Message Queue CKafka

Quick Start

Product Documentation

Contents

Quick Start
- Creating Instance
- Creating Topics
- Download Kafka Toolkit
- Getting Started
- Compatibility with Open Source Kafka

Quick Start
Creating Instance

Last updated：2020-07-28 16:23:05

Operation Scenarios

This document describes how to create and view an instance in the CKafka Console.

Directions

Creating instance

1. Log in to the CKafka Console.
2. On the CKafka instance list page, click **Create** to enter the instance purchase page.
3. On the CKafka instance purchase page, set the configuration information for purchase.
   - Billing Mode: pay-as-you-go billing is supported.
   - Region: select a region close to the physical partition
   - AZ: select a purchasable partition
   - Product Specification: select an appropriate model based on the peak bandwidth and disk capacity
   - Message Retention: a period in minutes ranging from 1 minute to 90 days

> **Note：**
>
> After the message retention period is set, messages will be deleted after expiration. Messages are deleted in batches based on CKafka segments but not immediately. Currently, the segment size is 1 GB, and if a segment is less than 1 GB, it will not be deleted. Therefore, if the retention period is set to 1 minute, but the data size of the segment cannot be accumulated to 1 GB in 1 minute, this period will not take effect, and you are recommended to increase the period depending on the speed of data accumulation.

   - Quantity: 1–20
4. Click **Buy Now** to complete the instance creation process.

Viewing instance

1. On the CKafka instance list page, click the "ID/Name" of the target instance to enter the instance details page.
2. On the instance details page, click the **Basic Info** tab to view the configuration information, message retention period, and automatic topic creation of the instance.
   Once you enable the automatic topic creation feature for the server, when you use or access metadata of a topic that does not exist, it will be automatically created with the configured number of replicas and partitions. You can view the automatically created topic in **Topic Management**.

> **Note：**
>
> - The total number of topics that can be automatically created varies by instance specification. For more information, please see Billing Overview.
> - The private IP and port (such as `10.6.206.110:9092`) in **Basic Info** represents the communication address used to get the backend service. There may be multiple ports in a real access address. If access control is configured on your server, open all ports after 9092 to the internet on it.

Creating Topics

Last updated : 2020-07-30 11:40:58

Operation Scenarios

This document guides you through how to create and view a topic under an existing instance in the CKafka Console.

Topic is the name of a category where messages can be stored and published. CKafka uses the concept of topic externally, that is, a producer writes messages to a topic, while a consumer read messages from it. To implement horizontal scaling, a topic actually consists of multiple partitions. When a performance bottleneck occurs, you can scale out the topic by adding more partitions.

Prerequisites

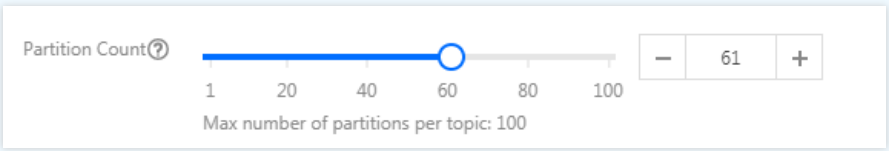
You have created an instance.

Directions

Creating a topic

1. On the CKafka instance list page, click the "ID/Name" of the target instance to enter the instance details page.
2. On the instance details page, click **Topic Management** at the top.
3. On the topic management page, click **Create**.
4. In the topic editing window, set configuration information such as the number of partitions and replicas.
   - Name: the topic name, which cannot be changed once entered and can only contain letters, digits, underscores, "-", and ".".
   - Number of Partitions: partition is a physical concept of partitioning. A topic can contain one or more partitions, and CKafka uses partition as the unit for allocation.
   - Number of Replicas: the number of partition replicas which are used to ensure high availability of partitions. For the sake of data reliability, you cannot create a single-replica topic currently. 2 replicas are enabled by default.
   - Allowlist: after the allowlist is enabled, the topic can be accessed only from IPs in the allowlist, thus effectively protecting data security. (The allowlist can be enabled either on the topic creation page or topic editing page.)
5. Click **Submit** to complete topic creation.

The number of replicas is also considered for partitions. For example, if you create 1 topic, 6 partitions, and 2 replicas, then there will be a total of 1 * 6 * 2 = 12 partitions.
If the maximum number of partitions allowed is exceeded during purchase, the following prompt will be displayed when you click **Submit**:



Viewing a topic

1. On the CKafka instance list page, click the "ID/Name" of the target instance to enter the instance details page.

2. On the instance details page, click the **Topic Management** tab to view the information of a topic such as monitoring metrics, number of partitions, and allowlist.

Deleting a topic

- Once a topic is deleted, messages stored in it will also be deleted. Please do so with caution.
- Topic deletion is an async operation. After deletion is successfully configured, the ZooKeeper configuration will take effect after 1 minute. If you create a topic with the same name as the deleted topic in this period, the system will return error code [4000]10011. In this case, please wait and try again later.

1. On the CKafka instance list page, click the "ID/Name" of the target instance to enter the instance details page.
2. On the instance details page, click **Topic Management** at the top.
3. On the topic management page, click **Delete** in the "Operation" column. Or, use the DeleteTopic API to delete the instance.

