马哥教育
IT人的高薪职业学院

消息队列与微服务
训练营(day2)

讲师： 张士杰/杰哥

http://www.magedu.com

1.拥有 RHCA、OpenStack、EXIN DevOps Master 等专业证书。

2.多年大型互联网一线工作经验，曾在互联网金融和互联网电商等公司任职运维架构师、云平台架构师等职位。

3.曾维护企业数千台服务器的业务规模。

4.拥有十一年一线运维工作经验。

5.熟练虚拟化、Docker、kubernetes、公有云等业务环境规划、实施与运维。

- 第一天:
  - MQ简介及应用场景
  - RabbitMQ单机及使用
  - RabbitMQ集群及使用

- **第二天:**
  - **zookeeper单机、集群及实战案例**
  - **kafka应用基础**
  - **kafka实战案例**

- 第三天:
  - 微服务简介
  - 生产者与消费者模型
  - 基于dubbo的生产者与消费者模型微服务实战案例

zookeeper简介:

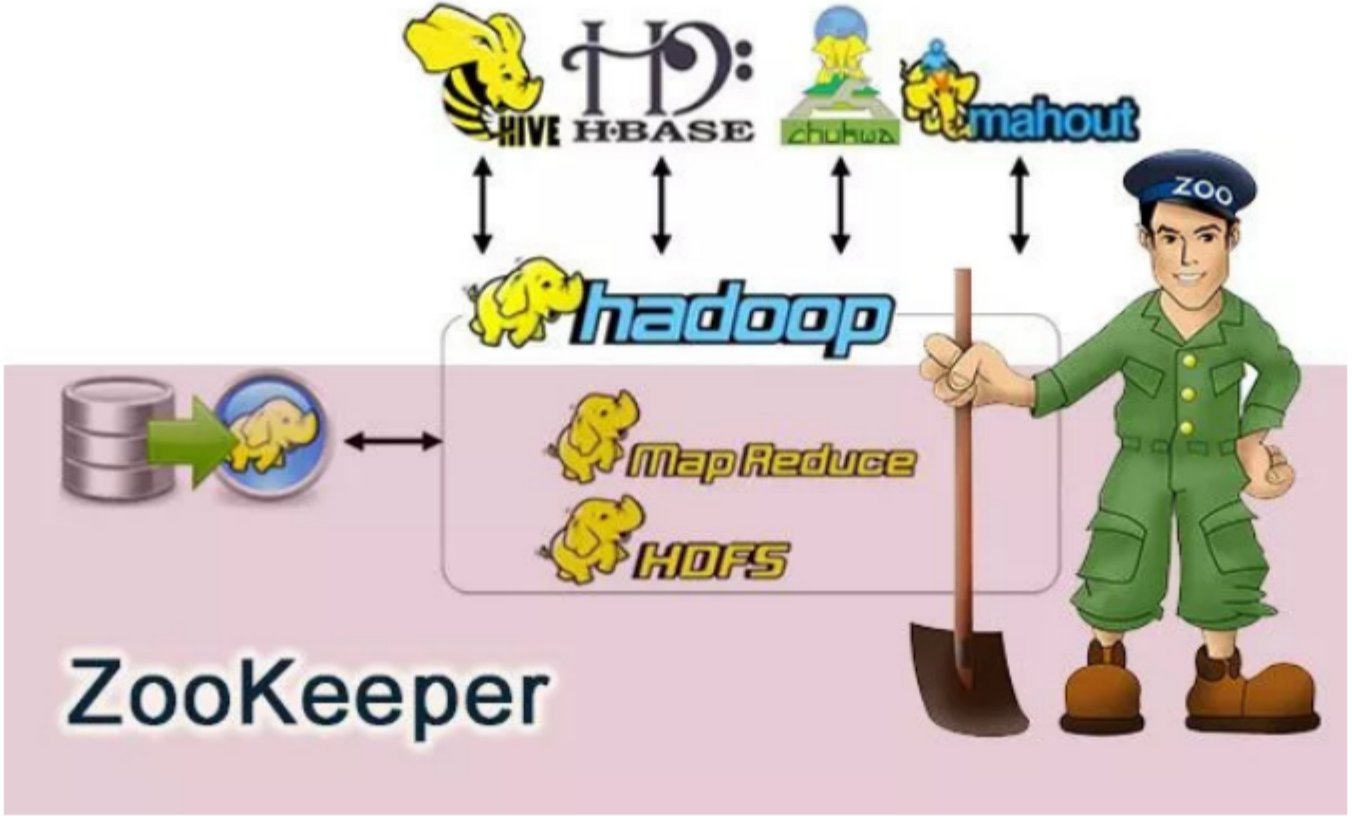
https://zookeeper.apache.org/

ZooKeeper是一个分布式协调服务，最早起源于雅虎研究院的一个研究小组。

ZooKeeper使用场景：

ZooKeeper是一个分布式服务框架，它主要是用来解决分布式应用中经常遇到的一些数据管理问题，如：命名服务、状态同步、配置中心、集群管理等。

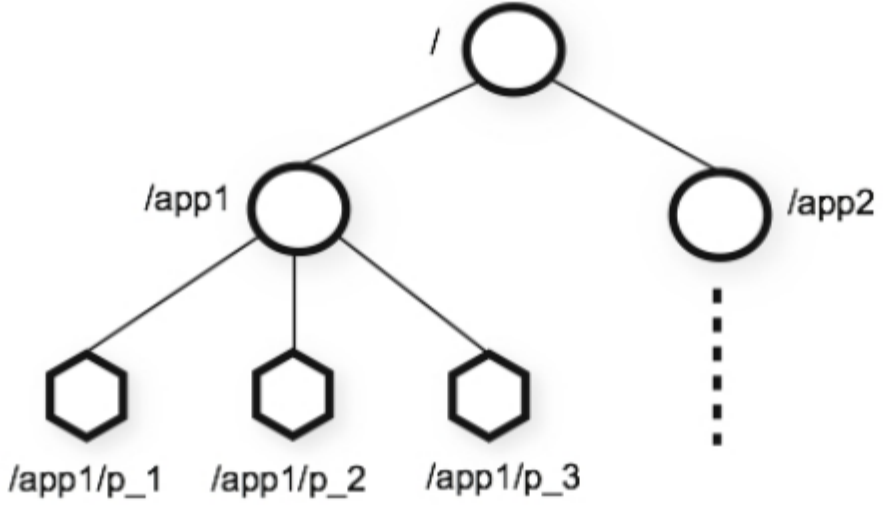
zookeeper使用场景:



