following table lists the endpoints provided by Message Queue for Apache Kafka .

Item	Default endpoint	Simple Authentication and Security Layer (SASL) endpoint
Network	VPC	VPC
Protocol	PLAINTEXT	SASL_PLAINTEXT
Port	9092	9094
SASL mechanism	N/A	• PLAIN: a simple username and password verification mechanism. Message Queue for Apache Kafka improves the PLAIN mechanism and allows you to dynamically add SASL users without restarting the instance. • SCRAM: a username and password verification mechanism, providing higher security than PLAIN. Message Queue for Apache Kafka uses SCRAM-SHA-256.
Demo	PLAINTEXT	None
Documentation	Use the default endpoint to send and subscribe to messages	None

5.2. Use the default endpoint to send and subscribe to messages

This topic describes how a Go client uses SDK for Go to connect to the default endpoint of Message Queue for Apache Kafka and send and subscribe to messages in a virtual private cloud (VPC).

Prerequisites

You have installed Go. For more information, see Install Go .

Install the Go libraries

1. Run the following commands to install the Go libraries:
   i. `go get github.com/Shopify/sarama/`

ii. `go get github.com/bsm/sarama-cluster`

2. Run the following commands to compile the Go libraries:
   i. `go install services`
   ii. `go install services/producer`
   iii. `go install services/consumer`

Prepare configurations

1. Create a Kafka configuration file *kafka.json*.

```
{
 "topics": ["XXX"],
 "servers": ["XXX","XXX","XXX"],
 "consumerGroup": "XXX"
}
```

Parameter	Description
topics	The name of the topic. You can obtain the name of the topic on the **Topics** page in the Message Queue for Apache Kafka console.
servers	The default endpoint of the Message Queue for Apache Kafka instance. You can obtain the default endpoint in the **Basic Information** section of the **Instance Details** page in the Message Queue for Apache Kafka console.
consumerGroup	The name of the consumer group. You can obtain the name of the consumer group on the **Consumer Groups** page in the Message Queue for Apache Kafka console.

2. Create a configurator *configs.go*.

```
package configs
import (
  "encoding/json"
  "fmt"
  "io/ioutil"
  "os"
  "path/filepath"
)
var (
  configPath string
)
func init() {
```

```
func init() {
  var err error
  workPath, err := os.Getwd()
  if err ! = nil {
    panic(err)
  }
  configPath = filepath.Join(workPath, "conf")
}
// LoadJsonConfig is used to load a configuration file to config in JSON format.
// config indicates a pointer to a struct. Otherwise, a "panic" error is thrown.
func LoadJsonConfig(config interface{}, filename string) {
  var err error
  var decoder *json.Decoder
  file := OpenFile(filename)
  defer file.Close()
  decoder = json.NewDecoder(file)
  if err = decoder.Decode(config); err ! = nil {
    msg := fmt.Sprintf("Decode json fail for config file at %s. Error: %v", filename, err)
    panic(msg)
  }
  json.Marshal(config)
}
func LoadJsonFile(filename string) (cfg string) {
  file := OpenFile(filename)
  defer file.Close()
  content, err := ioutil.ReadAll(file)
  if err ! = nil {
    msg := fmt.Sprintf("Read config to string error. file at %s. Error: %v", filename, err)
    panic(msg)
  }
  cfg = string(content)
  return cfg
}
func GetFullPath(filename string) string {
  return filepath.Join(configPath, filename)
}
func OpenFile(filename string) *os.File {
  fullPath := filepath.Join(configPath, filename)
  var file *os.File
  var err error
  if file, err = os.Open(fullPath); err ! = nil {
```

```
    msg := fmt.Sprintf("Can not load config at %s. Error: %v", fullPath, err)
    panic(msg)
  }
  return file
}
```

3. Create a configurator *types.go*.

```
package configs
type MqConfig struct {
  Topics    []string `json:"topics"`
  Servers    []string `json:"servers"`
  ConsumerId string   `json:"consumerGroup"`
}
```