Download Kafka Toolkit

Last updated：2020-06-30 14:04:17

Operation Scenarios

This task guides you through how to use the Kafka API after purchasing the CKafka service. After building a CKafka environment on an CVM instance, you need to download and decompress the Kafka toolkit and perform simple testing on the Kafka API.

Prerequisites

- You have already registered a Tencent Cloud account.
- You have already purchased an CVM instance.

Directions

1. Install the JDK environment

You need to log in to the CVM purchase page to purchase a CVM instance first. The instance is configured as follows for this test:

- OS: CentOS 6.8 64-bit
- CPU: 1 core
- Memory: 2 GB
- Public network bandwidth: 1 Mbps

After the purchase, install the JDK to the CVM instance by following the steps below:

1.1 Download the JDK

The JDK can be obtained with the `wget` command. If you need a different version, go to the official website to download one. You are recommended to use a JDK version above 1.7. The version used in this example is JDK 1.7.0_79.

1.2 Move it to a folder and decompress it

```
mkdir /usr/local/jdk
mv jdk-7u79-linux-x64.tar.gz /usr/local/jdk/
cd /usr/local/jdk/
tar -xzvf jdk-7u79-linux-x64.tar.gz
```

1.3 Configure the environment variable

```
vim /etc/profile
```

Add the following environment variable configuration to the end of the file:

```
export JAVA_HOME=/usr/local/jdk/jdk1.7.0_79 (extracted JDK folder)
export JRE_HOME=/usr/local/jdk/jdk1.7.0_79/jre
export PATH=$JAVA_HOME/bin:$JAVA_HOME/jre/bin:$PATH
export CLASSPATH=.:$JAVA_HOME/lib/tools.jar:$JAVA_HOME/lib/dt.jar:$JRE_HOME/lib
```

Run `wq` to save and exit. Then, run `source /etc/profile` to make the file take effect immediately.

1.4 Verify

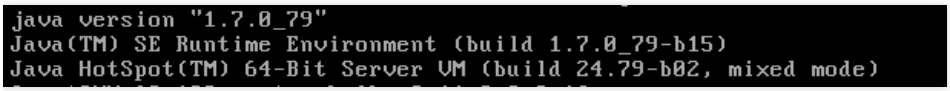
Verify whether the environment has been installed successfully by running the following command (or the `javac` command) and

checking whether the version numbers are the same:

```
cd $JAVA_HOME/bin
./java -version
```

> `$JAVA_HOME` is the home directory of the installed JDK.

If the following codes appear, the JDK has been installed successfully.



2. Download the Kafka toolkit

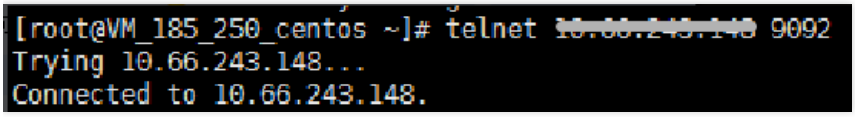
> - The `$ip $port` variable mentioned below refers to the access IP and port of CKafka.
> - A CKafka instance offers up to 6 access points (i.e., $ip $port) which can satisfy the access requests from clients in different network environments. When performing a test, you only need to select the access point corresponding to the specific network environment. For example, if your CVM instance runs in VPC, you can just select the CKafka access point ($ip $port) for VPC. For more information on access points, see the instance details page.

Download the Kafka installation package here and decompress it.
Currently, CKafka is 100% compatible with Kafka 0.9 and 0.10. You are recommended to download the installation package on an appropriate version (preferably v0.10.2).

```
tar -C /opt -xzvf kafka_2.10-0.10.2.0.tgz // Decompressed to the corresponding directory
```

Kafka is ready for use immediately after decompression with no environment configuration required.
You can test whether the CVM instance is connected to the CKafka instance by running the `telnet` command.

```
telnet $ip $port
```



3. Test the Kafka API

Send a message:

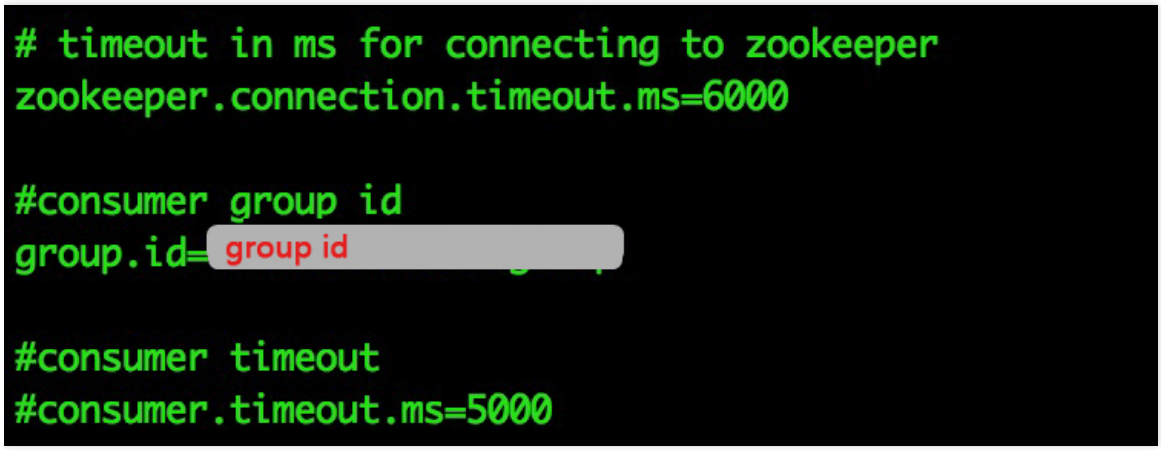
```
./kafka-console-producer.sh --broker-list $ip:$port --topic topicName
This is a message
This is another message
```

Here, the IP in the broker-list is the VIP of the CKafka instance, and topicName is the topic name in the CKafka instance.