ZooKeeper使用场景：

命名服务：

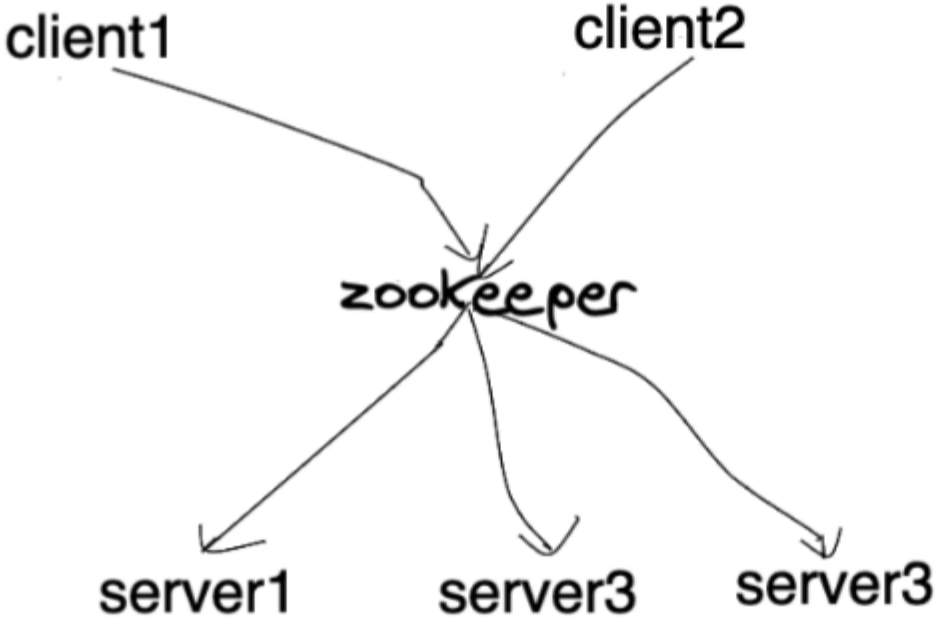
命名服务是分布式系统中比较常见的一类场景。命名服务是分布式系统最基本的公共服务之一。在分布式系统中，被命名的实体通常可以是集群中的机器、提供的服务地址或远程对象等——这些我们都可以统称它们为名字（Name），其中较为常见的就是一些分布式服务框架（如RPC、RMI）中的服务地址列表，通过使用命名服务，客户端应用能够根据指定名字来获取资源的实体、服务地址和提供者的信息等。



ZooKeeper使用场景：

状态同步：

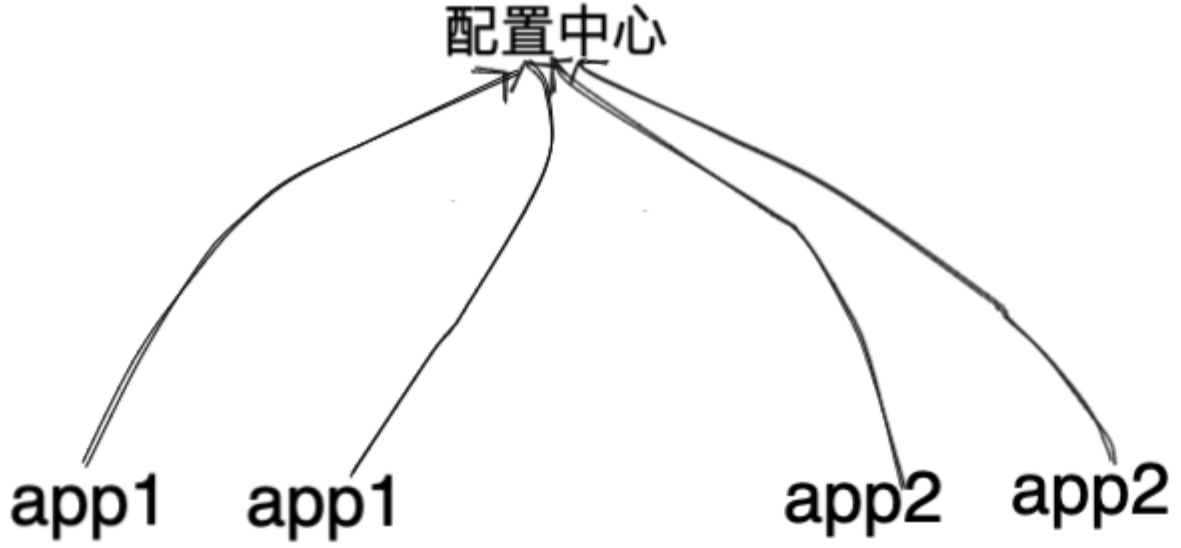
每个节点除了存储数据内容和node节点状态信息之外，还存储了已经注册的APP的状态信息，当有些节点或APP不可用，就将当前状态同步给客户端。



ZooKeeper使用场景：

配置中心：

现在我们大多数应用都是采用的是分布式开发的应用，部署到不同的服务器上，而同一个应用程序的配置文件在每个服务器都是一样的，还有就是多个程序存在相同的配置，当我们配置文件中有个配置属性需要改变，我们需要改变每个程序的配置属性，这样会很麻烦的去修改配置，那么可用使用ZooKeeper来实现配置中心。