Send messages

1. Create a message sender *producer.go*.

```
import (
  "fmt"
  "services"
  "time"
  "strconv"
  "github.com/Shopify/sarama"
)
var cfg *configs.MqConfig
var producer sarama.SyncProducer
func init() {
  fmt.Print("init kafka producer, it may take a few seconds to init the connection\n")
  var err error
  cfg = &configs.MqConfig{}
  configs.LoadJsonConfig(cfg, "kafka.json")
  mqConfig := sarama.NewConfig()
  mqConfig.Producer.Return.Successes = true
  mqConfig.Version=sarama.V0_10_2_1
  if err = mqConfig.Validate(); err ! = nil {
    msg := fmt.Sprintf("Kafka producer config invalidate. config: %v. err: %v", *cfg, err)
    fmt.Println(msg)
    panic(msg)
  }
  producer, err = sarama.NewSyncProducer(cfg.Servers, mqConfig)
  if err ! = nil {
    msg := fmt.Sprintf("Kafak producer create fail. err: %v", err)
```

```
    msg := fmt.Sprintf("Kafak producer create fail. err: %v", err)
    fmt.Println(msg)
    panic(msg)
  }
}
func produce(topic string, key string, content string) error {
  msg := &sarama.ProducerMessage{
    Topic: topic,
    Key:   sarama.StringEncoder(key),
    Value: sarama.StringEncoder(content),
    Timestamp: time.Now(),
  }
  _, _, err := producer.SendMessage(msg)
  if err ! = nil {
    msg := fmt.Sprintf("Send Error topic: %v. key: %v. content: %v", topic, key, content)
    fmt.Println(msg)
    return err
  }
  fmt.Printf("Send OK topic:%s key:%s value:%s\n", topic, key, content)
  return nil
}
func main() {
  key := strconv.FormatInt(time.Now().UTC().UnixNano(), 10)
  value := "this is a kafka message!"
  produce(cfg.Topics[0], key, value)
}
```

2. Run the following command to send messages:

```
go run producer.go
```

Subscribe to messages

1. Create a subscription program *consumer.go*.

```
import (
  "fmt"
  "os"
  "services"
  "os/signal"
  "github.com/Shopify/sarama"
  "github.com/bsm/sarama-cluster"
)
var cfg *configs.MqConfig
```

```
var consumer *cluster.Consumer
var sig chan os.Signal
func init() {
  fmt.Println("init kafka consumer, it may take a few seconds...")
  var err error
  cfg := &configs.MqConfig{}
  configs.LoadJsonConfig(cfg, "kafka.json")
  clusterCfg := cluster.NewConfig()
  clusterCfg.Consumer.Return.Errors = true
  clusterCfg.Consumer.Offsets.Initial = sarama.OffsetOldest
  clusterCfg.Group.Return.Notifications = true
  clusterCfg.Version = sarama.V0_10_2_1
  if err = clusterCfg.Validate(); err ! = nil {
    msg := fmt.Sprintf("Kafka consumer config invalidate. config: %v. err: %v", *clusterCfg, err)
    fmt.Println(msg)
    panic(msg)
  }
  consumer, err = cluster.NewConsumer(cfg.Servers, cfg.ConsumerId, cfg.Topics, clusterCfg)
  if err ! = nil {
    msg := fmt.Sprintf("Create kafka consumer error: %v. config: %v", err, clusterCfg)
    fmt.Println(msg)
    panic(msg)
  }
  sig = make(chan os.Signal, 1)
}
func Start() {
  go consume()
}
func consume() {
  for {
    select {
    case msg, more := <-consumer.Messages():
      if more {
        fmt.Printf("Partition:%d, Offset:%d, Key:%s, Value:%s Timestamp:%s\n", msg.Partition, msg.Off
set, string(msg.Key), string(msg.Value), msg.Timestamp)
        consumer.MarkOffset(msg, "")
      }
    case err, more := <-consumer.Errors():
      if more {
        fmt.Println("Kafka consumer error: %v", err.Error())
```

```
        }
      case ntf, more := <-consumer.Notifications():
        if more {
          fmt.Println("Kafka consumer rebalance: %v", ntf)
        }
      case <-sig:
        fmt.Errorf("Stop consumer server...")
        consumer.Close()
        return
      }
    }
}
func Stop(s os.Signal) {
  fmt.Println("Recived kafka consumer stop signal...")
  sig <- s
  fmt.Println("kafka consumer stopped!!!")
}
func main() {
  signals := make(chan os.Signal, 1)
  signal.Notify(signals, os.Interrupt)
  Start()
  select {
  case s := <-signals:
    Stop(s)
  }
}
```

2. Run the following command to consume messages:

```
go run consumer.go
```

6.SDK for PHP

6.1. Overview

A PHP client can connect to Message Queue for Apache Kafka and send and subscribe to messages over various endpoints.

