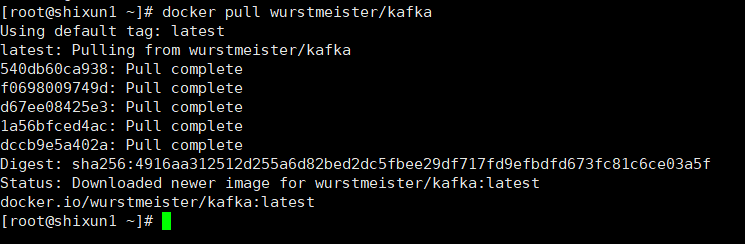
# 实现单台kafka应用测试

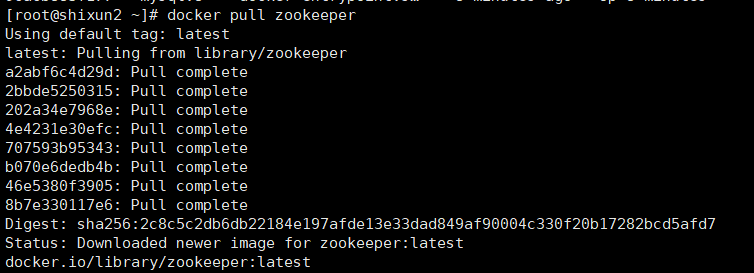
## 拉取镜像

环境要求：已经部署完成docker

命令：docker pull zookeeper 拉取zookeeper镜像

docker pull wurstmeister/kafka 拉取kafka镜像





## 创建网桥

命令：docker network create -d macvlan --subnet=192.168.9.0/24 --gateway=192.168.9.2 -o parent=ens33 mysql\_mac

## 创建docker容器

docker create -p 2181:2181 --name zookeeper --privileged=true \

-v /home/data/zookeeper/data/:/var/lib/zookeeper \

-v /home/data/zookeeper/log:/datalog \

--network my-macvlan-net --ip 192.168.9.10 \

Zookeeper

## 创建kafka容器

docker run -d --name kafka --publish 9092:9092 \

--env KAFKA\_ZOOKEEPER\_CONNECT=192.168.9.10:2181 \

--env KAFKA\_ADVERTISED\_HOST\_NAME=192.168.9.9 \

--env KAFKA\_ADVERTISED\_PORT=9092 \

--env KAFKA\_LOG\_DIRS=/kafka/kafka-logs-1 \

-v /usr/local/kafka/data/:/var/lib/zookeeper \

-v /usr/local/kafka/logs:/kafka/kafka-logs-1 \

--network my-macvlan-net --ip 192.168.9.9 \

wurstmeister/kafka

参数解释

·--env KAFKA\_ZOOKEEPER\_CONNECT：指定 ZooKeeper 的地址和端口。

·指定 Kafka 的广（advertised）主机名或 IP 地址。这是用于在生产者和消费 者之间传递正确的连接信息。

·--env KAFKA\_ADVERTISED\_PORT：指定Kafka 的广告端口号。

·--env KAFKA\_LOG\_DIRS：设置 Kafka 的日志目录路径。

·-v /usr/local/kafka/data/:/var/lib/zookeeper：将主机的目录挂载到容器的目录， 用于持久化存储 ZooKeeper 数据。

·-v /usr/local/kafka/logs:/kafka/kafka-logs-1：将主机的目录挂载到容器的目录， 用于持久化存储 Kafka 的日志文件。

·--network my-macvlan-net：将容器连接到名为 "my-macvlan-net" 的自定义网 络。

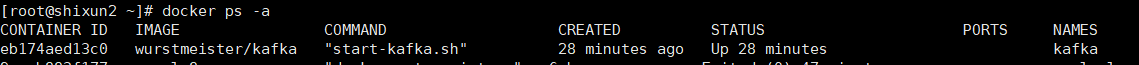
·--ip 192.168.9.9：为容器指定 IP 地址。

## 启动容器

命令：docker start kafka

命令：docker start zookeeper

容器状态为UP表示容器启动成功



容器状态为UP表示容器启动成功



## 创建查看 Topic

创建一个名为 test 的 Topic，该 Topic 包含一个分区和一个 Replica。

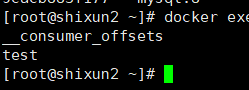
docker exec kafka kafka-topics.sh --create --zookeeper 192.168.9.10:2181 --replication-factor 1 --partitions 1 --topic test

