Message Queue CKafka

Quick Start

Product Introduction

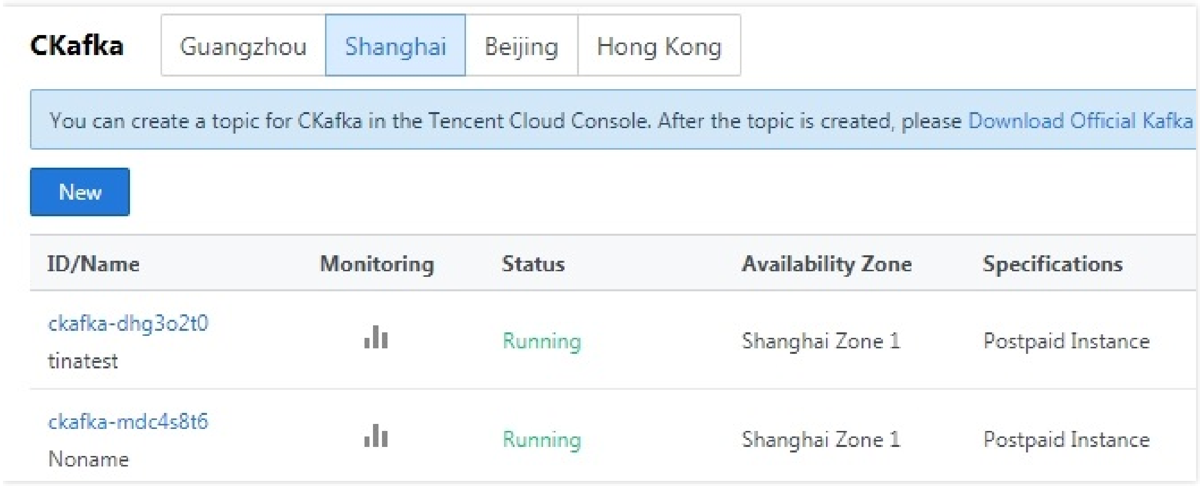
Contents

Quick Start
Getting Started

Last updated：2018-09-12 10:56:24

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.

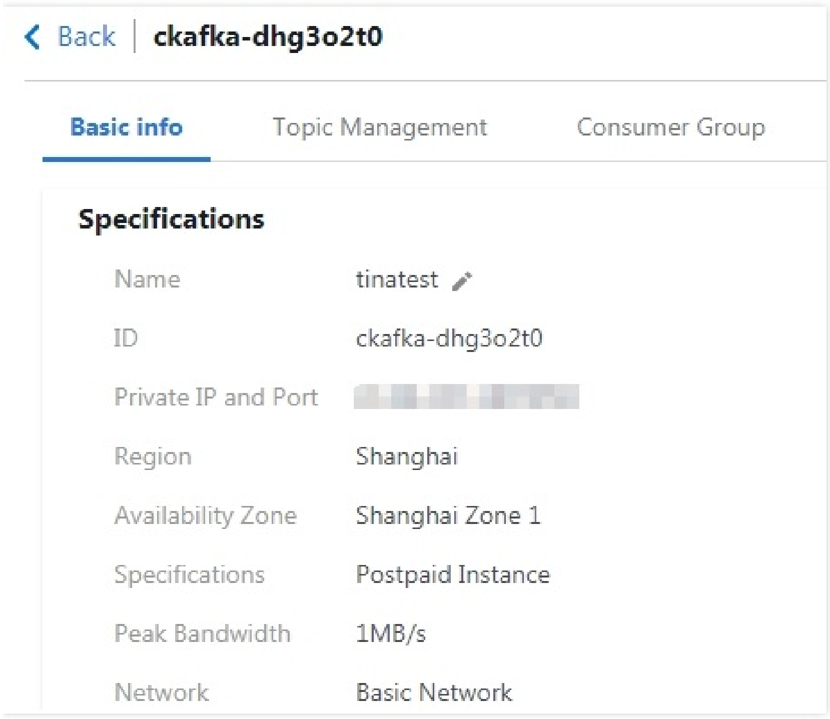


CKafka | Guangzhou | Shanghai | Beijing | Hong Kong

You can create a topic for CKafka in the Tencent Cloud Console. After the topic is created, please Download Official Kafka