Receive a message (CKafka hides the ZooKeeper cluster by default):

```
./kafka-console-consumer.sh --bootstrap-server $ip:$port --from-beginning --new-consumer --topic topicName
This is a message
This is another message
```

In the above command, as no consumer group is specified to consume messages, the system will randomly generate a group to consume messages, and the upper limit on group quantity may be reached quickly. Therefore, you are recommended to **specify a group** to receive messages. First, you need to configure the specified group name in `consumer.properties`, as shown below:

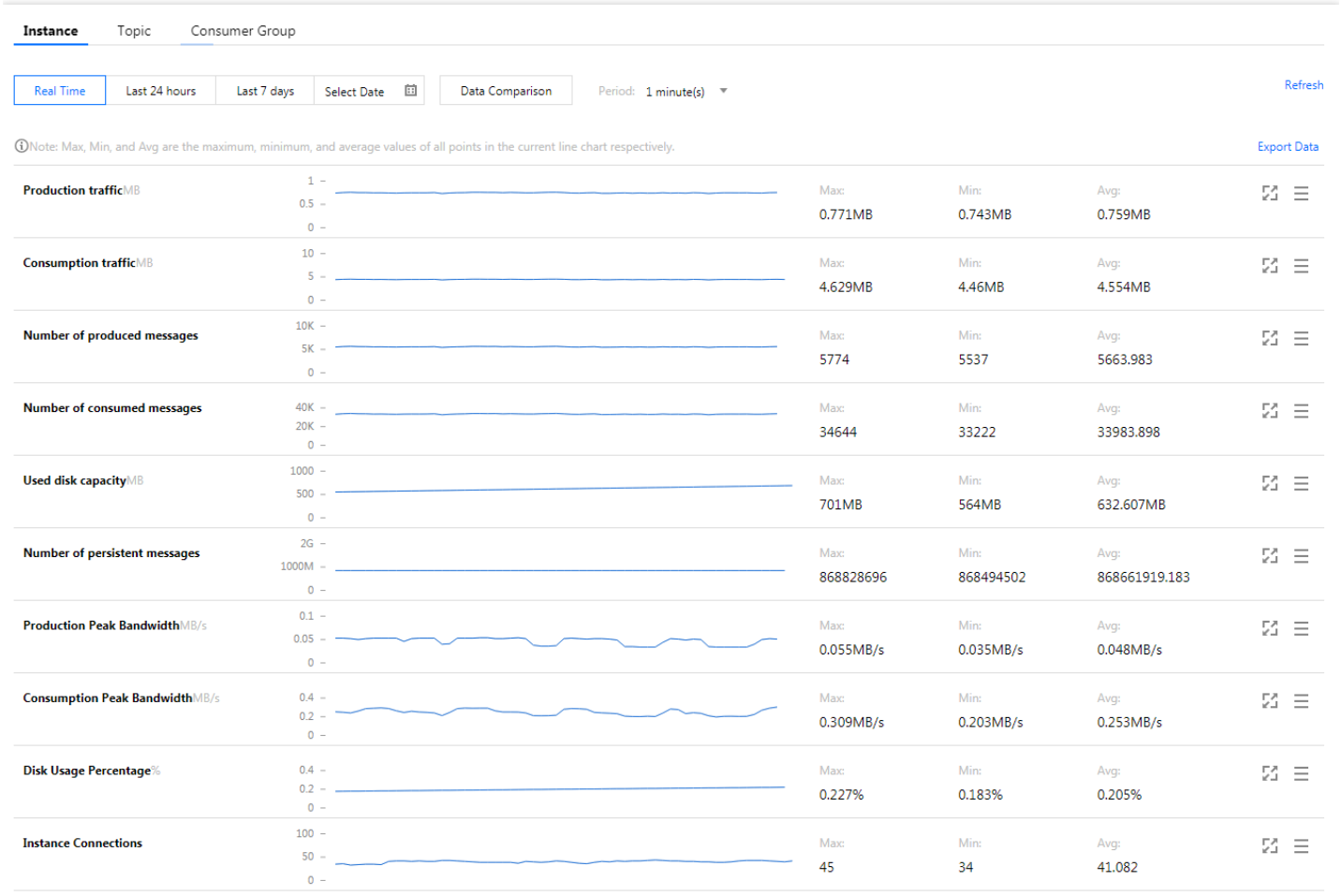


Then, run the following command to specify the consumer group:

```
./kafka-console-consumer.sh --bootstrap-server $ip:$port --from-beginning --new-consumer --topic topicName --consumer.config ../config/consumer.properties
```

When configuring the `ConsumerConfig` parameter, you are recommended to set `auto.offset.reset` to `earliest` so as to prevent messages from being skipped when a new consumer group does not exist.
Cause: When you create a consumer which falls into a new group and the `auto.offset.reset` value is `latest`, latest data (i.e., data produced after creating the consumer) will be consumed, while data previously produced will not be consumed.