·--replication-factor 1：指定了主题的副本因子。副本因子定义了每个分区的副 本数量。在这里，副本因子为 1，表示每个分区只有一个副本。

·--partitions 1：指定了主题的分区数。分区是 Kafka 中消息的逻辑单元，它允 许消息在集群中进行并行处理。在这里，主题只有一个分区。

创建后可以执行如下命令查看当前的 Topics：

docker exec kafka kafka-topics.sh --list --zookeeper 192.168.9.10:2181



Test为使用命令创建的容器，可以进行进行消息的生产和消费操作。

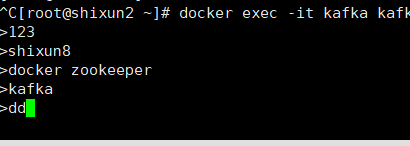
\_\_consumer\_offsets 是 Kafka 内部使用的特殊主题，用于存储消费者组的偏移量信息。

## 创建消息

注意：如果事先没有使用 kafka-topics 命令来手工创建 Topic，直接使用下面的内容进行消息创建时也会自动创建 Topics。

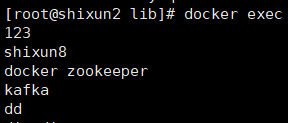
首先创建消息生产者。执行如下命令启动 Kafka 基于命令行的消息生产客户端，启动后可以直接在控制台中输入消息来发送，控制台中的每一行数据都会被视为一条消息来发送。

docker exec -it kafka kafka-console-producer.sh --broker-list 192.168.9.9:9092 --topic test



接着创建消息消费者。我们打开另一个命令窗口执行如下执行命令启动 Kafka 基于命令行的消息消费客户端，启动之后，马上可以在控制台中看到输出了之前我们在消息生产客户端中发送的消息。

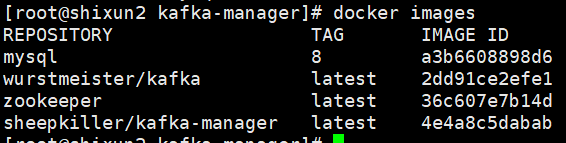
docker exec -it kafka kafka-console-consumer.sh --bootstrap-server 192.168.9.9:9092 --topic test --from-beginning