New

ID/Name	Monitoring	Status	Availability Zone	Specifications
ckafka-dhg3o2t0 tinatest		Running	Shanghai Zone 1	Postpaid Instance
ckafka-mdc4s8t6 Noname		Running	Shanghai Zone 1	Postpaid Instance



Back | ckafka-dhg3o2t0

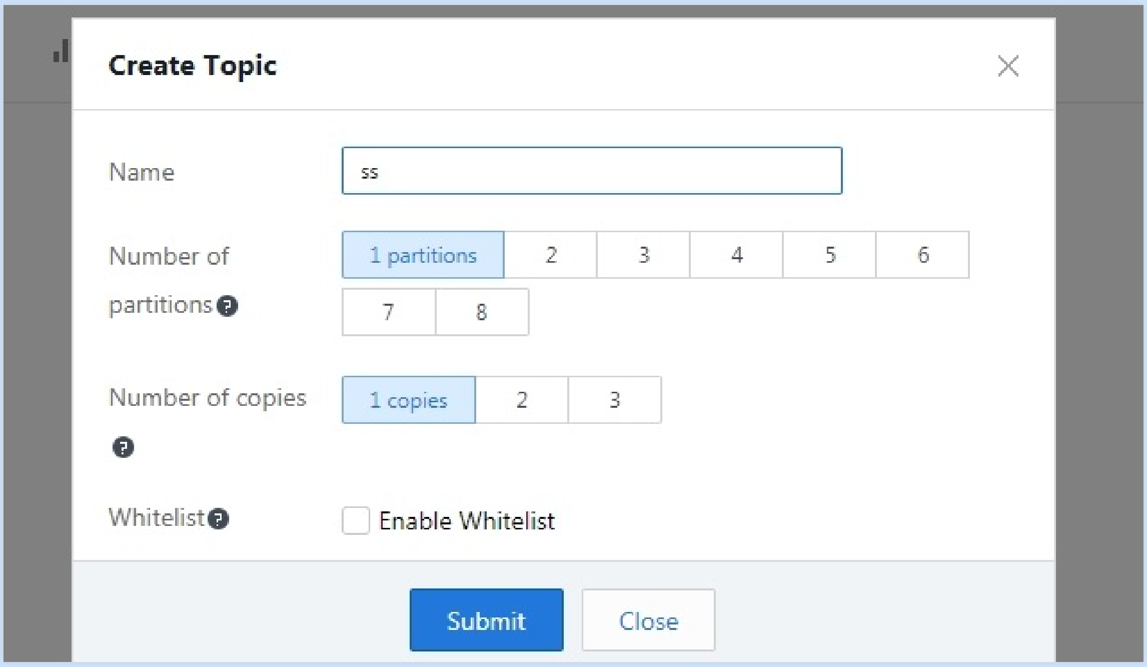
Basic info | Topic Management | Consumer Group

Specifications

Name	tinatest
ID	ckafka-dhg3o2t0
Private IP and Port	
Region	Shanghai
Availability Zone	Shanghai Zone 1
Specifications	Postpaid Instance
Peak Bandwidth	1MB/s
Network	Basic Network

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.


Create Topic

Name: ss

Number of partitions: [✓ 1 partitions] 2 3 4 5 6 7 8

Number of copies: [✓ 1 copies] 2 3

Whitelist: ☐ Enable Whitelist

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.