The following table lists the endpoints provided by Message Queue for Apache Kafka .

Item	Default endpoint	Simple Authentication and Security Layer (SASL) endpoint
Network	VPC	VPC
Protocol	PLAINTEXT	SASL_PLAINTEXT
Port	9092	9094
SASL mechanism	N/A	• PLAIN: a simple username and password verification mechanism. Message Queue for Apache Kafka improves the PLAIN mechanism and allows you to dynamically add SASL users without restarting the instance. • SCRAM: a username and password verification mechanism, providing higher security than PLAIN. Message Queue for Apache Kafka uses SCRAM-SHA-256.
Demo	PLAINTEXT	None
Documentation	Use the default endpoint to send and subscribe to messages	None

6.2. Use the default endpoint to send and subscribe to messages

This topic describes how a PHP client uses SDK for PHP to connect to the default endpoint of Message Queue for Apache Kafka and send and subscribe to messages in a virtual private cloud (VPC).

Prerequisites

- Install GCC
- Install PHP
- Install PECL

Install the C++ library

1. Run the following command to switch to the yum source configuration Directory: */etc/yum.repos.d/*.

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

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

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

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

```
yum install librdkafka-devel
```

Install the PHP library

1. Run the following command to install the PHP library:

```
pecl install rdkafka
```

2. In PHP's initialization file *php.ini* Add the following line statement to enable Kafka extension.

```
extension=rdkafka.so
```

Prepare configurations

1. Create a configuration file *setting.php*.

```
<? php
return [
  'bootstrap_servers' => "xxx:xx,xxx:xx",
  'topic_name' => 'xxx',
  'consumer_id' => 'xxx'
];
```

Parameter	Description
bootstrap_servers	The default endpoint of the Message Queue for Apache Kafka instance. You can obtain the default endpoint in the **Basic Information** section of the **Instance Details** page in the Message Queue for Apache Kafka console.

Parameter	Description
topic_name	The name of the topic. You can obtain the name of the topic on the **Topics** page in the Message Queue for Apache Kafka console.
consumer_id	The name of the consumer group. You can obtain the name of the consumer group on the **Consumer Groups** page in the Message Queue for Apache Kafka console.

Send messages

1. Create a message sender *kafka-producer.php*.

```
<? php
$setting = require __DIR__ . '/setting.php';
$conf = new RdKafka\Conf();
$conf->set('api.version.request', 'true');
$conf->set('message.send.max.retries', 5);
$rk = new RdKafka\Producer($conf);
## If want to debug, set log level to LOG_DEBUG
$rk->setLogLevel(LOG_INFO);
$rk->addBrokers($setting['bootstrap_servers']);
$topic = $rk->newTopic($setting['topic_name']);
$a = $topic->produce(RD_KAFKA_PARTITION_UA, 0, "Message hello kafka");
$rk->poll(0);
while ($rk->getOutQLen() > 0) {
  $rk->poll(50);
}
echo "send succ" . PHP_EOL;
```

2. Run the following command to send messages:

```
php kafka-producer.php
```

Subscribe to messages

1. Create a subscription program *kafka-consumer.php*.

```
<? php
$setting = require __DIR__ . '/setting.php';
$conf = new RdKafka\Conf();
$conf->set('api.version.request', 'true');
$conf->set('group.id', $setting['consumer_id']);
$conf->set('metadata.broker.list', $setting['bootstrap_servers']);
$topicConf = new RdKafka\TopicConf();
$conf->setDefaultTopicConf($topicConf);
$consumer = new RdKafka\KafkaConsumer($conf);
$consumer->subscribe([$setting['topic_name']]);
echo "Waiting for partition assignment... (make take some time when\n";
echo "quickly re-joining the group after leaving it.)\n";
while (true) {
  $message = $consumer->consume(30 * 1000);
  switch ($message->err) {
    case RD_KAFKA_RESP_ERR_NO_ERROR:
      var_dump($message);
      break;
    case RD_KAFKA_RESP_ERR__PARTITION_EOF:
      echo "No more messages; will wait for more\n";
      break;
    case RD_KAFKA_RESP_ERR__TIMED_OUT:
      echo "Timed out\n";
      break;
    default:
      throw new \Exception($message->errstr(), $message->err);
      break;
  }
}
? >
```

2. Run the following command to consume messages:

```
php kafka-consumer.php
```

7.SDK for Ruby

7.1. Overview

A Ruby client can connect to Message Queue for Apache Kafka and send and subscribe to messages over various endpoints.