View the corresponding CKafka monitor:

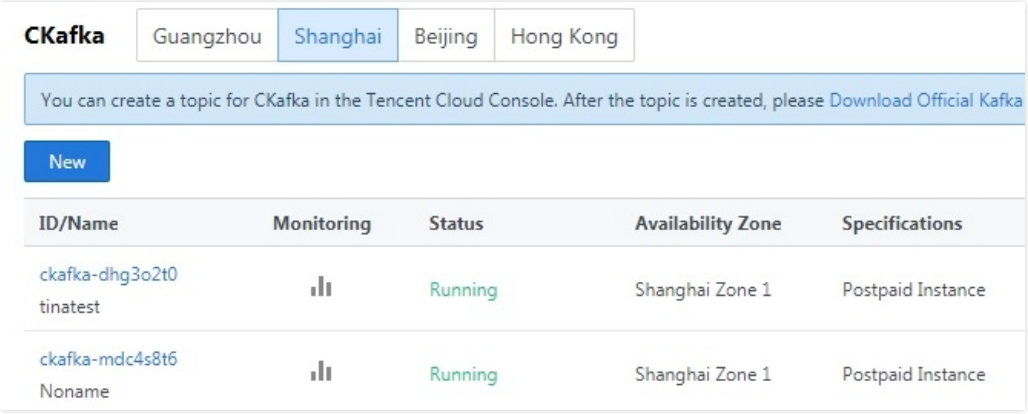


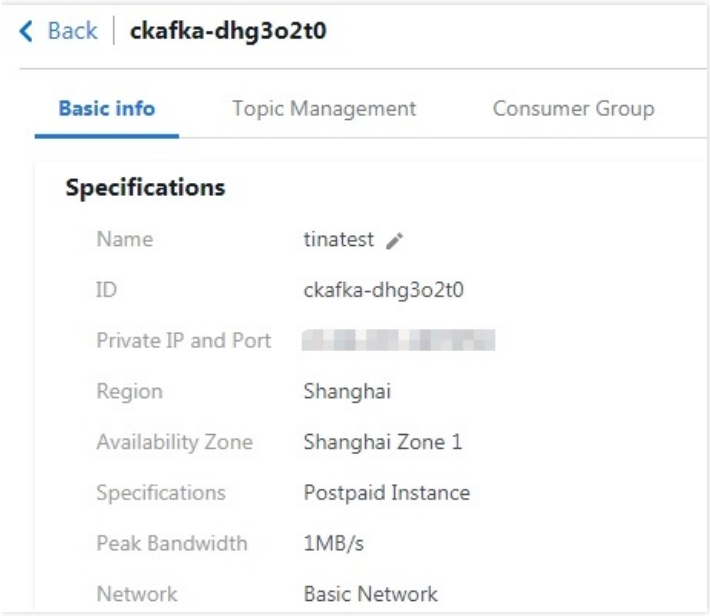
Getting Started

Last updated：2020-07-30 11:29:37

1. View the CKafka instance and create a topic

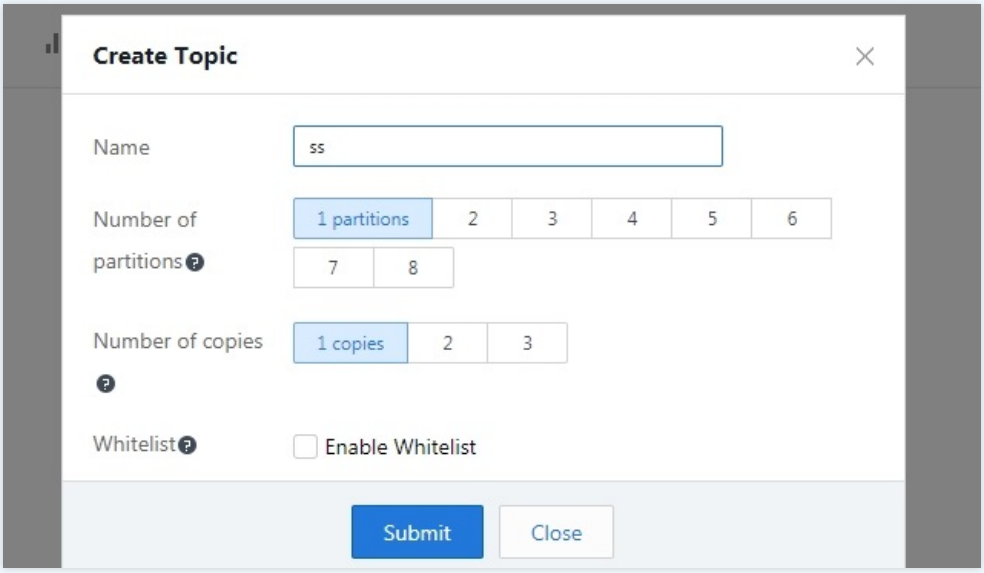
After your application is approved, the CKafka instance is displayed in the CKafka console. Click the instance to view its details, including region, private network vip and port.





In the topic management page, you can create a topic, and specify the number of partitions and replicas for the topic.

Note: The current topic name cannot be changed after input. In addition, when you have specified the number of partitions, you can only add partitions. Number of replicas cannot be changed once being specified.