ZooKeeper使用场景：

集群管理：

所谓集群管理，包括集群监控与集群控制两大块，前者侧重对集群运行时状态的收集，后者则是对集群进行操作与控制，在日常开发和运维过程中，我们经常会有类似于如下的需求：

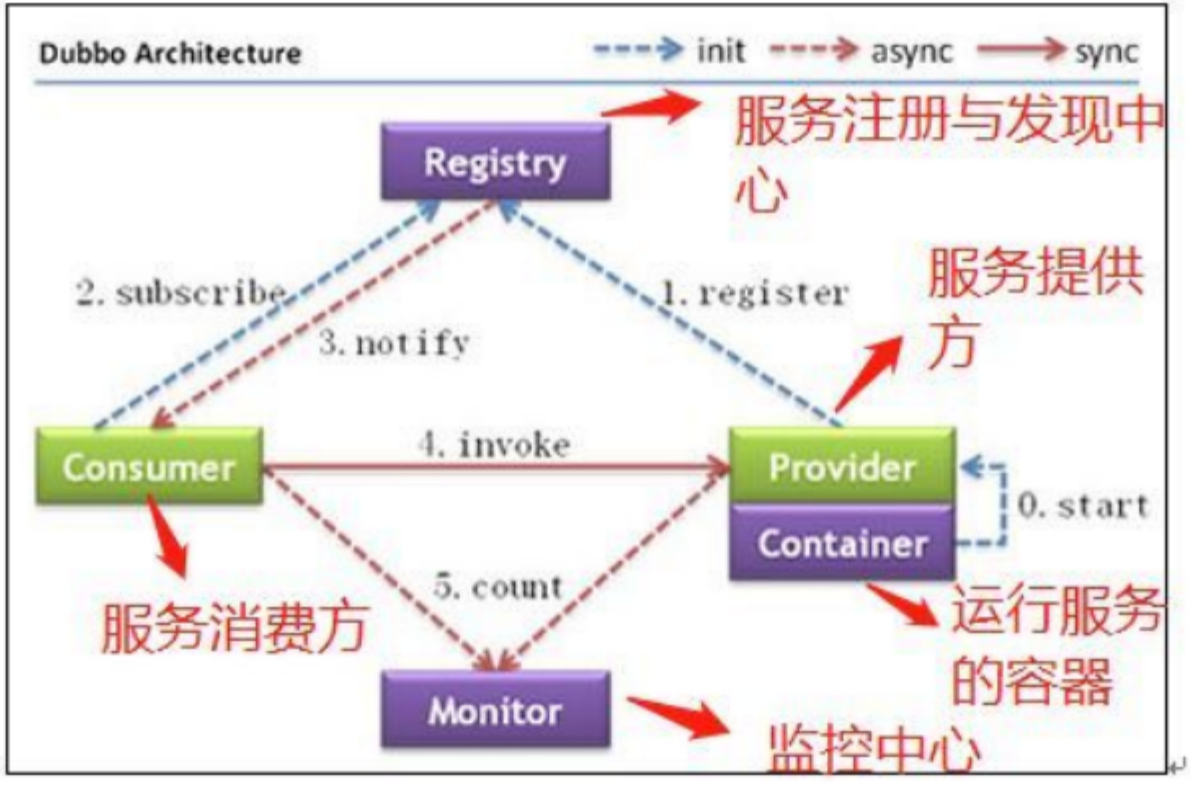
希望知道当前集群中究竟有多少机器在工作。
对集群中每台机器的运行时状态进行数据收集。
对集群中机器进行上下线操作。

ZooKeeper具有以下两大特性。

客户端如果对ZooKeeper的一个数据节点注册Watcher监听，那么当该数据节点的内容或是其子节点列表发生变更时，ZooKeeper服务器就会向订阅的客户端发送变更通知。
对在ZooKeeper上创建的临时节点，一旦客户端与服务器之间的会话失效，那么该临时节点也就被自动清除。

Watcher（事件监听器），是Zookeeper中的一个很重要的特性。Zookeeper允许用户在指定节点上注册一些Watcher，并且在一些特定事件触发的时候，ZooKeeper服务端会将事件通知到感兴趣的客户端上去，该机制是Zookeeper实现分布式协调服务的重要特性。

ZooKeeper使用场景：



ZooKeeper 单机：

https://github.com/apache/zookeeper #github
https://zookeeper.apache.org/doc/r3.4.14/zookeeperAdmin.html#sc_requiredSoftware #官方依赖介绍

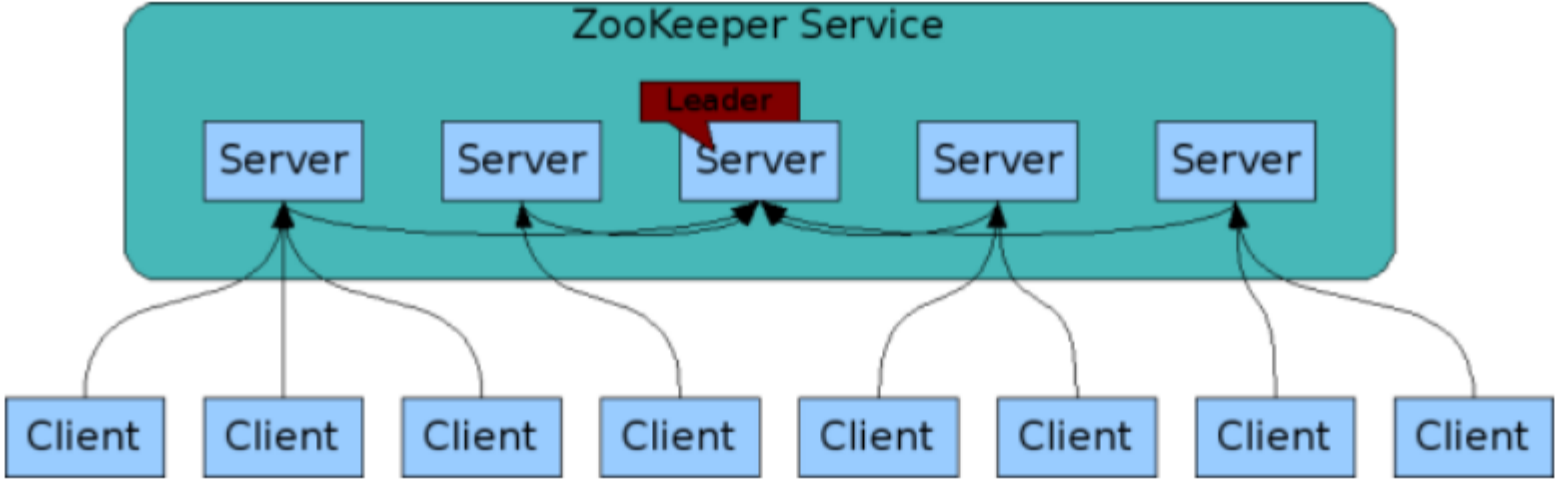
```
# tar xvf zookeeper-x.y.z.tar.gz
# ln -sv /path/zookeeper-x.y.z /path/zookeeper
'/path/zookeeper' -> '/path/zookeeper-x.y.z'

# cd /path/zookeeper/conf/
# cp zoo_sample.cfg  zoo.cfg

# grep ^[a-Z] zoo.cfg #当前配置
tickTime=2000 #服务器与服务器之间的单次心跳检测时间间隔，单位为毫秒
initLimit=10 #集群环境中leader服务器与follower服务器初始连接心跳次数，即多少个2000毫秒
syncLimit=5 # leader与follower之间连接完成之后，后期检测发送和应答的心跳次数，如果在设置的时间内(5*2000)不能相互进行通信，那么此节点将被视为不可用。
dataDir=/tmp/zookeeper #数据目录
clientPort=2181 #客户端访问端口
autopurge.snapRetainCount=3 # 保存最近的数据快照数量，超出的将会被清除
autopurge.purgeInterval=1  # 自动触发清除任务时间间隔，以小时为间隔单位，默认为0表示不自动清除。
```