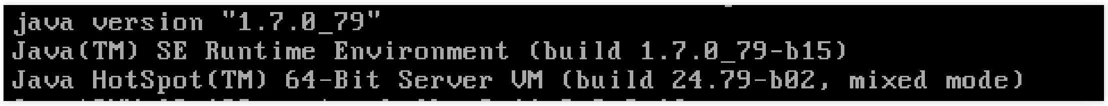
```
export JAVA_HOME=/usr/local/jdk/jdk1.7.0_79(JDK's decompressed directory)
export JRE_HOME=/usr/local/jdk/jdk1.7.0_79/jre
export PATH=$JAVA_HOME/bin:$JAVA_HOME/jre/bin:$PATH
export CLASSPATH=.:$JAVA_HOME/lib/tools.jar:$JAVA_HOME/lib/dt.jar:$JRE_HOME/lib
```

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.



```
java version "1.7.0_79"
Java(TM) SE Runtime Environment (build 1.7.0_79-b15)
Java HotSpot(TM) 64-Bit Server VM (build 24.79-b02, mixed mode)
```

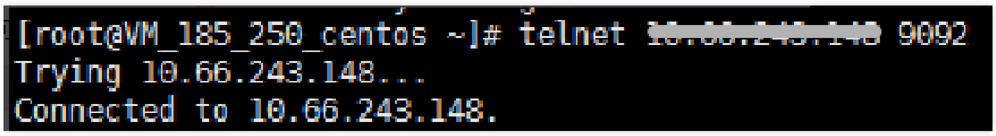
2.2 Download the Kafka toolkit

