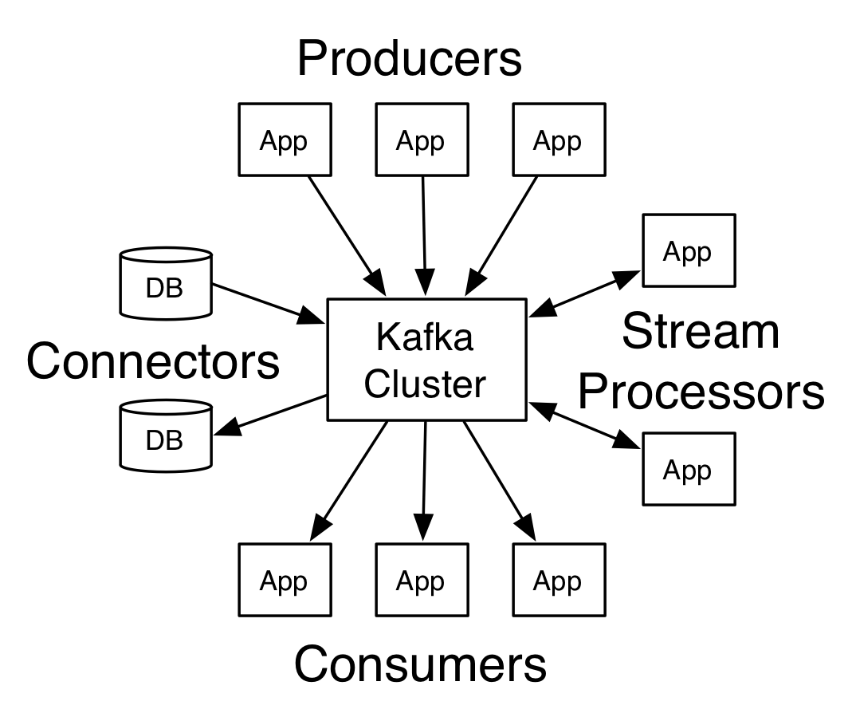
# Kafka文档

## 一、Kafka简介

Kafka是一个分布式的消息队列系统(Message Queue)。

官网：<https://kafka.apache.org/>



kafka集群有多个Broker服务器组成，每个类型的消息被定义为topic。

同一topic内部的消息按照一定的key和算法被分区(partition)存储在不同的Broker上。

消息生产者producer和消费者consumer可以在多个Broker上生产/消费topic

概念理解：

* Topics and Logs：

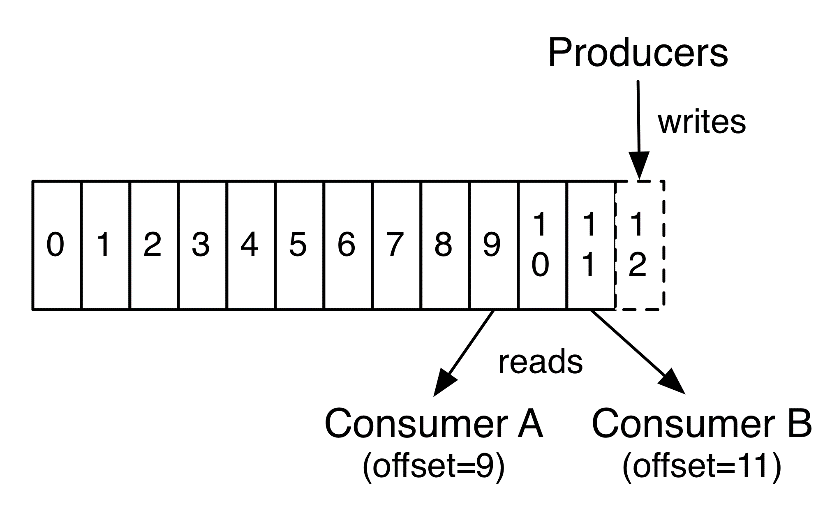
Topic即为每条发布到Kafka集群的消息都有一个类别，topic在Kafka中可以由多个消费者订阅、消费。