ZooKeeper 单机：

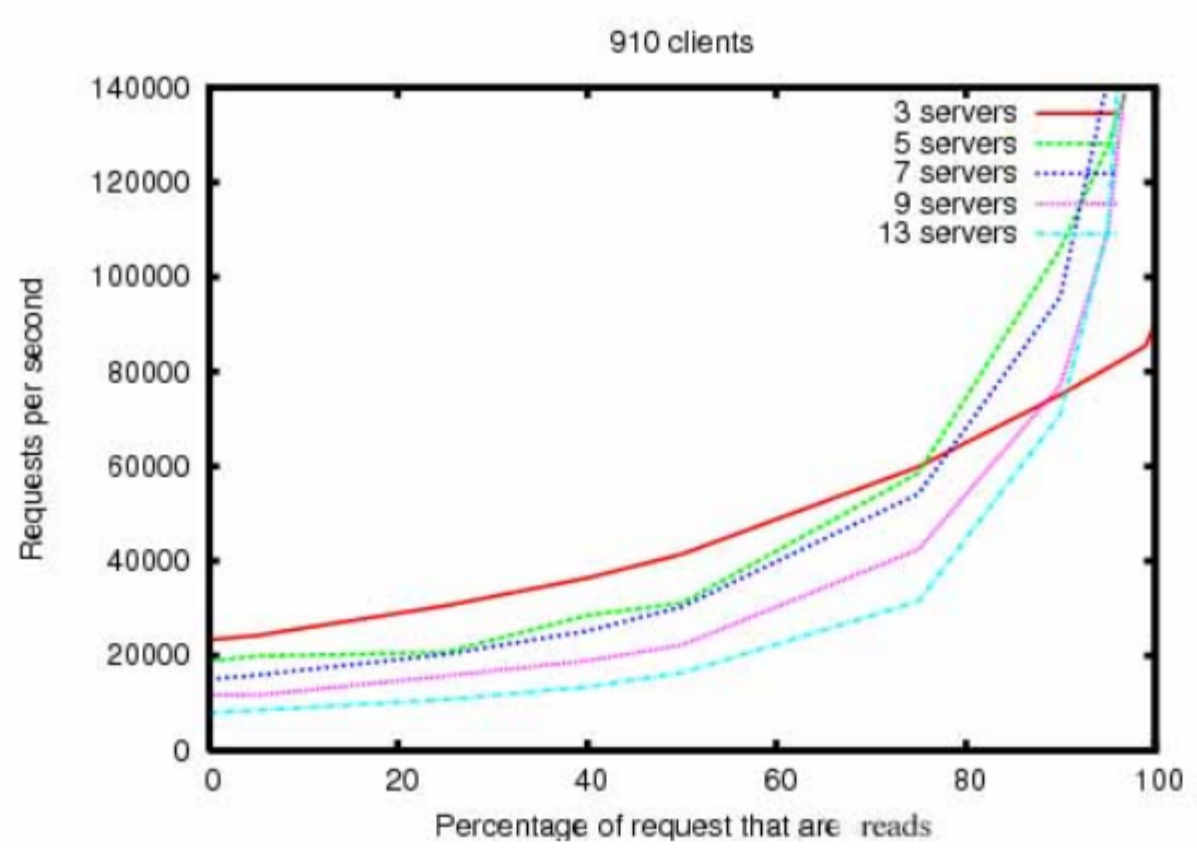

```
# /apps/zookeeper/bin/zkServer.sh  start
/usr/bin/java
ZooKeeper JMX enabled by default
Using config: /apps/zookeeper/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED #单机模式
```

ZooKeeper 集群:

ZooKeeper集群用于解决单点和单机性能及数据高可用等问题。



多节点性能：

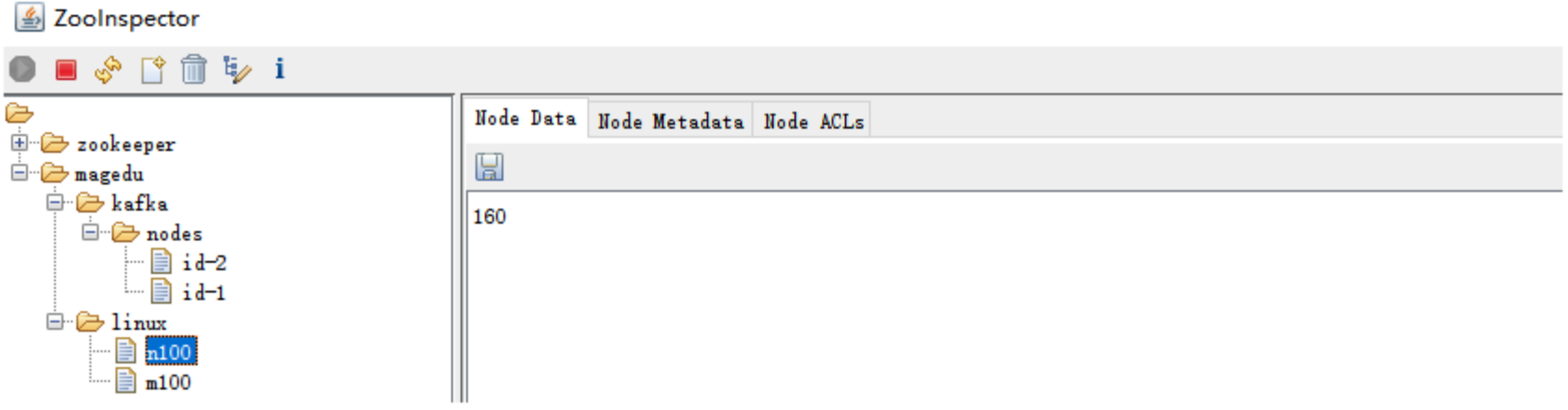


ZooKeeper 集群：

角色		主要工作描述
领导者		1. 事务请求的唯一调度和处理者，保证集群事务处理的顺序性； 2. 集群内部各服务器的调度者
学习者（Learner）	跟随者（Follower）	1. 处理客户端非事务请求，转发事务请求给Leader服务器 2. 参与事务请求Proposal的投票 3. 参与Leader选举的投票
	观察者（Observer）	Follower 和 Observer 唯一的区别在于 Observer 机器不参与 Leader 的选举过程，也不参与写操作的"过半写成功"策略，因此 Observer 机器可以在不影响写性能的情况下提升集群的读性能。
客户端（Client）		请求发起方

ZooKeeper 使用：


```
~# /apps/zookeeper/bin/zkCli.sh  -server 172.31.6.201:2181
0] help
2] ls /
9] ls /magedu/kafka/nodes
9] get /magedu/kafka/nodes/id-2
172.16.100.21
10] create    /magedu/linux
Created /magedu/linux
11] create    /magedu/linux/m100 120
Created /magedu/linux/m100
12] get  /magedu/linux/m100
120
13] delete  /magedu/linux/m100
```



ZooKeeper 集群:

1.各服务器配置JDK环境
2.zoo.cfg配置文件
3.集群ip配置

```
# grep –v "^#" /apps/zookeeper/conf/zoo.cfg
tickTime=2000
initLimit=10
syncLimit=5
dataDir=/apps/zookeeper/data
clientPort=2181
maxClientCnxns=4096
autopurge.snapRetainCount=128
autopurge.purgeInterval=1
server.1=172.31.7.101:2888:3888 # server.服务器编号=服务器IP:LF数据同步端口:LF选举端口
server.2=172.31.7.102:2888:3888
server.3=172.31.7.103:2888:3888
```

4.各服务器同步配置文件
5.创建数据目录
6.生成ID文件

```
        # echo "x" > /usr/local/zookeeper/data/myid
```

6.启动
7.各服务器验证zookeeper状态

```
root@mq-server1:/apps/zookeeper/conf# /apps/zookeeper/bin/zkServer.sh  status
/usr/bin/java
ZooKeeper JMX enabled by default
Using config: /apps/zookeeper/bin/../conf/zoo.cfg
Client port found: 2181. Client address: localhost. Client SSL: false.
Mode: follower
```

zookeeper集群：

节点角色状态：

LOOKING：寻找Leader状态，处于该状态需要进入选举流程
LEADING：领导者状态，处于该状态的节点说明是角色已经是Leader
FOLLOWING：跟随者状态，表示Leader已经选举出来，当前节点角色是follower
OBSERVER：观察者状态，表明当前节点角色是observer

节点选举ID:

ZXID（zookeeper transaction id，事务ID）：每个改变Zookeeper状态的操作都会形成一个对应的zxid，此ID会随着数据写入、update而更新。
SID：服务器的唯一标识
myid：服务器的唯一标识(SID)，通过配置myid文件指定，集群中唯一。

首次leader选举过程：

当集群中的zookeeper节点首次启动以后，会根据配置文件中指定的zookeeper节点地址进行通信并进行leader选择操作，过程如下：

1.每个zookeeper 都会发出投票，由于是第一次选举leader，因此每个节点都会把自己当做leader角色进行选举，每个zookeeper的投票中都会包含自己的myid和zxid，此时zookeeper 1的投票为myid为1，初始zxid有一个初始值，后期会随着数据更新而自动变化，zookeeper2的投票为myid为2，初始zxid为初始生成的值。
2.每个节点接受并检查对方的投票信息，比如投票时间、是否状态为LOOKING状态的投票。
3.对比投票，优先检查zxid，如果zxid不一样则zxid大的为leader，如果zxid相同则继续对比myid，myid大的一方为leader

成为Leader的必要条件： Leader要具有最高的zxid；当集群的规模是n时，集群中大多数的机器（至少n/2+1）得到响应并follow选出的Leader。
心跳机制：Leader与Follower利用PING来感知对方的是否存活，当Leader无法响应PING时，将重新发起Leader选举。

leader重新选举过程：

1.zk服务器故障以后或集群故障重新恢复后，会触发leader选举，选举过程会优先对比zxid(事务ID)，事务ID最新的zk服务器将成为集群的leader。

2.然后Leader就会打开2888端口并开始等待Follower连接2888端口同步数据，Follower连接到leader并将自己的zxid发送给Leader，Leader根据Follower的zxid确定同步点将数据发送给follower，发送完成同步后通知Follower已经成为uptodate状态，Follower收到uptodate消息后重新接受client的请求进行服务了。

Kafka简介：

kafka通过O(1)的磁盘数据结构提供消息的持久化，这种结构对于即使数以TB的消息存储也能够保持长时间的稳定性能，O(1)就是最低的时空复杂度算法，也就是耗时/耗空间与输入数据大小无关，无论输入数据增大多少倍，耗时/耗空间都不变，哈希算法就是典型的O(1)时间复杂度，无论数据规模多大，都可以在一次计算后找到目标，O(n)为数据量增大几倍，耗时也增大几倍，遍历就是典型的O(n)算法。

kafak特性：

高吞吐量：即使是非常普通的硬件Kafka也可以支持每秒数百万的消息。
可扩展：基于zk随时横向扩容。
持久存储：数据基于磁盘持久化保存。
高可用性：集群部分节点宕机不丢失数据不影响业务。

