kafka集群搭建（docker-compose）和使用

笔记本:	kafka		
创建时间:	2020/6/7 18:46	**更新时间:**	2020/6/7 19:01
作者:	jin.zhou@definesys.com		

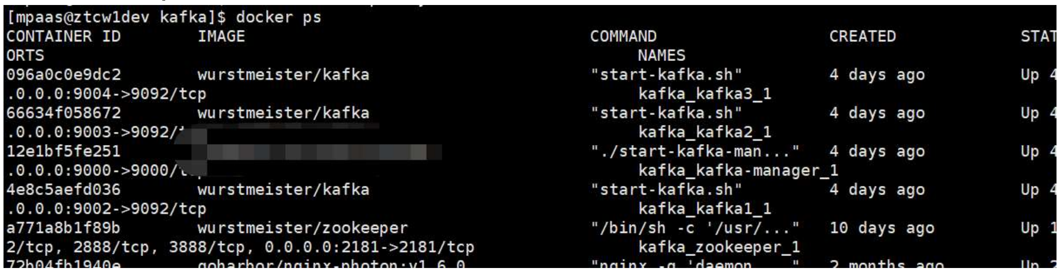
1、事先需要在机器上面安装docker，docker-compose环境

2、编写docker-compose.yml文件

下面的代码里面安装了一个zookeeper和3节点的kafka集群，把这个yml文件创建好了之后我们在当前的目录下面直接执行命令
docker-compose up -d 启动我们的集群

```
version: '3'
services:
  zookeeper:
    image: wurstmeister/zookeeper
    ports:
      - "2181:2181"
  kafka1:
    image: wurstmeister/kafka
    links:
      - zookeeper
    ports:
      - "9002:9092"
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
      KAFKA_AUTO_CREATE_TOPICS_ENABLE: "false"
      KAFKA_ADVERTISED_HOST_NAME: 172.16.161.51
      KAFKA_ADVERTISED_PORT: 9002
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
  kafka2:
    image: wurstmeister/kafka
    links:
      - kafka1
    ports:
      - "9003:9092"             # kafka 把9092端口随机映射到主机的端口
    environment:
      KAFKA_ADVERTISED_HOST_NAME: 172.16.161.51           #本机ip
      KAFKA_ADVERTISED_PORT: 9003
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_BROKER_ID: 2
      KAFKA_AUTO_CREATE_TOPICS_ENABLE: "false"
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
  kafka3:
    image: wurstmeister/kafka
    links:
      - kafka2
    ports:
      - "9004:9092"             # kafka 把9092端口随机映射到主机的端口
    environment:
      KAFKA_BROKER_ID: 3
      KAFKA_AUTO_CREATE_TOPICS_ENABLE: "false"
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
      KAFKA_ADVERTISED_HOST_NAME: 172.16.161.51           #本机ip
      KAFKA_ADVERTISED_PORT: 9004
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
```

使用docker ps 可以看到集群已经启动成功



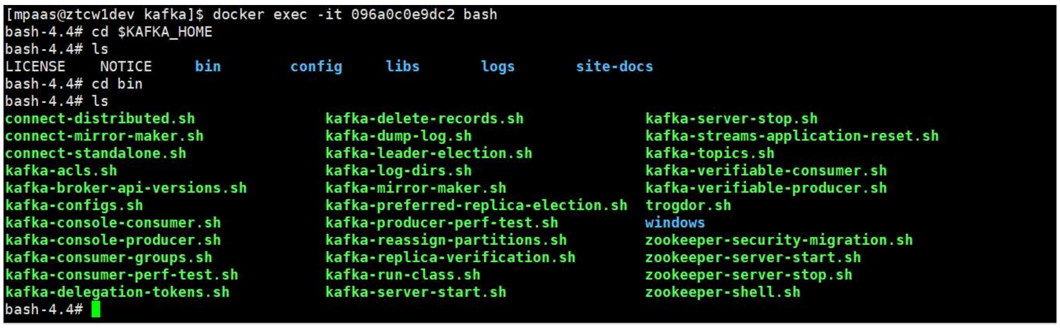
```
[mpaas@ztcw1dev kafka]$ docker ps
CONTAINER ID        IMAGE                                  COMMAND                  CREATED             STAT
ORTS                                                       NAMES
096a0c0e9dc2        wurstmeister/kafka                     "start-kafka.sh"         4 days ago          Up 4
.0.0.0:9004->9092/tcp                                          kafka_kafka3_1
66634f058672        wurstmeister/kafka                     "start-kafka.sh"         4 days ago          Up 4
.0.0.0:9003->9092/[redacted]                                   kafka_kafka2_1
12e1bf5fe251                                               "./start-kafka-man..."   4 days ago          Up 4
.0.0.0:9000->9000/                                             kafka_kafka-manager_1
4e8c5aefd036        wurstmeister/kafka                     "start-kafka.sh"         4 days ago          Up 4
.0.0.0:9002->9092/tcp                                          kafka_kafka1_1
a771a8b1f89b        wurstmeister/zookeeper                 "/bin/sh -c '/usr/..."   10 days ago         Up 1
2/tcp, 2888/tcp, 3888/tcp, 0.0.0.0:2181->2181/tcp              kafka_zookeeper_1
72b04fb1940e        goharbor/nginx-photon:v1.6.0           "nginx -g 'daemon ..."   2 months ago        Up 2
```