The following table lists the endpoints provided by Message Queue for Apache Kafka .

Item	Default endpoint	Simple Authentication and Security Layer (SASL) endpoint
Network	VPC	VPC
Protocol	PLAINTEXT	SASL_PLAINTEXT
Port	9092	9094
SASL mechanism	N/A	• PLAIN: a simple username and password verification mechanism. Message Queue for Apache Kafka improves the PLAIN mechanism and allows you to dynamically add SASL users without restarting the instance. • SCRAM: a username and password verification mechanism, providing higher security than PLAIN. Message Queue for Apache Kafka uses SCRAM-SHA-256.
Demo	PLAINTEXT	None
Documentation	Use the default endpoint to send and subscribe to messages	None

7.2. Use the default endpoint to send and subscribe to messages

This topic describes how a Ruby client uses SDK for Ruby to connect to the default endpoint of Message Queue for Apache Kafka and send and subscribe to messages in a virtual private cloud (VPC).

Prerequisites

Ruby is installed. For more information, see Install Ruby .

Install the Ruby library

1. Run the following command to install the Ruby library:

```
gem install ruby-kafka -v 0.6.8
```

Send messages

1. Create a message sender *producer.rb*.

```
# frozen_string_literal: true
$LOAD_PATH.unshift(File.expand_path(".. /../lib", __FILE__))
require "kafka"
logger = Logger.new($stdout)
#logger.level = Logger::DEBUG
logger.level = Logger::INFO
brokers = "xxx:xx,xxx:xx"
topic = "xxx"
kafka = Kafka.new(
    seed_brokers: brokers,
    client_id: "simple-producer",
    logger: logger,
    )
producer = kafka.producer
begin
  $stdin.each_with_index do |line, index|
  producer.produce(line, topic: topic)
  producer.deliver_messages
end
ensure
  producer.deliver_messages
  producer.shutdown
end
```

Parameter	Description
brokers	The default endpoint of the Message Queue for Apache Kafka instance. You can obtain the default endpoint in the **Basic Information** section of the **Instance Details** page in the Message Queue for Apache Kafka console.
topic	The name of the topic. You can obtain the name of the topic on the **Topics** page in the Message Queue for Apache Kafka console.

2. Run the following command to send messages:

```
ruby producer.rb
```

Subscribe to messages

1. Create a subscription program *consumer.rb*.

```
# frozen_string_literal: true
$LOAD_PATH.unshift(File.expand_path(".. /../lib", __FILE__))
require "kafka"
logger = Logger.new(STDOUT)
#logger.level = Logger::DEBUG
logger.level = Logger::INFO
brokers = "XXX:XX,XXX:XX"
topic = "XXX"
consumerGroup = "XXX"
kafka = Kafka.new(
    seed_brokers: brokers,
    client_id: "test",
    socket_timeout: 20,
    logger: logger,
    )
consumer = kafka.consumer(group_id: consumerGroup)
consumer.subscribe(topic, start_from_beginning: false)
trap("TERM") { consumer.stop }
trap("INT") { consumer.stop }
begin
  consumer.each_message(max_bytes: 64 * 1024) do |message|
  logger.info("Get message: #{message.value}")
  end
rescue Kafka::ProcessingError => e
  warn "Got error: #{e.cause}"
  consumer.pause(e.topic, e.partition, timeout: 20)
  retry
end
```

Parameter	Description
brokers	The default endpoint of the Message Queue for Apache Kafka instance. You can obtain the default endpoint in the **Basic Information** section of the **Instance Details** page in the Message Queue for Apache Kafka console.

Parameter	Description
topic	The name of the topic. You can obtain the name of the topic on the **Topics** page in the Message Queue for Apache Kafka console.
consumerGroup	The name of the consumer group. You can obtain the name of the consumer group on the **Consumer Groups** page in the Message Queue for Apache Kafka console.

2. Run the following command to consume messages:

```
ruby consumer.rb
```

8.SDK for Node.js

8.1. Overview

A Node.js client can connect to Message Queue for Apache Kafka and send and receive messages over various endpoints.