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
```



```
[root@VM_185_250_centos ~]# telnet [redacted] 9092
Trying 10.66.243.148...
Connected to 10.66.243.148.
```

2.3 Simple tests for Kafka APIs

Send messages

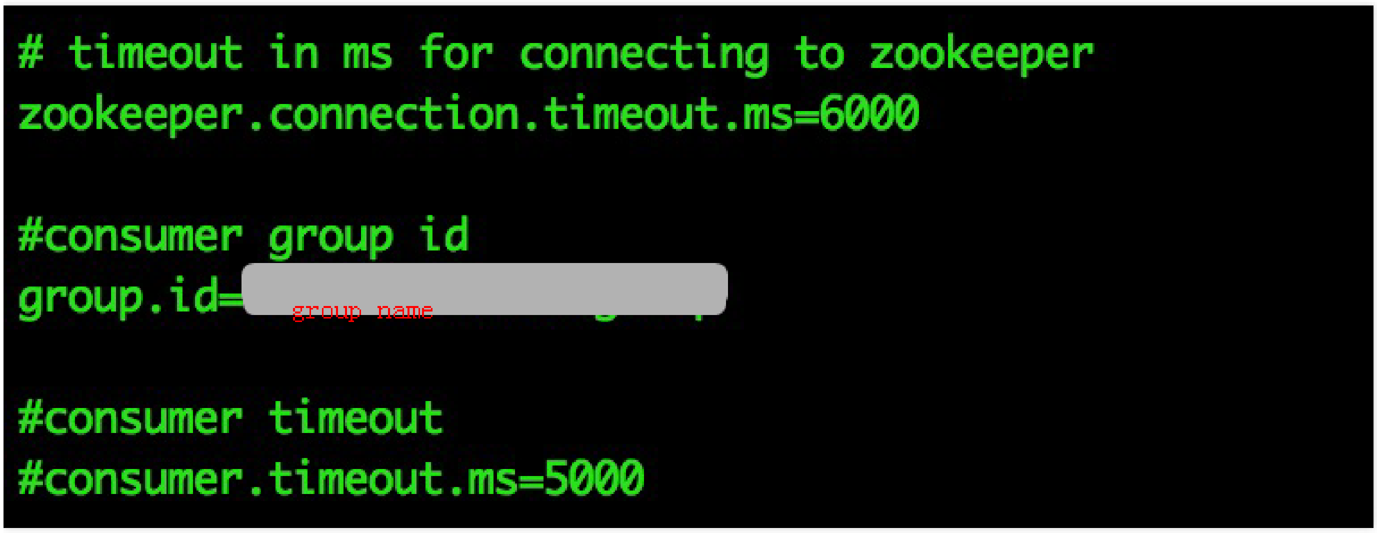
```
./kafka-console-producer.sh --broker-list xxx.xxx.xxx.xxx:9092 --topic topicName
This is a message
This is another message
```

> 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 to
picName
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.

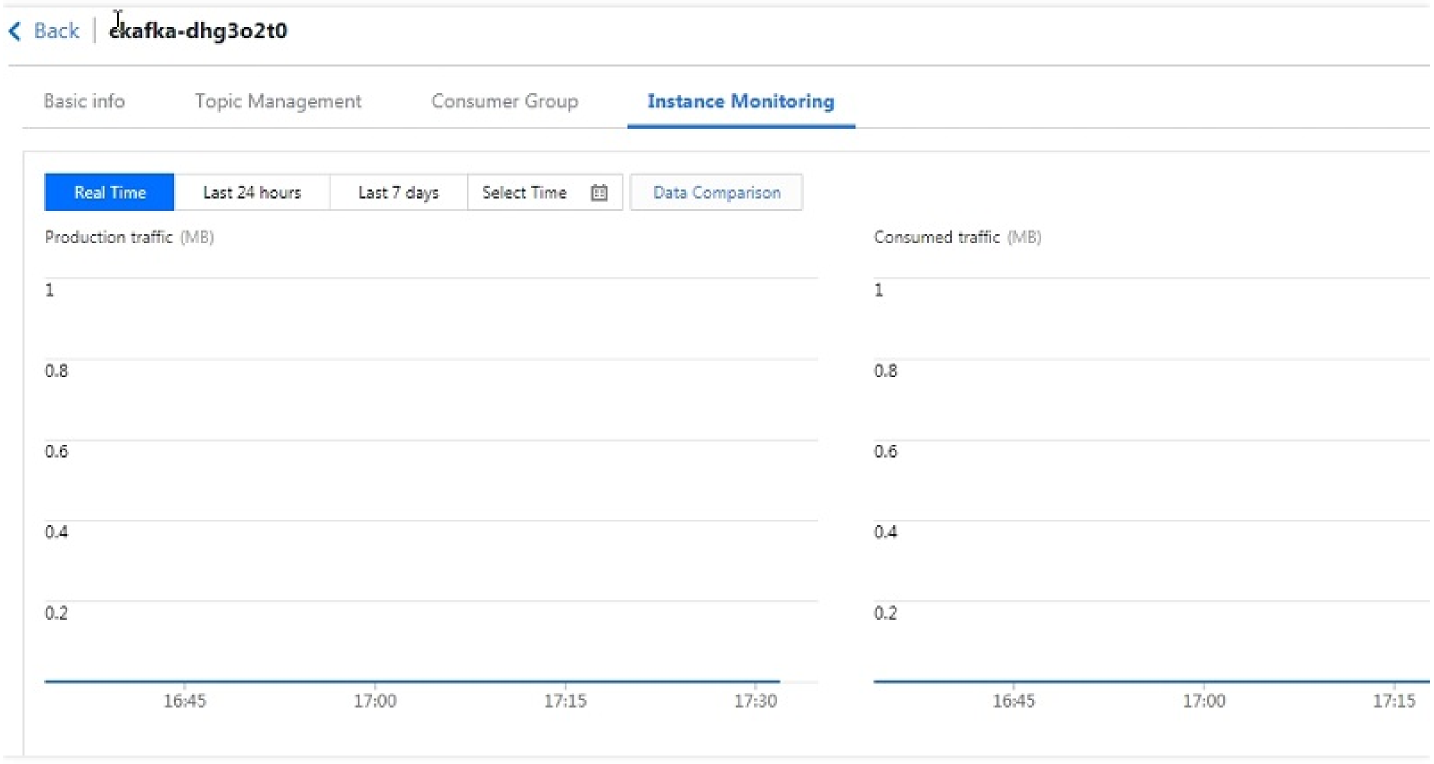