After the topic and partitions are created, you can perform the production and consumption operations on this instance via the Kafka client on CVM.

2. Download the Kafka toolkit locally

2.1 Install the JDK environment

This document describes how to set up a CKafka environment on your CVM. First, purchase a CVM on the purchase page and then log in to the CVM. The configuration of the test machine in this example is as follows:

Machine configuration
Operating system: CentOS 6.8 64-bit
CPU: 1 core
Memory: 2 GB
Public network bandwidth: 1 Mbps

Next, install JDK on the CVM.
(1). Download JDK, which can be obtained with the wget command. You can also download a different version from the official website.
It is recommended to use a version later than JDK 1.7. The version used in this example is JDK 1.7.0_79.

(2). Move it to a fixed folder and decompress it:

```
mkdir /usr/local/jdk
mv jdk-7u79-linux-x64.tar.gz /usr/local/jdk/
cd /usr/local/jdk/
tar -xzvf jdk-7u79-linux-x64.tar.gz
```

(3). Configure the environment variables.

```
vim /etc/profile
```

Add the configuration of the following environment variables to the end of the file.

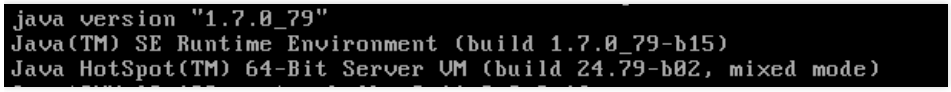
```
export JAVA_HOME=/usr/local/jdk/jdk1.7.0_79(JDK's decompressed directory)
export JRE_HOME=/usr/local/jdk/jdk1.7.0_79/jre
export PATH=$JAVA_HOME/bin:$JAVA_HOME/jre/bin:$PATH
export CLASSPATH=.:$JAVA_HOME/lib/tools.jar:$JAVA_HOME/lib/dt.jar:$JRE_HOME/lib
```

Save and exit wq, and then use the command 'source / etc / profile' to make the file take effect immediately.

(4). Verification
Verify whether the environment has been installed using the following command (javac command also applies), and then check whether the version numbers are consistent.

```
java -version
```

The following figure indicates the JDK installation is complete.



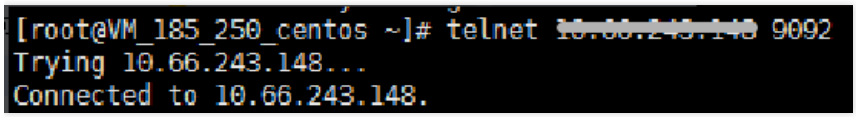
2.2 Download the Kafka toolkit

Download and decompress the Kafka installer package
CKafka is 100% compatible with Kafka 0.9. You're recommended to download this version of Kafka installer package from the official website http://kafka.apache.org/.

```
wget "http://mirrors.hust.edu.cn/apache/kafka/0.9.0.1/kafka_2.11-0.9.0.1.tgz"
tar -xzvf kafka_2.11-0.9.0.1.tgz
mv kafka_2.11-0.9.0.1 /opt/
```

The Kafka is ready for use immediately after download and decompression, without the need of configuring other environments. You can test whether this machine is connected to the CKafka instance with telnet command.

```
telnet ip 9092
```



2.3 Simple tests for Kafka APIs

Send messages

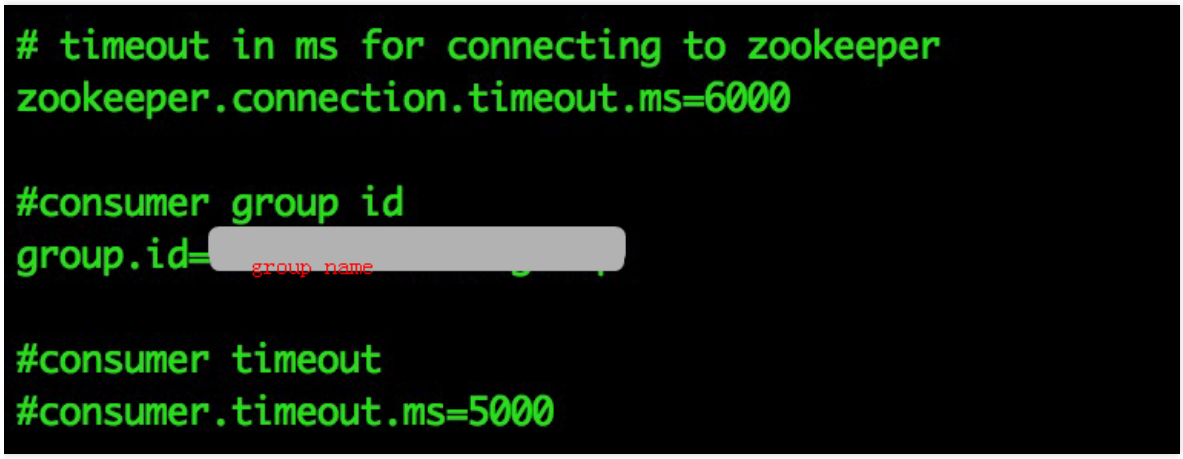
```
./kafka-console-producer.sh --broker-list xxx.xxx.xxx.xxx:9092 --topic topicName
This is a message
This is another message
```

> The IP in the broker-list is the vip in the CKafka instance, and topicName is the topic name in the CKafka instance.