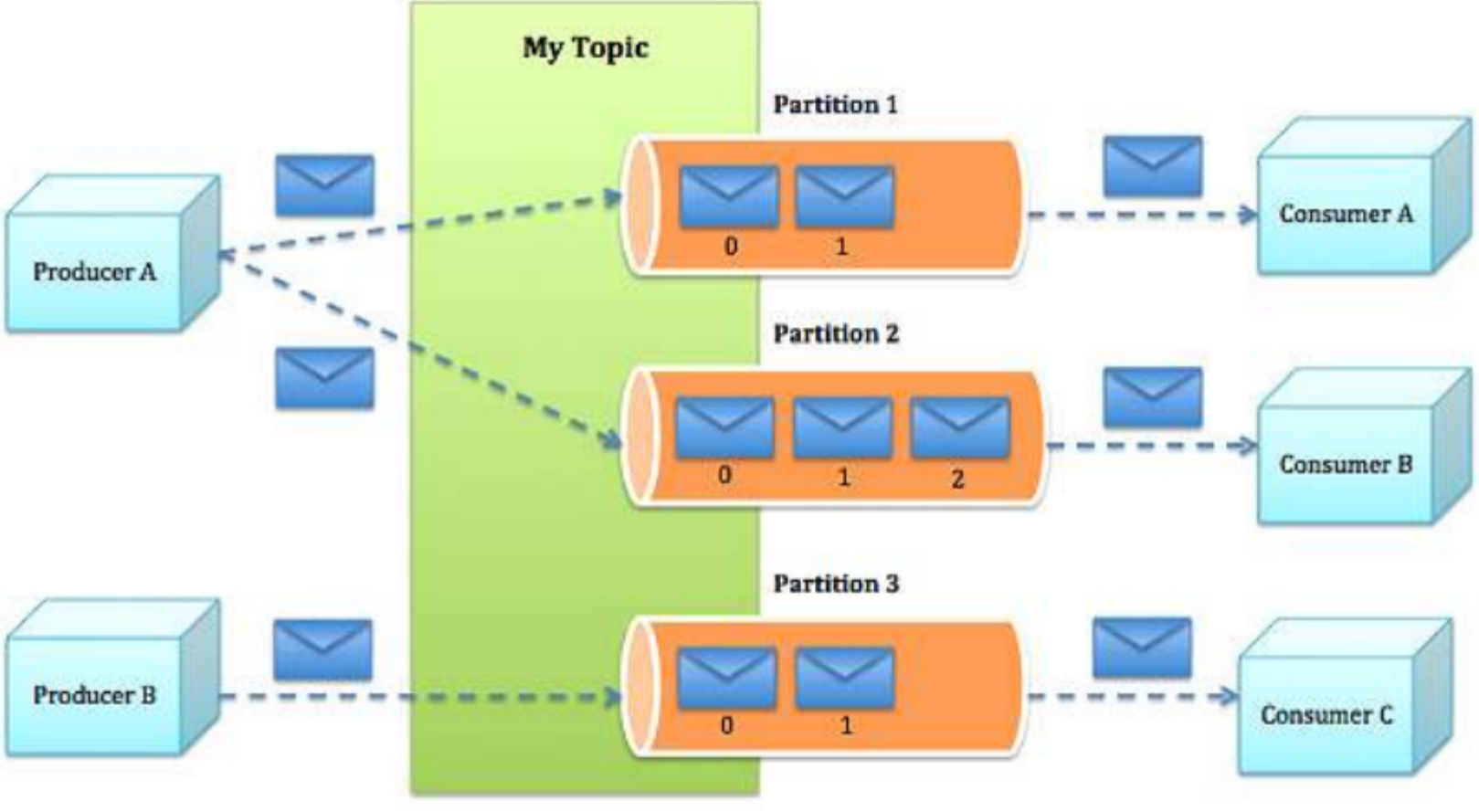
Kafka角色：

Broker：Kafka集群包含一个或多个服务器，这种服务器被称为broker。
Topic：每条发布到Kafka集群的消息都有一个类别，这个类别被称为topic，（物理上不同topic的消息分开存储在不同的文件夹，逻辑上一个topic的消息虽然保存于一个或多个broker上，但用户只需指定消息的topic即可生产或消费数据而不必关心数据存于何处），topic在逻辑上对record(记录、日志)进行分组保存，消费者需要订阅相应的topic才能消费topic中的消息。
Partition：是物理上的概念，每个topic包含一个或多个partition，创建topic时可指定parition数量，每个partition对应于一个文件夹，该文件夹下存储该partition的数据和索引文件，为了实现实现数据的高可用，比如将分区0的数据分散到不同的kafka节点，每一个分区都有一个broker作为leader和一个broker作为Follower。
分区的优势(分区因子为3)：