```
# timeout in ms for connecting to zookeeper
zookeeper.connection.timeout.ms=6000

#consumer group id
group.id=[redacted]

#consumer timeout
#consumer.timeout.ms=5000
```

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 to
picName --consumer.config ../config/consumer.properties
```

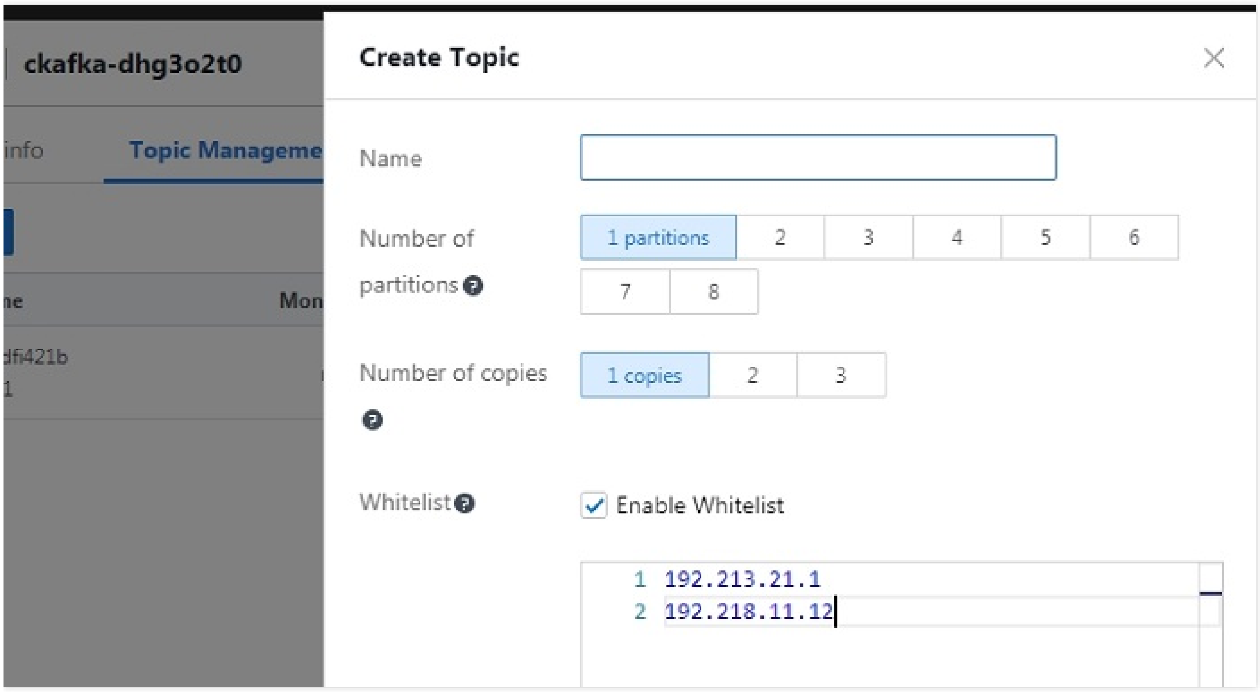
Check the CKafka monitor.