Receive messages (Zookeeper cluster is hidden in CKafka by default)

```
./kafka-console-consumer.sh --bootstrap-server xxx.xxx.xxx.xxx:9092 --from-beginning --new-consumer --topic topicName
This is a message
This is another message
```

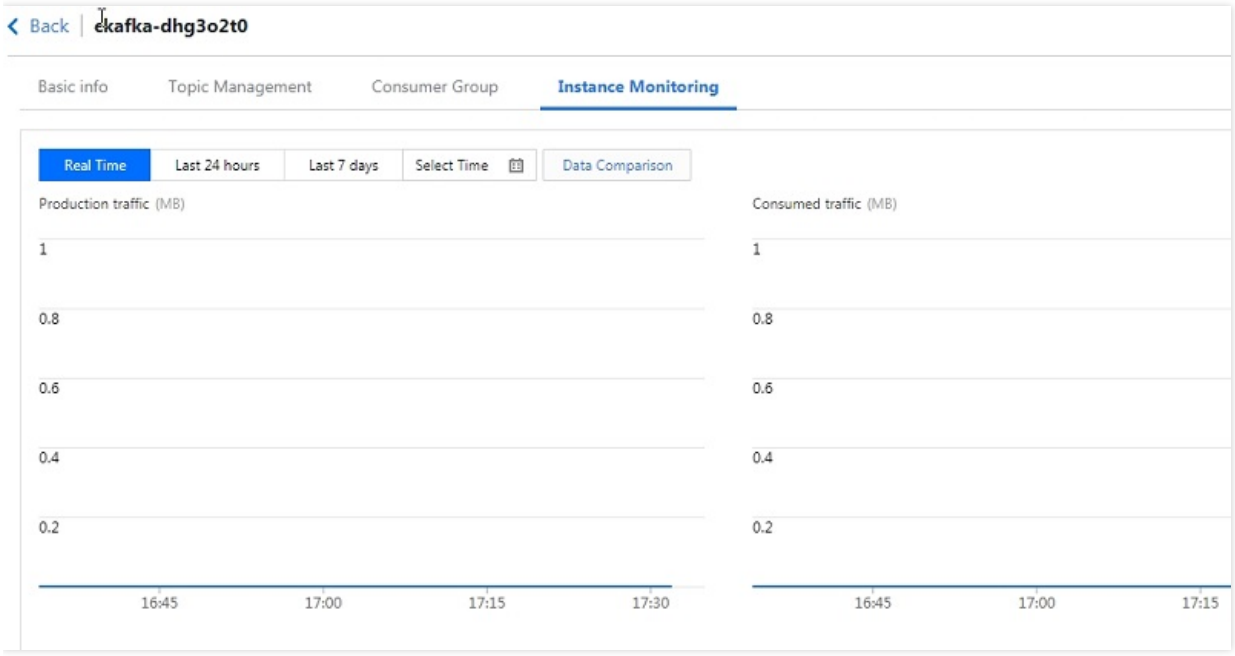
In the above commands, no consumer group is specified for consumption, so the system generates a group at random for consumption. In this way, it is very likely to reach the limit of group. Therefore, it is recommended to receive messages by **specifying a group**.
First, configure the specified group name in the consumer.properties, as shown below.



After the configuration, run the command that specifies the consumer group as shown below:

```
./kafka-console-consumer.sh --bootstrap-server xxx.xxx.xxx.xxx:9092 --from-beginning --new-consumer --topic topicName --consumer.config ../config/consumer.properties
```

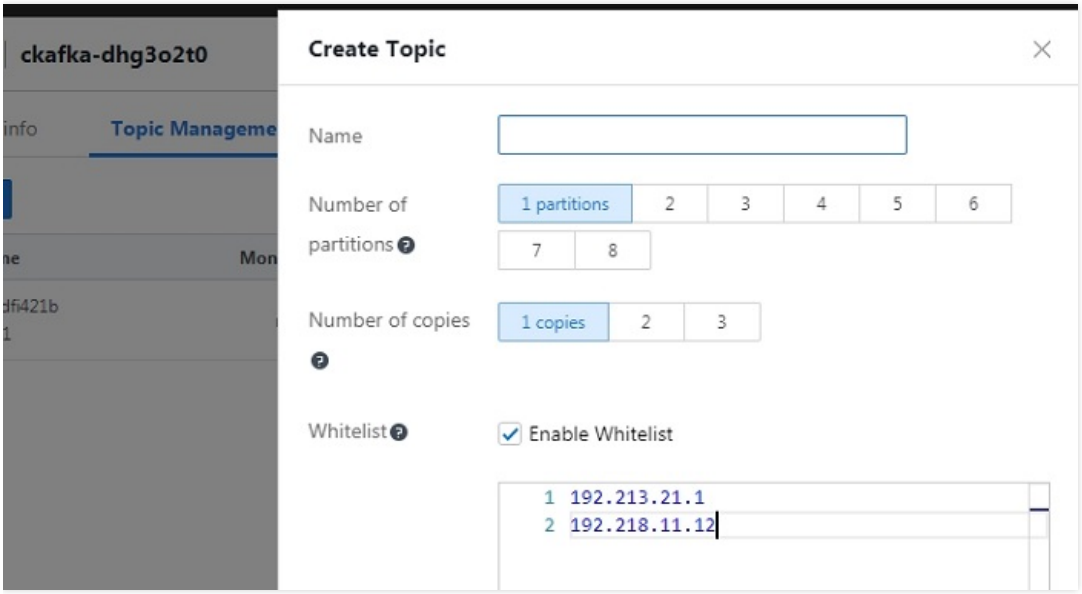
Check the CKafka monitor.