Item	Default endpoint	Simple Authentication and Security Layer (SASL) endpoint
Network	VPC	VPC
Protocol	PLAINTEXT	SASL_PLAINTEXT
Port	9092	9094
SASL mechanism	N/A	• PLAIN: a simple username and password verification mechanism. Message Queue for Apache Kafka improves the PLAIN mechanism and allows you to dynamically add SASL users without restarting the instance. • SCRAM: a username and password verification mechanism, providing higher security than PLAIN. Message Queue for Apache Kafka uses SCRAM-SHA-256.
Demo	PLAINTEXT	None
Documentation	Use the default endpoint to send and subscribe to messages	None

8.2. Use the default endpoint to send and subscribe to messages

This topic describes how a Node.js client uses SDK for Node.js to connect to the default endpoint of Message Queue for Apache Kafka and send and subscribe to messages in a virtual private cloud (VPC).

Prerequisites

- Install GCC
- Install Node.js.

  > **Notice** The version of Node.js must be 4.0.0 or later.

- Install OpenSSL

Install the Node.js library

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

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

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

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

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

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

Prepare configurations

1. Create a Kafka configuration file *setting.js*.

```
module.exports = {
  'bootstrap_servers': ["kafka-ons-internet.aliyun.com:8080"],
  'topic_name': 'xxx',
  'consumer_id': 'xxx'
}
```

Parameter	Description
bootstrap_servers	The default endpoint of the Message Queue for Apache Kafka instance. You can obtain the default endpoint in the **Basic Information** section of the **Instance Details** page in the Message Queue for Apache Kafka console.
topic_name	The name of the topic. You can obtain the name of the topic on the **Topics** page in the Message Queue for Apache Kafka console.
consumer_id	The name of the consumer group. You can obtain the name of the consumer group on the **Consumer Groups** page in the Message Queue for Apache Kafka console.

Send messages

1. Create a message sender *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({
  /*'debug': 'all', */
  'api.version.request': 'true',
  'bootstrap.servers': config['bootstrap_servers'],
  'dr_cb': true,
  'dr_msg_cb': true
});
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 Ali Kafka'),
   null,
   Date.now()
  );
 } catch (err) {
  console.error('A problem occurred when sending our message');
  console.error(err);
 }
}
producer.on('delivery-report', function(err, report) {
  console.log("delivery-report: producer ok");
```

```
});
producer.on('event.error', function(err) {
  console.error('event.error:' + err);
})
setInterval(produce,1000,"Interval");
```

2. Run the following command to send messages:

```
node producer.js
```

Subscribe to messages

1. Create a subscription program *consumer.js*.

```
const Kafka = require('node-rdkafka');
const config = require('./setting');
console.log(Kafka.features);
console.log(Kafka.librdkafkaVersion);
console.log(config)
var consumer = new Kafka.KafkaConsumer({
  /*'debug': 'all',*/
  'api.version.request': 'true',
  'bootstrap.servers': config['bootstrap_servers'],
  'group.id' : config['consumer_id']
});
consumer.connect();
consumer.on('ready', function() {
 console.log("connect ok");
 consumer.subscribe([config['topic_name']]);
 consumer.consume();
})
consumer.on('data', function(data) {
 console.log(data);
});
consumer.on('event.log', function(event) {
    console.log("event.log", event);
});
consumer.on('error', function(error) {
  console.log("error:" + error);
});
consumer.on('event', function(event) {
     console.log("event:" + event);
});
```

2. Run the following command to consume messages:

```
node consumer.js
```

9.Sending on clients

9.1. Where can I find the best practices for sending messages?

For more information, see Best practices for producers.

9.2. How can I test the message sending?

On the **Topics** page in the console, you can click **Send Message** next to each topic to test message sending and verify cluster running.

9.3. How can I determine whether a message is sent after I send the message on the client?

Most clients return Callback or Future after a message is sent. If the callback is successful, the message sending is successful.

You can also check whether the message is sent properly in the console in the following ways:

- View the partition status of a topic. The number of messages in each partition is displayed in real time.
- View the traffic monitoring of a topic. The minute-level traffic curve is displayed.

9.4. How many connections to the broker can a producer establish?

Each producer has at least one exclusive TCP connection to each broker to send messages, and the number of connections to process metadata may also increase.

For more information, see How TCP Connections are Managed by kafka-clients scala library.

9.5. Does the callback setting on the Java client affect the message sending speed?

It depends on whether your callback is time-consuming and the setting of the `max.in.flight.requests.per.connection` parameter.

To reduce the time consumed in the callback, we recommend that you not frequently perform time-consuming operations in the callback. Instead, you can perform batch callback after a certain number of acknowledgements (ACKs) are accumulated, or perform the callback in an asynchronous thread without being blocked.

The maximum number of messages that can be sent before the blocking ends is specified by the `max.in.flight.requests.per.connection` parameter.

9.6. Why does a PHP client experience a long delay when it sends a message?

The long sending delay is caused by the PHP language feature.

kafka php sending delay

The PHP client reinitializes a KafkaProducer object each time it sends a message. The initialization involves various operations, such as connecting to each broker and updating metadata. The sending delay is greater than 100 ms in a VPC and 500 ms on the Internet. In contrast, a Java client re-uses KafkaProducer and has a low sending delay.

10.Consumption on clients

10.1. How do I troubleshoot issues that occur upon my first access from the client?

When accessing Message Queue for Apache Kafka for the first time, you may encounter the following issues:

- Network connection
- Client version
- Configuration

You can solve 99% of issues by troubleshooting the preceding three factors step-by-step.

10.2. What can I do if messages are accumulated?

The consumer actively pulls Message Queue for Apache Kafka messages from the broker. Generally, pulling messages from the broker is not a consumption bottleneck due to the batch pulling mechanism.

Message accumulation is generally caused by slow consumption or blocked consumption threads.

We recommend that you print the time consumed by message consumption, or view the stack information for the thread execution information.

> **Notice** You can use jstack to print the stack information of consumer Java processes.

10.3. Why does rebalancing frequently occur on the client?

Problem description

Rebalancing frequently occurs on the client.

Causes

The possible causes are as follows:

- Rebalancing may be frequently triggered for the open-source version 0.10 of Message Queue for Apache Kafka .
- The consumer of Message Queue for Apache Kafka has no independent thread to maintain the heartbeat. Instead, it maintains the heartbeat through the polling interface. Therefore, if the consumption response is slow, the heartbeat times out, causing rebalancing.
  - `session.timeout.ms` specifies the timeout period of heartbeats, which can be set on the client.
  - `max.poll.records` specifies the maximum number of messages returned for each polling.

Solutions

Procedure

1. Check the open-source version of the Message Queue for Apache Kafka instance. If the open-source version of your instance is the version 0.10, we recommend that you upgrade the instance version to the stable version 0.10.2. For more information, see Upgrade the instance version.
2. Adjust the following parameters:
   - Set `session.timeout.ms` below 30 seconds. We recommend that you set it to 25 seconds.
   - Set `max.poll.records` to a value far less than the following result: number of messages consumed by each thread per second x number of consumption threads x number of seconds specified by `session.timeout`.
3. Improve the consumption speed of the client.

10.4. Why does a consumer pull a message slowly from the broker or fail to pull a message from the broker?

Problem description

This problem occurs when a topic contains messages and the consumption does not reach the latest offset.

Causes

The possible causes are as follows:

- The consumption traffic reaches the network bandwidth.
- The traffic for a single message exceeds the network bandwidth.
- The traffic for the messages pulled by the consumer each time exceeds the network bandwidth.

> **Note** The number of messages pulled by a consumer each time is affected by the following parameters:
>
> - max.poll.records: the maximum number of messages pulled each time.
> - fetch.max.bytes: the maximum number of bytes pulled each time.
> - max.partition.fetch.bytes: the maximum number of bytes pulled by each partition each time.

Solutions

Procedure

1. Log on to the Message Queue for Apache Kafka console to query messages. If messages are found, perform the following steps.
2. On the **Monitoring and Alerts** page, check whether the consumption traffic has reached the network bandwidth. If the consumption traffic has reached the network bandwidth, you need to expand the network bandwidth.
3. Check whether the traffic for a single message in the topic exceeds the network bandwidth. If so, increase the network bandwidth or reduce the size of the message.
4. Check whether the traffic for the messages pulled by the consumer each time exceeds the network bandwidth. If so, you need to adjust the following parameters:

- Network bandwidth > fetch.max.bytes
- Network bandwidth > max.partition.fetch.bytes x Total number of subscribed partitions