每个topic包含一个或多个partition（分区），partition数量可以在创建topic时指定，每个分区日志中记录了该分区的数据以及索引信息。如下图：



Kafka只保证一个分区内的消息有序，不能保证一个主题的不同分区之间的消息有序。如果你想要保证所有的消息都绝对有序可以只为一个主题分配一个分区。

分区会给每个消息记录分配一个顺序ID号（偏移量）， 能够唯一地标识该分区中的每个记录。Kafka集群保留所有发布的记录，不管这个记录有没有被消费过，Kafka提供相应策略通过配置从而对旧数据处理。



实际上，每个消费者唯一保存的元数据信息就是消费者当前消费日志的位移位置。位移位置是由消费者控制，即、消费者可以通过修改偏移量读取任何位置的数据。

* Distribution -- 分布式
* Producers -- 生产者

指定topic来发送消息到Kafka Broker

* Consumers -- 消费者

根据topic消费相应的消息

## 二、Kafka集群部署

**集群规划**：

Zookeeper集群共三台服务器，分别为：node06、node07、node08。

Kafka集群共三台服务器，分别为：node06、node07、node08。

### 1、Zookeeper集群准备

kafka是一个分布式消息队列，需要依赖ZooKeeper，请先安装好zk集群。

Zookeeper集群安装步骤略。

### 2、安装Kafka

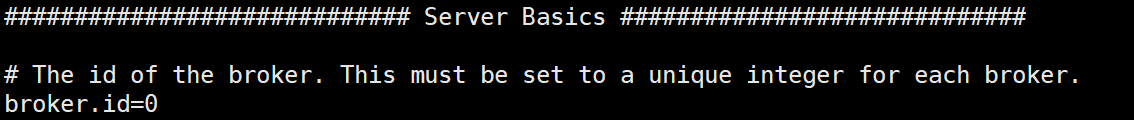
下载压缩包（官网地址：<http://kafka.apache.org/downloads.html>）