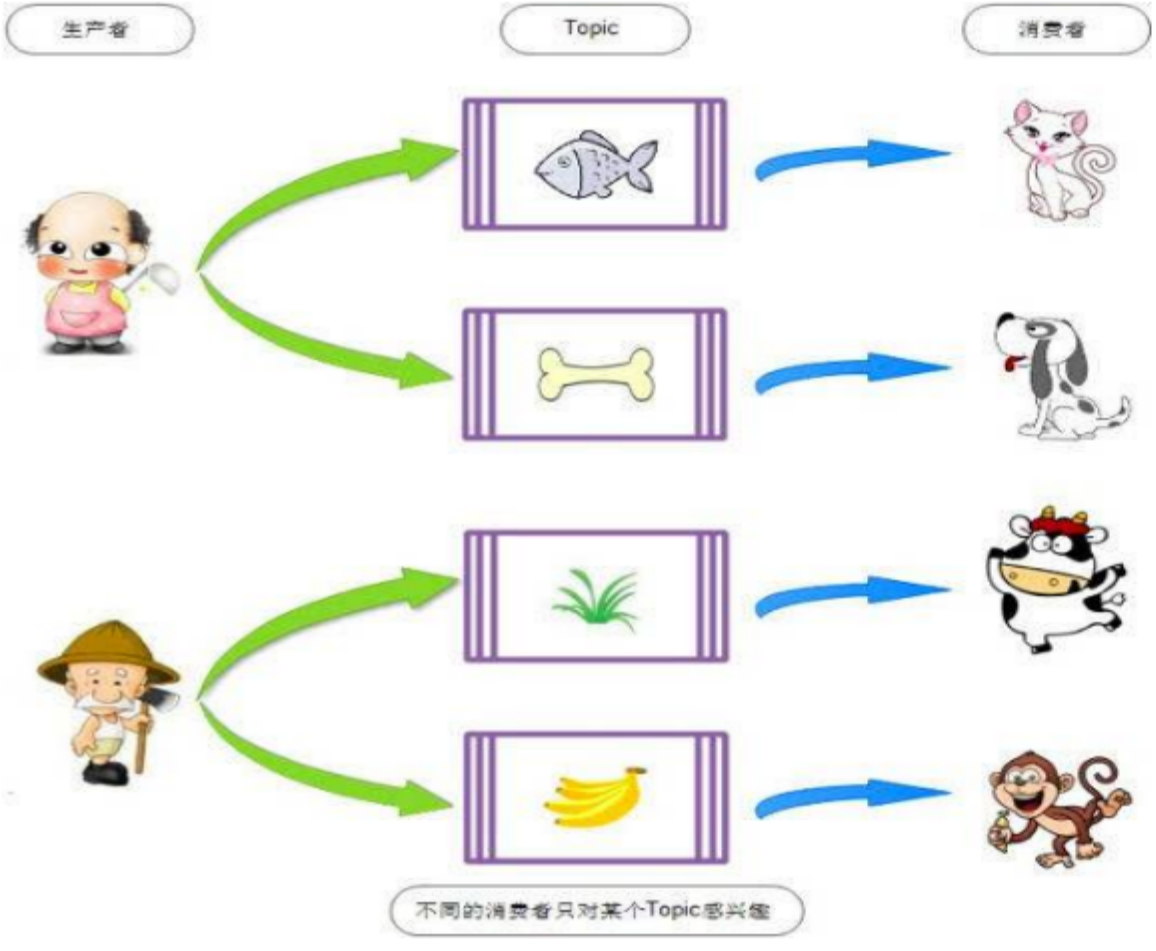
一：实现存储空间的横向扩容，即将多个kafka服务器的空间结合利用
二：提升性能，多服务器读写
三：实现高可用，不同的分区分布在不同的kafka服务器，比如分区0的leader为服务器A，则服务器B和服务器C为A的follower，而分区1的leader为服务器B，则服务器A和C为服务器B的follower，而分区2的leader为C，则服务器A和B为C的follower。

Producer：负责发布消息到Kafka broker。
Consumer：消费消息，每个consumer属于一个特定的consuer group（可为每个consumer指定group name，若不指定group name则属于默认的group），使用consumer high level API时，同一topic的一条消息只能被同一个consumer group内的一个consumer消费，但多个consumer group可同时消费这一消息。

Kafka角色：



Kafka角色：



Kafka部署：

版本选择：
http://kafka.apache.org/downloads
Scala 2.12– kafka_2.12–2.7.0.tgz　#版本格式 kafka_scala版本_kafka版本

