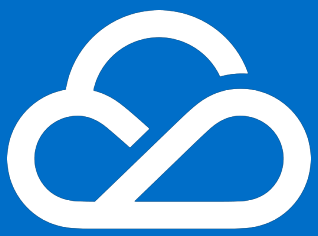
腾讯云消息队列 CKafka

快速入门

产品文档



腾讯云

【版权声明】

【商标声明】

【服务声明】

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

文档目录

快速入门

CKafka使用入门

1. 查看CKafka实例，创建topic

申请通过后，您的CKafka控制台中会展示CKafka实例，点击实例信息可以查看实例详情。包括地域，内网vip，端口等信息。

在topic管理页面，您可以创建topic，指定topic的分区个数和副本个数。

> 注，当前topic名称输入后无法更改。此外，分区个数指定后，只能进行新增分区的操作。副本数指定后无法更改。

创建好topic和分区后，可以通过云主机的kafka客户端对该实例进行生产和消费的操作。

2. 本地下载Kafka工具包

2.1 安装JDK环境

本教程在腾讯云主机上搭建CKafka环境，首先可以在购买页选购云主机，并登入。本次测试机器配置如下：

机器配置
操作系统 CentOS 6.8 64位
CPU 1核
内存 2GB
公网带宽 1Mbps

之后需要给云主机安装JDK。

a 下载JDK，可以通过wget命令获取，如果需要其他不同版本也可以在官网进行下载。

建议使用1.7以上版本的JDK，本教程的版本为jdk1.7.0_79

b 移动到固定文件夹并解压缩：

```
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
```

c 配置环境变量

```
vim /etc/profile
```

在文件末尾加入如下环境变量的配置

```
export JAVA_HOME=/usr/local/jdk/jdk1.7.0_79(JDK的解压目录)
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
```

wq保存退出后，使用

```
source /etc/profile
```

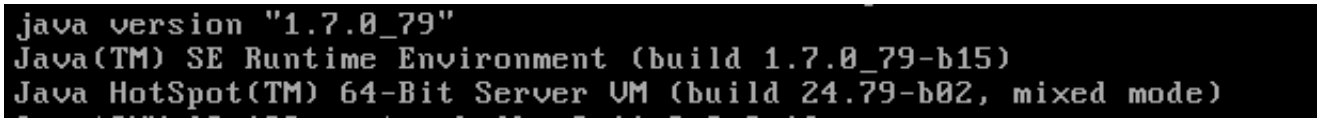
命令使文件立即生效

d 验证

通过以下命令验证环境是否安装完成（javac命令也可以），查看版本号是否一致

```
java -version
```

出现下图则证明jdk安装完成。



2.2 下载Kafka工具包

下载并解压kafka安装包

Kafka官网地址: http://kafka.apache.org/ 当前CKafka 100%兼容Kafka

0.9版本，建议您下载相应版本的安装包

wget "http://mirrors.hust.edu.cn/apache/kafka/0.9.0.1/kafka_2.11-0.9.0.1.tgz"