使用命令创建主题

创建好了集群之后我们可以尝试着在kafka里面创建一个topic

1、进入容器内部

使用命令 `docker exec -it 容器id bash`



```
[mpaas@ztcw1dev kafka]$ docker exec -it 096a0c0e9dc2 bash
bash-4.4# cd $KAFKA_HOME
bash-4.4# ls
LICENSE    NOTICE     bin        config     libs       logs       site-docs
bash-4.4# cd bin
bash-4.4# ls
connect-distributed.sh             kafka-delete-records.sh              kafka-server-stop.sh
connect-mirror-maker.sh            kafka-dump-log.sh                    kafka-streams-application-reset.sh
connect-standalone.sh              kafka-leader-election.sh             kafka-topics.sh
kafka-acls.sh                      kafka-log-dirs.sh                    kafka-verifiable-consumer.sh
kafka-broker-api-versions.sh       kafka-mirror-maker.sh                kafka-verifiable-producer.sh
kafka-configs.sh                   kafka-preferred-replica-election.sh  trogdor.sh
kafka-console-consumer.sh          kafka-producer-perf-test.sh          windows
kafka-console-producer.sh          kafka-reassign-partitions.sh         zookeeper-security-migration.sh
kafka-consumer-groups.sh           kafka-replica-verification.sh        zookeeper-server-start.sh
kafka-consumer-perf-test.sh        kafka-run-class.sh                   zookeeper-server-stop.sh
kafka-delegation-tokens.sh         kafka-server-start.sh                zookeeper-shell.sh
bash-4.4# 
```

如图，我们进入到kafka的安装路径

2、kafka 创建主题
使用以下命令创建一个主题名为testTopic ，zookeeper的地址是zookeeper:2182，副本数量是2个，有3个分区

```
kafka-topics.bat --create -zookeeper  zookeeper:2182 --replication-factor 2 --partitions 3 --topic  testTpoic
```

创建好主题之后就可以使用了

简单的消息生产者

在下面这段代码里面，使用向我们刚刚创建的主题发送了5条消息，并且指定了分区是0

```java
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerRecord;

import java.util.Properties;
import java.util.UUID;

/**
 * @author : jenkin
 * @date : Created at 2020/5/27 16:56
 * @description:
 * @modified By:
 * @version: 1.0
 */
public class Productor {

    private final KafkaProducer<String, String> producer;
    private Productor() {
        Properties props = new Properties();
        props.put("bootstrap.servers",
"172.16.161.51:9002,172.16.161.51:9003,172.16.161.51:9004");//xxx服务器ip
        props.put("acks", "all");//所有follower都响应了才认为消息提交成功，即"committed"
        props.put("retries", 0);//retries = MAX 无限重试，直到你意识到出现了问题:)
        props.put("batch.size", 16384);//producer将试图批处理消息记录，以减少请求次数.默认的批量处理消息字节数
        props.put("linger.ms", 1);//延迟1ms发送，这项设置将通过增加小的延迟来完成--即，不是立即发送一条记录，producer将会等待给定的延迟时间以允许其他消息记录发送，这些消息记录可以批量处理
        props.put("buffer.memory", 33554432);//producer可以用来缓存数据的内存大小。
        props.put("key.serializer",

"org.apache.kafka.common.serialization.IntegerSerializer");
        props.put("value.serializer",

"org.apache.kafka.common.serialization.StringSerializer");
        producer = new KafkaProducer<String, String>(props);
    }
    public void produce(String topic,int partition) {
        int messageNo = 1;
        final int COUNT = 5;
        while(messageNo < COUNT) {
            String key = String.valueOf(messageNo);
            String data = String.format("hello KafkaProducer message %s from  "+topic+" "+ UUID.randomUUID(), key);
            try {
                System.out.println("发送消息: "+data);
                producer.send(new ProducerRecord<String, String>(topic,partition,null, data));
            } catch (Exception e) {
                e.printStackTrace();
            }
            messageNo++;
        }
    }
    public static void main(String[] args) {
        new Productor().produce("testTopic",0);
    }
}
```

使用消费者消费消息

在下面的代码里面我们使用一个消费者消费刚刚创建的主题里面的数据，并且指定了分区是0

```
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.common.TopicPartition;

import java.util.Collections;
import java.util.List;
import java.util.Properties;
import java.util.concurrent.*;

/**
 * @author : jenkin
 * @date : Created at 2020/5/27 16:57
 * @description:
 * @modified By:
 * @version: 1.0
 */
public class ConsumerDemo {

    public static void main(String[] args){
        new Thread(()->{
            TopicPartition testLogTopic = new
TopicPartition("testTopic", 0);
            consumer("",testLogTopic);
        }).start();
    }
    public static  void consumer(String key, TopicPartition partitions){
        Properties properties = new Properties();
        properties.put("bootstrap.servers",
"172.16.161.51:9002,172.16.161.51:9003,172.16.161.51:9004");//服务器ip
        properties.put("group.id", "jd-group");
        properties.put("auto.commit.interval.ms", "1000");
        properties.put("auto.offset.reset", "latest");
        properties.put("session.timeout.ms", "30000");
        properties.put("key.deserializer",
"org.apache.kafka.common.serialization.StringDeserializer");
        properties.put("value.deserializer",
"org.apache.kafka.common.serialization.StringDeserializer");
        KafkaConsumer<String, String>  kafkaConsumer = new
KafkaConsumer<>(properties);
        kafkaConsumer.assign(Collections.singleton(partitions));
//
kafkaConsumer.subscribe(Collections.singleton(partitions.topic()));
        while (true) {
            if(Thread.currentThread().isInterrupted()){
                    break;
                }
                ConsumerRecords<String, String> records =
kafkaConsumer.poll(100);
                for (ConsumerRecord<String, String> record : records) {
                    System.out.println(Thread.currentThread().getId()+"------"+record.offset()+ "-------收到消息----------"+record.value()+"  来自主题："+record.topic()+"  分区："+record.partition());
                    kafkaConsumer.commitSync();
                }
            }
            kafkaConsumer.close();
        }
}
```