部署三台服务器的高可用kafka环境

部署环境：
Server1: 172.31.6.201
Server2: 172.31.6.202
Server3: 172.31.6.203

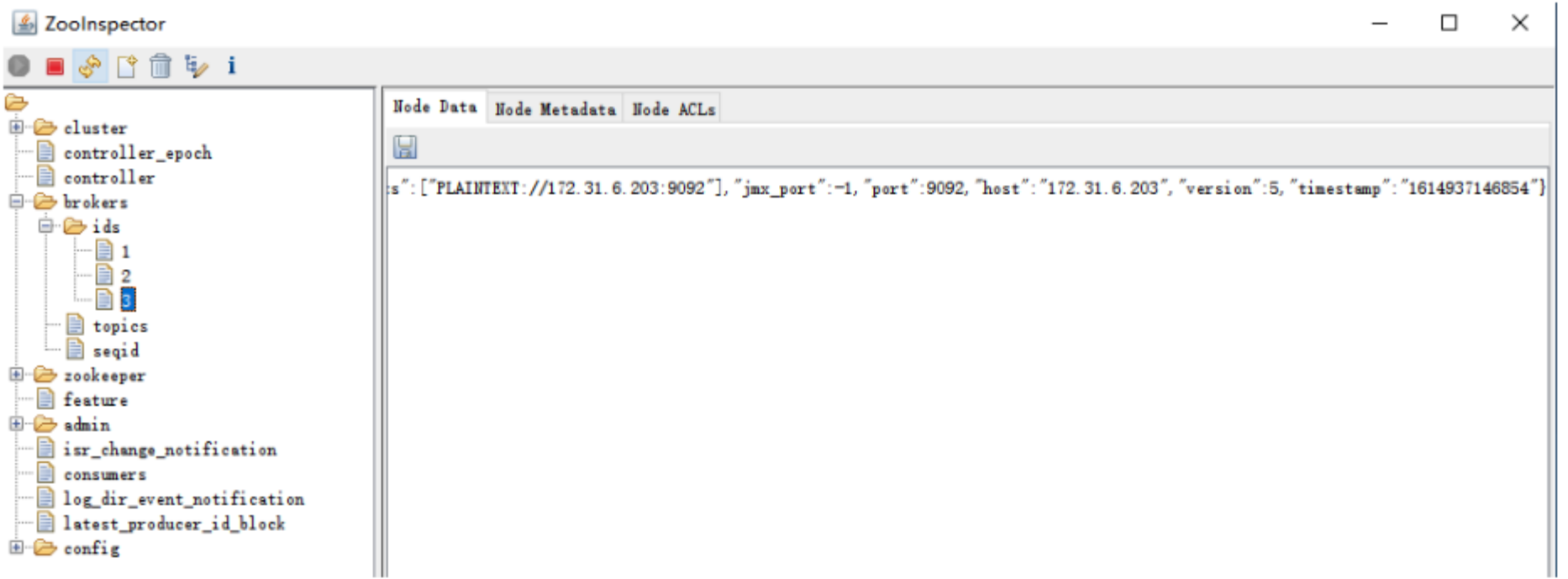
http://kafka.apache.org/quickstart

Kafka部署：


```
~# tar xvf kafka_2.13-2.7.0.tgz
~# cd /apps/kafka/config/
# vim server.properties
21 broker.id=1每个broker在集群中的唯一标识，正整数。
31 listeners=PLAINTEXT://172.18.0.101:9092 #监听地址
60 log.dirs=/apps/kafka/kafka-logs  #kakfa用于保存数据的目录，所有的消息都会存储在该目录当中
65 num.partitions=1 #设置创新新的topic默认分区数量
103 log.retention.hours=168 #设置kafka中消息保留时间，默认为168小时即7天
#zookeeper.connect 指定连接的zk的地址,zk中存储了broker的元数据信息,格式如下：
123 zookeeper.connect=172.31.6.201:2181,172.31.6.202:2181,172.31.6.203:2181
126 zookeeper.connection.timeout.ms=18000 #设置连接zookeeper的超时时间，默认18秒钟早期版本6秒钟

# mkdir /apps/kafka/kafka-logs #创建数据目录

#各节点启动kafka
# /apps/kafka/bin/kafka-server-start.sh -daemon /apps/kafka/config/server.properties
```

```
[2021-03-05 17:39:02,790] INFO Kafka version: 2.7.0 (org.apache.kafka.common.utils.AppInfoParser)
[2021-03-05 17:39:02,790] INFO Kafka commitId: 448719dc99a19793 (org.apache.kafka.common.utils.AppInfoParser)
[2021-03-05 17:39:02,790] INFO Kafka startTimeMs: 1614937142787 (org.apache.kafka.common.utils.AppInfoParser)
[2021-03-05 17:39:02,792] INFO [KafkaServer id=1] started (kafka.server.KafkaServer)
```

Kafka部署：



Kafka测试：

创建名为magedu的topic(标题、话题)，partitions(分区)为3，replication(每个分区的副本数/每个分区的分区因子)为3的topic(主题)：

在任意kafaka服务器操作：

```
# /apps/kafka/bin/kafka-topics.sh --create  --zookeeper 172.31.6.201:2181,172.31.6.202:2181,172.31.6.203:2181 --partitions 3 --replication-factor 3 --topic magedu
Created topic magedu.
```

验证topic：

```
# /apps/kafka/bin/kafka-topics.sh  --describe --zookeeper 172.31.6.201:2181,172.31.6.202:2181,172.31.6.203:2181  --topic magedu
Topic: magedu        PartitionCount: 3     ReplicationFactor: 3     Configs:
        Topic: magedu   Partition: 0          Leader: 2               Replicas: 2,3,1 Isr: 2,3,1
        Topic: magedu   Partition: 1          Leader: 3               Replicas: 3,1,2 Isr: 3,1,2
        Topic: magedu   Partition: 2          Leader: 1               Replicas: 1,2,3 Isr: 1,2,3
```

状态说明：magedu有三个分区分别为0、1、2，分区0的leader是2（broker.id），分区0有三个副本，并且状态都为Isr（In-sync，表示可以参加选举成为leader）。

Kafka测试：

获取所有topic：

```
# /apps/kafka/bin/kafka-topics.sh  --list --zookeeper 172.31.6.201:2181,172.31.6.202:2181,172.31.6.203:2181
```

测试发送消息：

```
/apps/kafka/bin/kafka-console-producer.sh --broker-list  172.31.6.201:9092,172.31.6.201:9092,172.31.6.201:9092 --topic  magedu
```

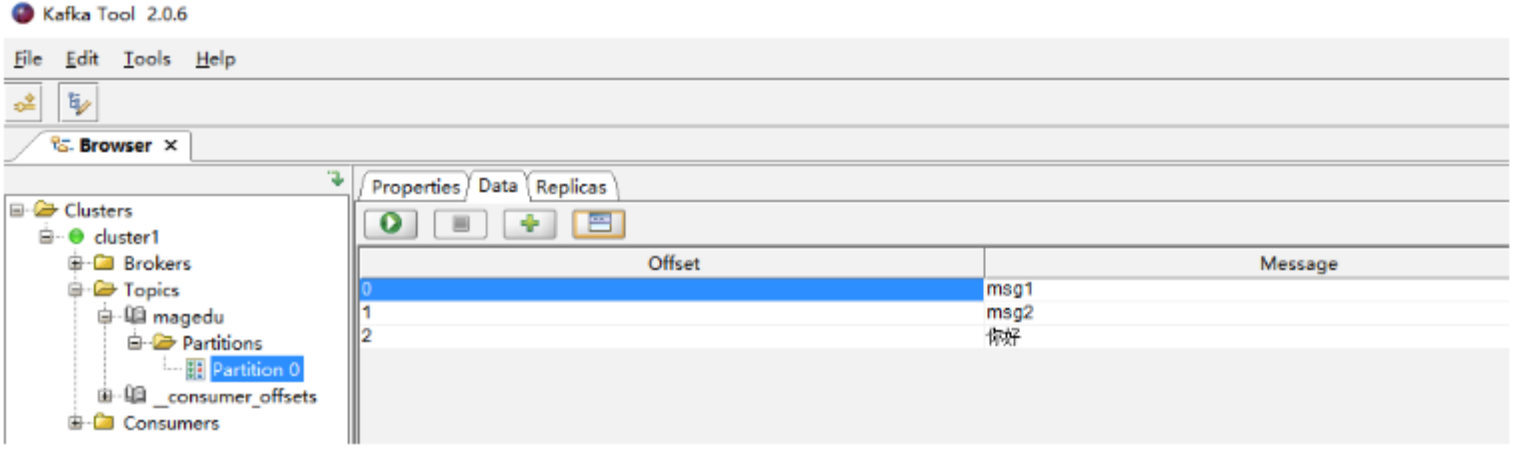
测试获取消息：

```
# /apps/kafka/bin/kafka-console-consumer.sh  --topic magedu --bootstrap-server 172.31.6.201:9092,172.31.6.201:9092,172.31.6.201:9092  --from-beginning
```

删除topic：

```
# /apps/kafka/bin/kafka-topics.sh --delete --zookeeper 172.31.6.201:2181,172.31.6.202:2181,172.31.6.203:2181 --topic magedu
Topic magedu is marked for deletion.
```

Kafka部署：



日志收集案例：

elasticsearch cluster
logstash
kafka+zookeeper
filebeat

马哥教育
IT人的高薪职业学院

Thank You!

讲师：张士杰（杰哥）
http://www.magedu.com