# kafka-manager部署

## 拉取镜像

命令:docker pull sheepkiller/kafka-manager



启动容器

命令：

docker run -itd -p 9000:9000 --name kafka-manager \

-e ZK\_HOSTS=192.168.9.10:2181 \

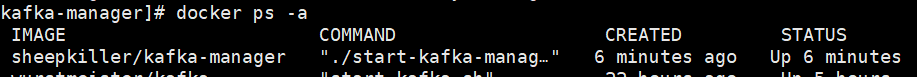
-e APPLICATION\_SECRET=letmein \

--network=my-macvlan-net --ip=192.168.9.11 \

sheepkiller/kafka-manager

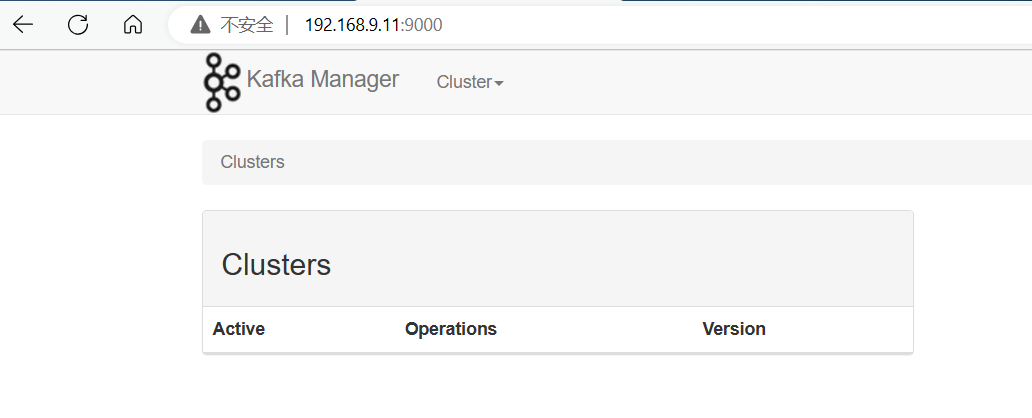
查看容器状态是否为UP

命令：docker ps -a



## kafka-manager 页面

浏览器访问ip：9000查看页面，出现以下界面则证明kafka-manage安装成功



# 集群化部署

## 创建文件存放目录

### 主节点：

# 创建 zookeeper 主节点配置存放目录

mkdir -p /mydata/zookeeper\_kafka/master/conf

# 创建 zookeeper 主节点数据存放目录

mkdir -p /mydata/zookeeper\_kafka/master/data

# 创建 zookeeper 主节点数据日志存放目录

mkdir -p /mydata/zookeeper\_kafka/master/datalog

# 创建 zookeeper 主节点日志存放目录

mkdir -p /mydata/zookeeper\_kafka/master/logs

### 从节点1：

# 创建 zookeeper 节点1 配置存放目录

mkdir -p /mydata/zookeeper\_kafka/node1/conf

# 创建 zookeeper 节点1 数据存放目录

mkdir -p /mydata/zookeeper\_kafka/node1/data

# 创建 zookeeper 节点1 数据日志存放目录

mkdir -p /mydata/zookeeper\_kafka/node1/datalog

# 创建 zookeeper 节点1 日志存放目录

mkdir -p /mydata/zookeeper\_kafka/node1/logs

### 从节点2：

# 创建 zookeeper 节点2 配置存放目录

mkdir -p /mydata/zookeeper\_kafka/node2/conf

# 创建 zookeeper 节点2 数据存放目录

mkdir -p /mydata/zookeeper\_kafka/node2/data

# 创建 zookeeper 节点2 数据日志存放目录

mkdir -p /mydata/zookeeper\_kafka/node2/datalog

# 创建 zookeeper 节点2 日志存放目录

mkdir -p /mydata/zookeeper\_kafka/node2/logs

## 修改配置文件：

# zookeeper 主节点配置存放目录

cd /mydata/zookeeper\_kafka/master/conf

# 编辑配置文件

vim zoo.cfg

# zookeeper 节点1 配置存放目录

cd /mydata/zookeeper\_kafka/node1/conf

# 编辑配置文件

vim zoo.cfg

# zookeeper 节点2 配置存放目录

cd /mydata/zookeeper\_kafka/node2/conf

# 编辑配置文件

vim zoo.cfg

### zoo.cfg详细信息

# Zookeeper保存数据的目录，默认情况下，Zookeeper将写数据的日志文件也保存在这个目录里

dataDir=/data

# 事物日志存储地点，如果没提供的话使用的则是 dataDir

dataLogDir=/datalog

# 服务器之间或客户端与服务器之间维持心跳的时间间隔，也就是每个 tickTime 时间就会发送一个心跳。tickTime以毫秒为单位

tickTime=2000

# 集群中的follower服务器(F)与leader服务器(L)之间初始连接时能容忍的最多心跳数（tickTime的数量）

initLimit=5

# 集群中的follower服务器与leader服务器之间请求和应答之间能容忍的最多心跳数（tickTime的数量）

syncLimit=2