3. Other features

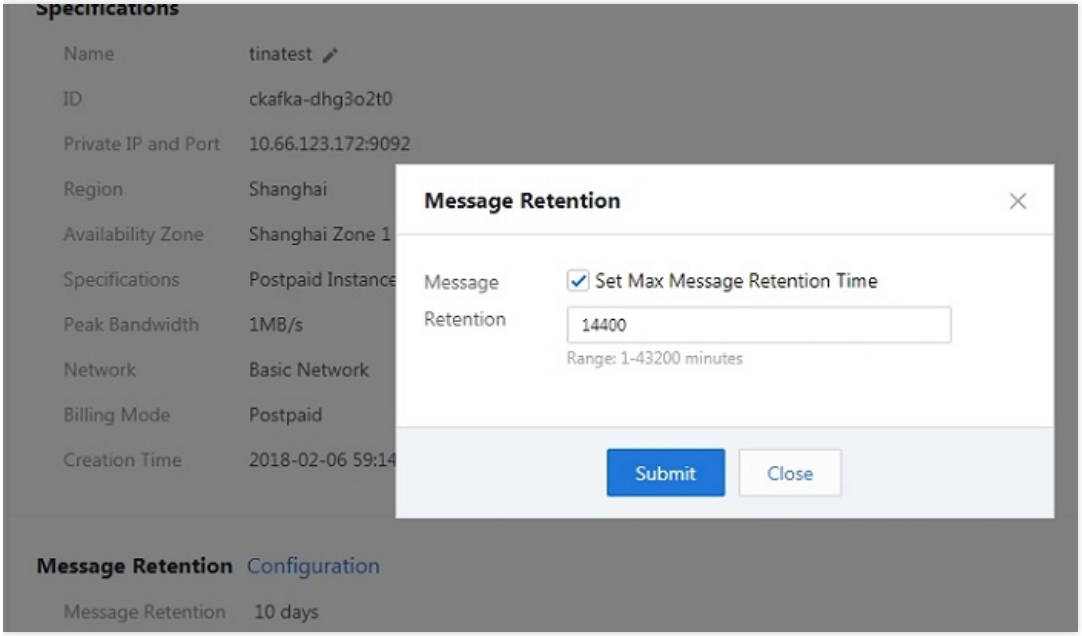
3.1 Enable the allowlist

CKafka supports enabling IP allowlist for a topic to ensure the data security.
You can enable the IP allowlist in both "New Topic" and "Edit Topic" pages.



3.2 Set message retention time

CKafka supports setting the message retention time (in minutes). The minimum is 1 minute, and the maximum is 30 days.



Compatibility with Open Source Kafka

Last updated : 2020-10-16 16:03:49

CKafka is compatible with the Producer and Consumer APIs of Apache Kafka 0.9 and later versions (v0.9, v0.10, v1.1.1, and v2.4.2 can be purchased). To connect to an on-premises Kafka of an earlier version (such as v0.8), partial rewriting of APIs is needed. This document compares the Producer and Consumer APIs of Kafka 0.8 and its later versions, and describes how to rewrite the APIs.

Kafka Producer

Overview

In Kafka 0.8.1, the Producer API is rewritten. This Producer client version is officially recommended because it provides better performance and more features. The community will maintain the new version of the Producer API (referred to as the New Producer API).

Comparison between new producer API and old producer API

- New Producer API Demo

```
Properties props = new Properties();
props.put("bootstrap.servers", "localhost:4242");
props.put("acks", "all");
props.put("retries",0);
props.put("batch.size", 16384);
props.put("linger.ms", 1);
props.put("buffer.memory", 33554432);
props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");
Producer<String, String> producer = new KafkaProducer<>(props);
producer.send(new ProducerRecord<String, String>("my-topic", Integer.toString(0), Integer.toString(0)));
producer.close();
```

- Old Producer API Demo

```
Properties props = new Properties();
props.put("metadata.broker.list", "broker1:9092");
props.put("serializer.class", "kafka.serializer.StringEncoder");
props.put("partitioner.class", "example.producer.SimplePartitioner");
props.put("request.required.acks", "1");
ProducerConfig config = new ProducerConfig(props);
Producer<String, String> producer = new Producer<String, String>(config);
KeyedMessage<String, String> data = new KeyedMessage<String, String>("page_visits", ip, msg);
producer.send(data);
producer.close();
```

As shown in the previous code, the basic usage of the new and old versions are the same, except for some parameter settings. This means the cost of API partial rewriting is not high.

Compatibility description

Producer API 0.8.x can be connected to CKafka successfully without partial rewriting. We recommend using the New Kafka Producer API.

Kafka Consumer

Overview

The open-source Apache Kafka 0.8 provides two types of the Consumer API:

- High Level Consumer API (blocking configuration details)
- Simple Consumer API (support for parameter configuration adjustment)

Kafka 0.9.x has introduced the New Consumer API that inherits the features of the two types of the old consumer API (v0.8) and reduces the load on ZooKeeper.
The following describes how to transform the old consumer API (v0.8) to the new consumer API (v0.9).