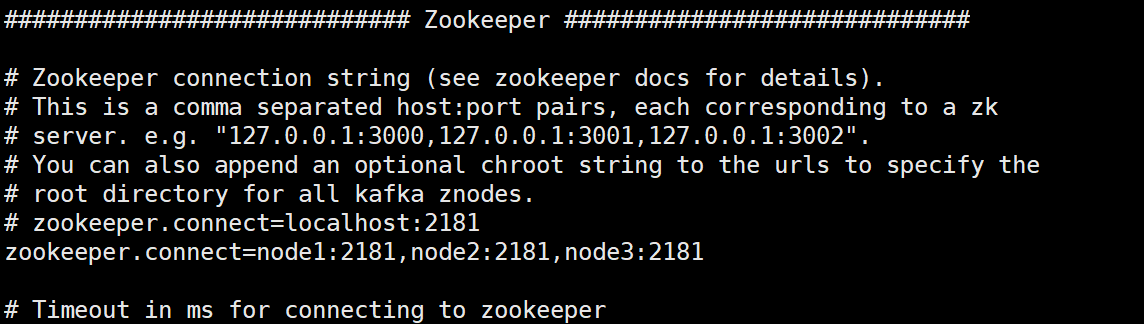
解压：

tar zxvf kafka\_2.10-0.9.0.1.tgz -C /opt/

mv kafka\_2.10-0.9.0.1/ kafka

修改配置文件：config/server.properties





核心配置参数说明：

broker.id: broker集群中唯一标识id，0、1、2、3依次增长（broker即Kafka集群中的一台服务器）

注：

当前Kafka集群共三台节点，分别为：node1、node2、node3。对应的broker.id分别为0、1、2。

zookeeper.connect: zk集群地址列表

将当前node1服务器上的Kafka目录同步到其他node2、node3服务器上：

scp -r /opt/kafka/ node2:/opt

scp -r /opt/kafka/ node3:/opt

修改node2、node3上Kafka配置文件中的broker.id（分别在node2、3服务器上执行以下命令修改broker.id）

sed -i -e 's/broker.id=.\*/broker.id=1/' /opt/kafka/config/server.properties

sed -i -e 's/broker.id=.\*/broker.id=2/' /opt/kafka/config/server.properties

### 3、启动Kafka集群

A、启动Zookeeper集群。

B、启动Kafka集群。

分别在三台服务器上执行以下命令启动：

bin/kafka-server-start.sh config/server.properties

### 4、测试

创建话题

（kafka-topics.sh --help查看帮助手册）

创建topic：

bin/kafka-topics.sh --zookeeper node06:2181,node07:2181,node08:2181 --create --replication-factor 2 --partitions 3 --topic test

*（参数说明：*

*--replication-factor：指定每个分区的复制因子个数，默认1个*

*--partitions：指定当前创建的kafka分区数量，默认为1个*

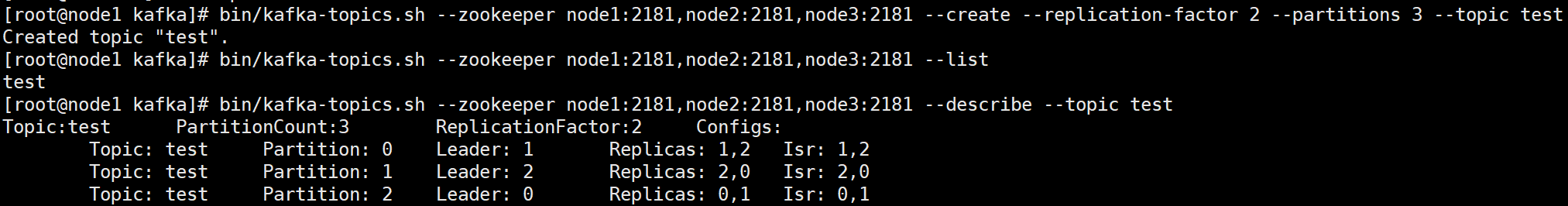
*--topic：指定新建topic的名称）*

查看topic列表：

bin/kafka-topics.sh --zookeeper node06:2181,node07:2181,node08:2181 --list

查看“test”topic描述：

bin/kafka-topics.sh --zookeeper node06:2181,node07:2181,node08:2181 --describe --topic test



创建生产者：

bin/kafka-console-producer.sh --broker-list node06:9092,node07:9092,node08:9092 --topic test

创建消费者：

bin/kafka-console-consumer.sh --zookeeper node06:2181,node07:2181,node08:2181 --from-beginning --topic test

注：

查看帮助手册：

bin/kafka-console-consumer.sh help

## 三、Flume & Kafka

### 1、Flume安装

Flume安装流程：

解压jar包

mv conf/flume-env.sh.template flume-env.sh

vi flume-env.sh java环境变量

./bin flume-ng version

/conf/下 创建配置文件fk.conf内容如下：

a1.sources = r1

a1.sinks = k1

a1.channels = c1

# Describe/configure the source

a1.sources.r1.type = avro

a1.sources.r1.bind = node06

a1.sources.r1.port = 41414

# Describe the sink

a1.sinks.k1.type = org.apache.flume.sink.kafka.KafkaSink

a1.sinks.k1.topic = testflume

a1.sinks.k1.brokerList = node06:9092,node07:9092,node08:9092

a1.sinks.k1.requiredAcks = 1

a1.sinks.k1.batchSize = 20

a1.sinks.k1.channel = c1

# Use a channel which buffers events in memory

a1.channels.c1.type = memory

a1.channels.c1.capacity = 1000000

a1.channels.c1.transactionCapacity = 10000

# Bind the source and sink to the channel

a1.sources.r1.channels = c1

a1.sinks.k1.channel = c1

### 2、Flume + Kafka

启动zk集群

A、启动Kafka集群。

bin/kafka-server-start.sh config/server.properties

B、配置Flume集群，并启动Flume集群。

bin/flume-ng agent -n a1 -c conf -f conf/fk.conf -Dflume.root.logger=DEBUG,console