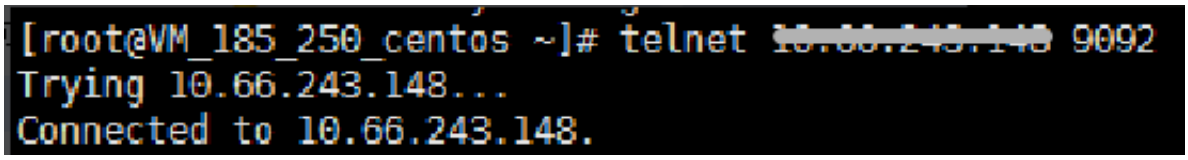
tar -xzvf kafka_2.11-0.9.0.1.tgz

mv kafka_2.11-0.9.0.1 /opt/

下载解压完成后，无需配置其他环境，直接可用

可以通过telnet指令测试本机是否连通到CKafka实例

telnet ip 9092



2.3 Kafka API简单测试

发送消息

./kafka-console-producer.sh --broker-list xxx.xxx.xxx.xxx:9092 --topic topicName

This is a message

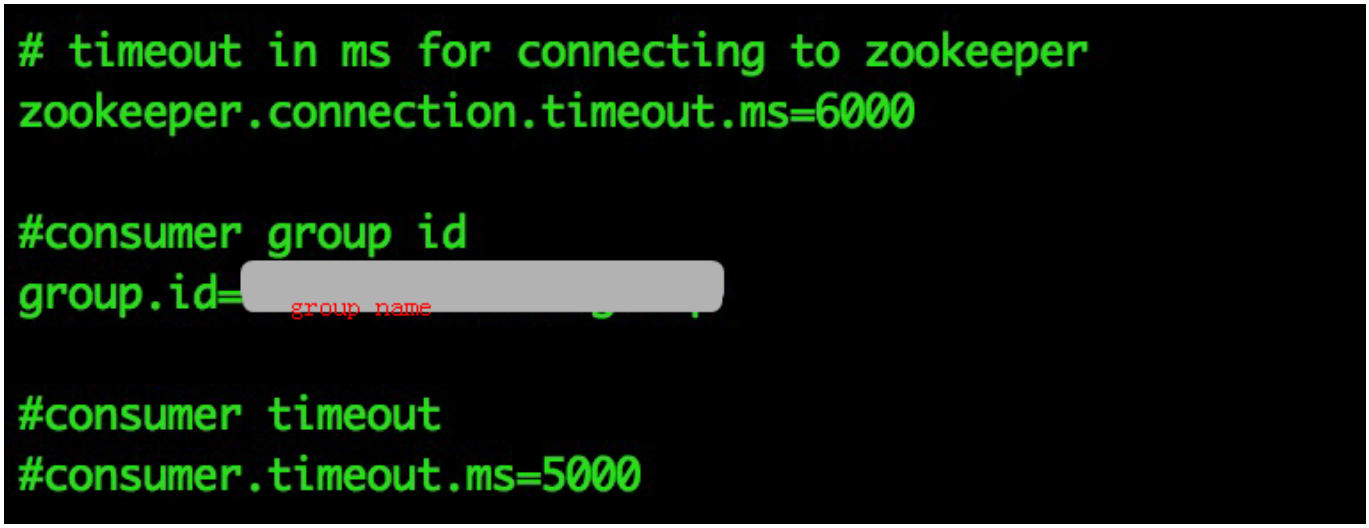
This is another message

其中broker-list中的ip即为CKafka实例中的vip，topicName为CKafka实例中的topic名称

接收消息(CKafka默认隐藏Zookeeper集群)

./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

上述命令中，由于没有指定consumer
group进行消
费，系统会随机生成一个gro
up进行消费。这样做容易达到group上限。因此推荐指定Group的方式接收消息，
首先需要在consumer.properties中配置下指定的group name，如下图所示



配置完成后，指定consumer group的命令如下所示：

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

查看对应的CKafka监控

3. 其他功能

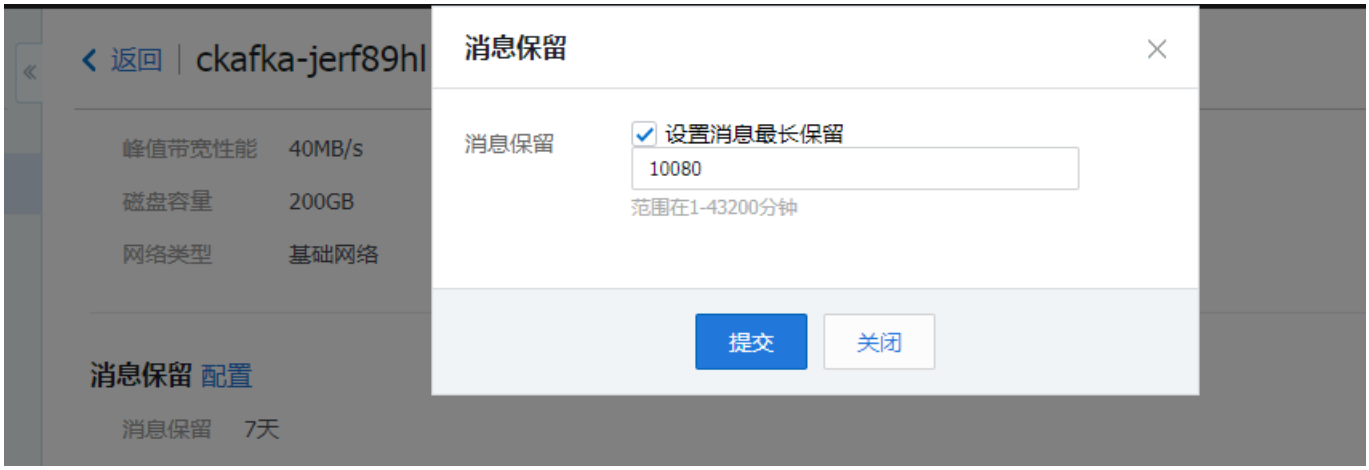
3.1 开启白名单

CKafka支持在topic维度开启ip白名单的功能，有效保证数据安全。

在新建topic和编辑topic页面均可以开启ip白名单。

3.2 设置消息保留时间

CKafka支持设置消息保留时间，以分钟为单位，最短1分钟，最长保留30天。



CKafka与开源Kafka兼容性说明

Ckafka起初以兼容0.9.x版本为目标，后续开发了0.10.x兼容版本。Ckafka兼容0.9系列以及0.10系列的生产/消费接口。但是现在暂不开放Zookeeper地址，所以对于需要Zookeeper地址的High Level Consumer API暂不提供支持。