Comparison between new consumer API and old consumer API

Old consumer API (v0.8)

- **High Level Consumer API** (Demo)
  The High Level Consumer API can meet general requirements if you care only about data, except for message offset. This API, built on the consumer group logic, blocks the offset management, and supports Broker fault handling and Consumer load balancing. It allows developers to get started with the Consumer client quickly.
  Consider the following when you use the High Level Consumer API:
  - If the number of consumer threads is greater than the number of partitions, certain consumer threads cannot obtain data.
  - If the number of partitions is greater than the number of threads, certain threads consume more than one partition.
  - The changes in partitions and consumers will affect rebalancing.

- **Low Level Consumer API** (Demo)
  The Low Level Consumer API is recommended if you need message offset and features like repeated consumption or skip read, or if you want to consume specific partitions and ensure more consumption semantics. But in this case, you need to handle offsets and Broker exceptions.
  When using the Low Level Consumer API, you need to:
  - Track and maintain the offset and control the consumption progress.
  - Find the leader of partitions for the topic, and deal with partition changes.

New consumer API (v0.9)

Kafka 0.9.x has introduced the New Consumer API that inherits the features of the two types of Old Consumer API while providing consumer coordination (High Level API) and lower-level access to customize consumption policies. The New Consumer also simplifies the consumer client and introduces a central coordinator to solve the herd effect and split-brain problems resulting from the separate connections to ZooKeeper, and to reduce the load on ZooKeeper.

Advantages:

- Introduces coordinators
  The current version of High Level Consumer has the herd effect and split-brain problems. Placing failure detection and rebalancing logics into a highly available central coordinator solves both problems while greatly reducing the load on the ZooKeeper.
- Allows you to assign partitions
  To keep certain states of each local partition unchanged, you need to keep partition mappings unchanged. Some other scenarios are designed to associate the Consumer with the region-dependent Broker.
- Allows you to manage offsets
  You can manage offsets as needed to implement repeated consumption, skipped consumption, and other semantics.
- Triggers callbacks after rebalancing based on your specifications
- Provides non-blocking Consumer API

Comparison between new consumer API and old consumer API

Category	Introduced Version	Automatic Save of Offset	Self-management of Offset	Automatic Exception Handling	Automatic Rebalance Processing	Automatic Search of Leader	Advantage/Disadvantage
High Level Consumer	Before 0.9	Supported	Not supported	Supported	Supported	Supported	Herd effect and split-brain problems need to be handled.
Simple Consumer	Before 0.9	Not supported	Supported	Not supported	Not supported	Not supported	Multiple types of exception need to be handled.
New Consumer	After 0.9	Supported	Supported	Supported	Supported	Supported	Mature. This is the recommended version.

Transforming old consumer to new consumer

- New Consumer

```
//The main configuration difference is that the ZooKeeper parameters are replaced.
Properties props = new Properties();
props.put("bootstrap.servers", "localhost:9092");
props.put("group.id", "test");
props.put("enable.auto.commit", "true");
props.put("auto.commit.interval.ms", "1000");
props.put("session.timeout.ms", "30000");
props.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
props.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
//Compared with old consumers, new consumers are easier to create.
KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props);
consumer.subscribe(Arrays.asList("foo", "bar"));
while (true) {
ConsumerRecords<String, String> records = consumer.poll(100);
for (ConsumerRecord<String, String> record : records)
System.out.printf("offset = %d, key = %s, value = %s", record.offset(), record.key(), record.value());}
```

- Old Consumer (High Level)

```
//Old consumers require ZooKeeper
Properties props = new Properties();
props.put("zookeeper.connect", "localhsot:2181");
props.put("group.id", "test");
props.put("auto.commit.enable", "true");
props.put("auto.commit.interval.ms", "1000");
props.put("auto.offset.reset", "smallest");
ConsumerConfig config = new ConsumerConfig(props);
//Require connector creation
ConsumerConnector connector = Consumer.createJavaConsumerConnector(config);
//Create a message stream
Map<String, Integer> topicCountMap = new HashMap<String, Integer>();
topicCountMap.put("foo", 1);
Map<String, List<KafkaStream<byte[], byte[]>>> streams =
connector.createMessageStreams(topicCountMap);
//Obtain data
KafkaStream<byte[], byte[]> stream = streams.get("foo").get(0);
ConsumerIterator<byte[], byte[]> iterator = stream.iterator();
MessageAndMetadata<byte[], byte[]> msg = null;
while (iterator.hasNext()) {
msg = iterator.next();
```

```
System.out.println(//
" group " + props.get("group.id") + //
", partition " + msg.partition() + ", " + //
new String(msg.message()));
}
```

Comparing with the old consumer, the new consumer has simpler coding and uses Kafka addresses instead of ZooKeeper parameters. In addition, the new consumer has parameter settings for interactions with coordinators, in which the default settings are suitable for general use.

Compatibility description

Both CKafka and the new version of Kafka in the open-source community support the rewritten new consumer API, which blocks the interaction between the consumer client and ZooKeeper (ZooKeeper is not exposed to users any longer). The new consumer solves the herd effect and split-brain problems resulting from direct interaction with ZooKeeper, and integrates the features of the old consumer, thus making the consumption more reliable.