### 3、测试

* 分别启动Zookeeper、Kafka、Flume集群。

zkServer.sh start

bin/kafka-server-start.sh config/server.properties

bin/flume-ng agent -n a1 -c conf -f conf/fk.conf -Dflume.root.logger=DEBUG,console

* 创建topic：（不用）

bin/kafka-topics.sh --zookeeper node06:2181,node07:2181,node08:2181 --create --replication-factor 2 --partitions 3 --topic testflume

bin/kafka-topics.sh --zookeeper node06:2181,node07:2181,node08:2181 --create --replication-factor 2 --partitions 1 --topic LogError

bin/kafka-topics.sh --zookeeper node06:2181,node07:2181,node08:2181 --list

bin/kafka-topics.sh --zookeeper node06:2181,node07:2181,node08:2181 --create --replication-factor 2 --partitions 1 --topic LogError

* 启动消费者：

bin/kafka-console-consumer.sh --zookeeper node06:2181,node07:2181,node08:2181 --from-beginning --topic testflume

启动生产者

bin/kafka-topics.sh --zookeeper node06:2181,node07:2181,node08:2181 --create --replication-factor 2 --partitions 1 --topic mylog\_cmcc

bin/kafka-console-consumer.sh --zookeeper node06:2181,node07:2181,node08:2181 --from-beginning --topic mylog\_cmcc

* 运行“RpcClientDemo”代码，通过rpc请求发送数据到Flume集群。

Flume中source类型为AVRO类型，此时通过Java发送rpc请求，测试数据是否传入Kafka。

其中，Java发送Rpc请求Flume代码示例如下：

（参考Flume官方文档：<http://flume.apache.org/FlumeDeveloperGuide.html>）

**import** org.apache.flume.Event;

**import** org.apache.flume.EventDeliveryException;

**import** org.apache.flume.api.RpcClient;

**import** org.apache.flume.api.RpcClientFactory;

**import** org.apache.flume.event.EventBuilder;

**import** java.nio.charset.Charset;

/\*\*

\* Flume官网案例

\* http://flume.apache.org/FlumeDeveloperGuide.html

\* **@author** root

\*/

**public** **class** RpcClientDemo {

**public** **static** **void** main(String[] args) {

MyRpcClientFacade client = **new** MyRpcClientFacade();

// Initialize client with the remote Flume agent's host and port

client.init("node1", 41414);

// Send 10 events to the remote Flume agent. That agent should be

// configured to listen with an AvroSource.

String sampleData = "Hello Flume!";

**for** (**int** i = 0; i < 10; i++) {

client.sendDataToFlume(sampleData);

System.***out***.println("发送数据：" + sampleData);

}

client.cleanUp();

}

}

**class** MyRpcClientFacade {

**private** RpcClient client;

**private** String hostname;

**private** **int** port;

**public** **void** init(String hostname, **int** port) {

// Setup the RPC connection

**this**.hostname = hostname;

**this**.port = port;

**this**.client = RpcClientFactory.*getDefaultInstance*(hostname, port);

// Use the following method to create a thrift client (instead of the

// above line):

// this.client = RpcClientFactory.getThriftInstance(hostname, port);

}

**public** **void** sendDataToFlume(String data) {

// Create a Flume Event object that encapsulates the sample data

Event event = EventBuilder.*withBody*(data, Charset.*forName*("UTF-8"));

// Send the event

**try** {

client.append(event);

} **catch** (EventDeliveryException e) {

// clean up and recreate the client

client.close();

client = **null**;

client = RpcClientFactory.*getDefaultInstance*(hostname, port);

// Use the following method to create a thrift client (instead of

// the above line):

// this.client = RpcClientFactory.getThriftInstance(hostname, port);

}

}

**public** **void** cleanUp() {

// Close the RPC connection

client.close();

}

}

## 四、Storm & Kafka

官网地址：

<http://storm.apache.org/about/integrates.html>

## 五、flume+kafka+spout

bin/kafka-topics.sh --zookeeper node06:2181,node07:2181,node08:2181 --create --replication-factor 2 --partitions 1 --topic LogError