Kafka Producer Type

Producer变化

Kafka 0.8.1版本中，Producer API被重写。该客户端被官方推荐，其拥有更好的性能以及更多的功能，后续社区将维护新版本的Producer API。

Producer改造

1）新API写法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();
```

2）旧API写法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();
```

可以看出两者区别不大，基本使用方法保持一致只是一些参数配置的变化，改造代价不大。

Ckafka兼容性说明

对于Ckafka而言，0.8.x的 新/旧 的Producer
API都可以顺利接入Ckafka无需改造，我们推荐与社区一样使用新的Kafka Producer
Api，其拥有更多配置，功能更加完善。

Kafka Consumer Type

Old Consumer

- High Level Consumer API
  如果只需要数据而不需要考虑消息offset相关的处理时，High Level API就满足一般性消费要求，High Level Consumer API围绕着Consumer Group这个逻辑概念展开，它屏蔽Offset管理、具有Broker异常处理、Consumer负载均衡功能。使开发者可以快速的上手Consumer客户端。
  在使用High Level Consumer时需要注意以下几点：
  1）如果消费线程大于Partition个数，意味着某些消费线程将无法获得数据
  2）如果Partition个数大于线程数目，某些线程会消费多个Partition
  3）Partition和消费者变动会影响Rebalance

参考DEMO

- Low Level Consumer API

  如果使用者关心消息的offset并且希望进行重复消费或者跳读等功能、又或者希望指定某些partition进行消费时和确保更多消费语义时推荐使用Low Level Consumer API。但是使用者需要自己处理Offset以及Broker的异常情况。

  在使用Low Level Consumer时需要注意以下几点：

  1）自行跟踪维护Offset，控制消费进度

  2）查找Topic相应Partition的Leader，以及处理Partition变更情况

参考DEMO

- New Consumer （After 0.9.x）

  为什么使用 New Consumer

  Kafka 0.9.x 版本引入了 New Consumer，其融合了Old Consumer两种Consumer API的特性,同时提供消费者的协调(高级API)和lower-level的访问,来构建自定义的消费策略。New Consumer还简化了消费者客户端并且引入了中心Coordinator解决分别连接Zookeeper产生的 Herd Effect和Split Brain 问题同时还减轻了Zookeeper的负载。

  优势：

  1）Coordinator引入

  当前版本的High Level Consumer存在Herd Effect和Split Brain的问题。将失败探测和Rebalance的逻辑放到一个高可用的中心Coordinator，那么这两个问题即可解决。同时还可大大减少Zookeeper的负载。

  2）允许自己分配Partition

  为了保持本地每个分区的一些状态不变，所以需要将Partition的映射也保持不变。另外一些场景是为了让Consumer与地域相关的Broker关联。

  3）允许自己管理Offset

  可以根据自己需要去管理Offset，实现重复、跳跃消费等语意。

  4）Rebalance后触发用户指定的回调

  5）非阻塞式Consumer API

New Consumer 对比 Old Consumer

种类	引入版本	Offset自动保存	Offset自行管理	自动进行异常处理	Rebalance自动处理	Leader自动查找	缺点
High Level	Before 0.9	支持	不支持	支持	支持	支持	Herd

种类	引入版本	Offset自动保存	Offset自行管理	自动进行异常处理	Rebalance自动处理	Leader自动查找	缺点
Consumer							Effect和Split Brain
Simple Consumer	Before 0.9	不支持	支持	不支持	不支持	不支持	需要处理多种异常情况
New Consumer	After 0.9	支持	支持	支持	支持	支持	成熟，当前版本推荐

Old Consumer 转成 New Consumer

1. New Consumer

```
//config中主要变化是 zookeeper参数被替换了
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");
// 相比old consumer 而言，这里创建消费者更加简单
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 ）

```
// old consumer 需要 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);
// 需要创建connector
ConsumerConnector connector = Consumer.createJavaConsumerConnector(config);
// 创建message stream
Map<String, Integer> topicCountMap = new HashMap<String, Integer>();
topicCountMap.put("foo", 1);
Map<String, List<KafkaStream<byte[], byte[]>>> streams =
 connector.createMessageStreams(topicCountMap);
// 获取数据
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()));
}}
```

对比可以看到，改造成New Consumer编写更加简单，最主要的变化是隐藏了Zookeeper的参数的输入替代成了Kafka地址输入。同时，New
Consumer也增加了与Coodinator交互的参数配置，一般情况下使用默认配置就足够。

Ckafka版本推荐

Ckafka与社区新版本Kafka一致支持New Consumer方式，屏蔽了Consumer客户端与Zookeeper的交互（Zookeeper不再向用户暴露）。使用New Consumer解决原有与Zookeeper直接交互的 Herd Effect和Split Brain 问题，以及融合了原有Old Consumer的特性，使消费环节更加可靠。