3. Other features

3.1 Enable the whitelist

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



Name

Number of partitions: [✓ 1 partitions] 2 3 4 5 6 7 8

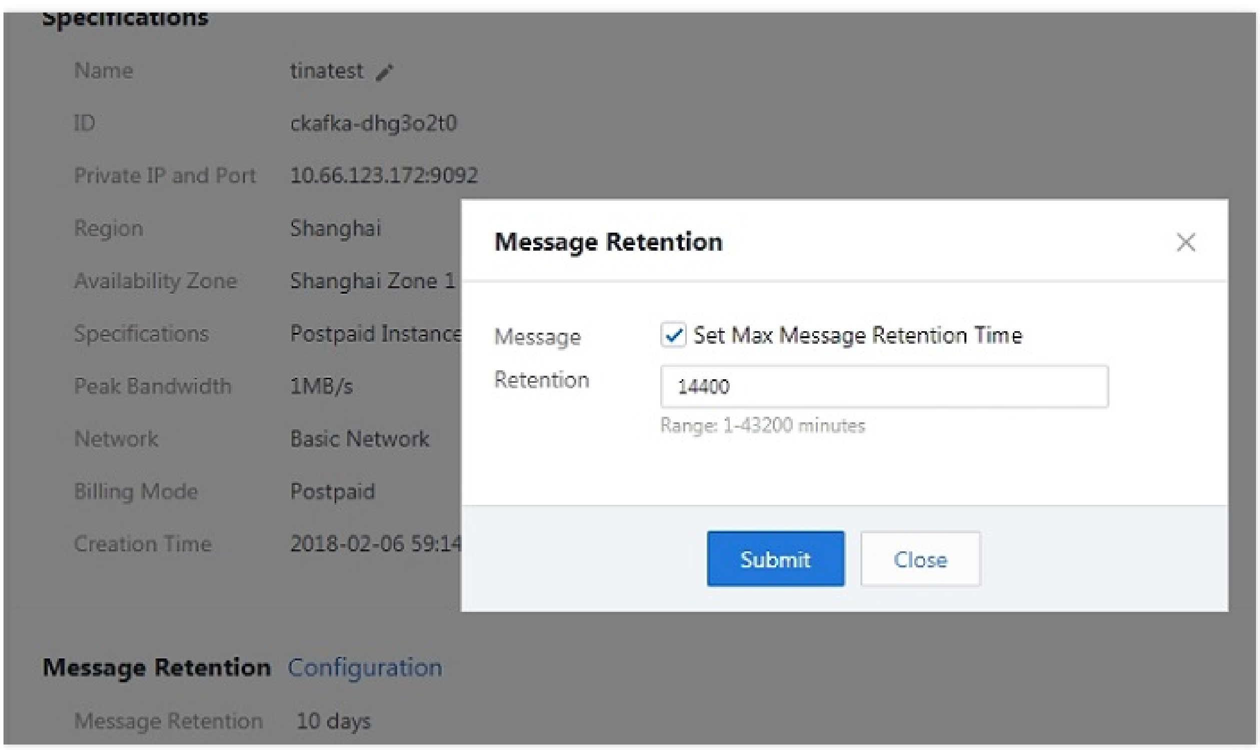
Number of copies: [✓ 1 copies] 2 3

Whitelist: ✓ Enable Whitelist

```
192.213.21.1
192.218.11.12
```

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.


Specifications

Name	tinatest
ID	ckafka-dhg3o2t0
Private IP and Port	10.66.123.172:9092
Region	Shanghai
Availability Zone	Shanghai Zone 1
Specifications	Postpaid Instance
Peak Bandwidth	1MB/s
Network	Basic Network
Billing Mode	Postpaid
Creation Time	2018-02-06 59:14

Message Retention

Message Retention: ✓ Set Max Message Retention Time

14400

Range: 1-43200 minutes

Message Retention Configuration

Message Retention 10 days

Compatibility with Open Source Kafka

Last updated：2018-01-30 17:05:30

Ckafka was initially designed to be compatible with version 0.9.x of Producer/Consumer APIs. With the 0.10.x-compatible versions developed, Ckafka now has the compatibility with 0.9.x and 0.10.x versions of producer/consumer APIs. As the Zookeeper address is unavailable, the High Level Consumer API requiring Zookeeper address is not supported.

Kafka Producer Type

Changes to Producer

In Kafka 0.8.1, the Producer API is overwritten. This version of client is officially recommended for its better performance and more features. The community will maintain a new version of the Producer API.

Transformation of Producer

(1) DEMO on how the new API is written

```
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();
```

(2) DEMO on how the old API is written

```
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 above, there is not much difference between the two. The basic usage remains the same, except for the changes in the configuration of some parameters. Therefore, the transformation cost is not high.

Ckafka Compatibility

Both the new and old 0.8.x Producer APIs can be interfaced with Ckafka smoothly without modification. We recommend using the new Kafka Producer API, as the community does, for more configurations and features.

Kafka Consumer Type

Old Consumer

- High Level Consumer API
  The High Level API can meet the general consumption requirements if only data is needed and you don't need to consider the processing of message offset. The High-Level Consumer API, built on the logic of Consumer Group, blocks the Offset management and features the capability of Broker exception handling and Consumer load balancing, allowing developers to get started with the Consumer client quickly.
  The following need to be considered when you use the High Level Consumer:
  (1) If the number of consumer threads is greater than the number of Partitions, some consumer threads cannot get the data.
  (2) If the number of Partitions is greater than the number of threads, some threads consume more than one Partitions.
  (3) The changes in Partitions and consumers can affect the Rebalance.

DEMO

- Low Level Consumer API
  Low Level Consumer API is recommended if you care about the message offset and need such features as repeated consumption or skip read, or want to specify certain partitions for consumption and ensure more consumption semantics. But in this case, you need to handle the exceptions of Offset and Broker by yourself.
  The following need to be considered when you use the Low Level Consumer:
  (1) You need to track and maintain the Offset and control the consumption progress by yourself.
  (2) You need to find the Leader of Partitions for the Topic, and deal with the Partition changes.

DEMO

- New Consumer (After 0.9.x)
  Why use the New Consumer
  Kafka 0.9.x has introduced the New Consumer, which incorporates the features of Old Consumer while providing consumer coordination (advanced API) and lower-level access to build the custom 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:
  (1) Introduction of Coordinator
  The current version of High Level Consumer has the Herd Effect and Split Brain problems .Putting the logics of failure detection and Rebalance into a highly available central Coordinator can solve both of the problems, while greatly reducing the load on the Zookeeper.
  (2) Allow you to assign partitions by yourself.

In order to keep some statuses of each local Partition the same, you need to maintain the mapping of Partitions. Some other scenarios are designed to associate the Consumer with the region-dependent Broker.
(3) Allow you to manage Offset
You can manage the Offset as needed and implement repeated consumption, skipped consumption and other semantics.
(4) Trigger user-specified callbacks after Rebalance
(5) Non-blocking Consumer API

Comparison between New Consumer and Old Consumer

Category	Introduced Version	AutoSave of Offset	Self-management of Offset	Automatic Exception Handling	Automatic Rebalance Processing	Auto Search of Leader	Disadvantages
High Level Consumer	Before 0.9	Yes	No	Yes	Yes	Yes	Herd Effect and Split Brain
Simple Consumer	Before 0.9	No	Yes	No	No	No	Multiple exceptions need to be handled
New Consumer	After 0.9	Yes	Yes	Yes	Yes	Yes	Mature. The current version is recommended

Old Consumer is transformed to New Consumer

1. New Consumer

```
//The main change in config is that the Zookeeper parameter is 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 to old Consumer, the new Consumer makes it easier to create consumers
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());}
```

1. Old Consumer (High Level)

```
// Zookeeper is needed
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);
// Creation of connector is needed
ConsumerConnector connector = Consumer.createJavaConsumerConnector(config);
// Create message stream
Map<String, Integer> topicCountMap = new HashMap<String, Integer>();
topicCountMap.put("foo", 1);
Map<String, List<KafkaStream<byte[], byte[]>>> streams =
connector.createMessageStreams(topicCountMap);
// Get 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()));
}}
```

As shown above, transforming into New Consumer simplifies the writing. The main change is the input of Zookeeper parameters is replaced by the input of Kafka address. In addition, the configuration of parameters interacting with Coordinator are added. Generally, it is sufficient to use the default configuration.

Recommended Ckafka version

Both Ckafka and the new version of Kafka in the community support the New Consumer, 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 the direct interaction with Zookeeper, and integrates the features of Old Consumer, thus making the consumption more reliable.