# 默认值为3，不支持以系统属性方式配置。用于配置Zookeeper在自动清理的时候需要保留的快照数据文件数量和对应的事务日志文件。此参数的最小值为3，如果配置的值小于3会自动调整到3

autopurge.snapRetainCount=3

# 默认值为0，单位为小时，不支持以系统属性方式配置。用于配置Zookeeper进行历史文件自动清理的频率。如果配置为0或负数，表示不需要开启定时清理功能

autopurge.purgeInterval=0

# 默认为60，不支持以系统属性方式配置。从Socket层面限制单个客户端与单台服务器之间的并发连接数，即以ip地址来进行连接数的限制。

# 如果设置为0，表示不做任何限制。仅仅是单台客户端与单个Zookeeper服务器连接数的限制，不能控制所有客户端的连接数总和

maxClientCnxns=60

# 3.5.0中的新功能：当设置为false时，可以在复制模式下启动单个服务器，单个参与者可以使用观察者运行，并且群集可以重新配置为一个节点，并且从一个节点。

# 对于向后兼容性，默认值为true。可以使用QuorumPeerConfig的setStandaloneEnabled方法或通过将“standaloneEnabled = false”或“standaloneEnabled = true”添加到服务器的配置文件来设置它。

standaloneEnabled=false

# 内嵌的管理控制台，停用这个服务

admin.enableServer=false

# 开启四字命令，将所有命令添加到白名单中

4lw.commands.whitelist=\*

# 集群中服务的列表

server.1=192.168.9.12:2888:3888;2181

server.2=192.168.9.13:2888:3888;2181

server.3=192.168.9.14:2888:3888;2181

# 创建集群容器

## Zookeeper容器集群

### 节点1

# 启动命令

docker run -d --restart always \

--name zookeeper\_kafka\_master \

--network my-macvlan-net \

--ip 192.168.9.12 \

-p 2181:2181 \

-e ZOO\_MY\_ID=1 \

-v /mydata/zookeeper\_kafka/master/conf/zoo.cfg:/conf/zoo.cfg \

-v /mydata/zookeeper\_kafka/master/data:/data \

-v /mydata/zookeeper\_kafka/master/datalog:/datalog \

-v /mydata/zookeeper\_kafka/master/logs:/logs \

zookeeper

### 节点2

# 启动命令

docker run -d --restart always \

--name zookeeper\_kafka\_node1 \

--network my-macvlan-net \

--ip 192.168.9.13 \

-p 2182:2181 \

-e ZOO\_MY\_ID=2 \

-v /mydata/zookeeper\_kafka/node1/conf/zoo.cfg:/conf/zoo.cfg \

-v /mydata/zookeeper\_kafka/node1/data:/data \

-v /mydata/zookeeper\_kafka/node1/datalog:/datalog \

-v /mydata/zookeeper\_kafka/node1/logs:/logs \

zookeeper

### 节点三

# 启动命令

docker run -d --restart always \

--name zookeeper\_kafka\_node2 \

--network my-macvlan-net \

--ip 192.168.9.14 \

-p 2183:2181 \

-e ZOO\_MY\_ID=3 \

-v /mydata/zookeeper\_kafka/node2/conf/zoo.cfg:/conf/zoo.cfg \

-v /mydata/zookeeper\_kafka/node2/data:/data \

-v /mydata/zookeeper\_kafka/node2/datalog:/datalog \

-v /mydata/zookeeper\_kafka/node2/logs:/logs \

zookeeper

### 检查集群状态

步骤：

·在容器中开启一个交互模式的终端

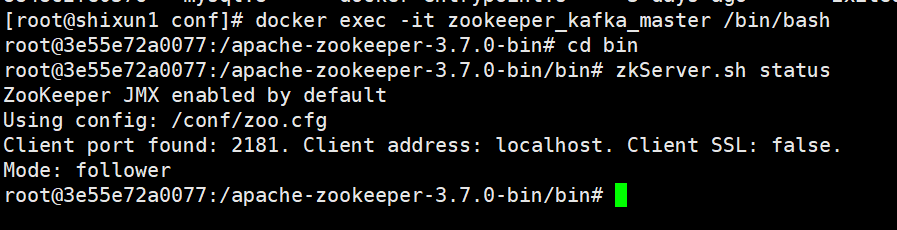
·查看 zookeeper 状态

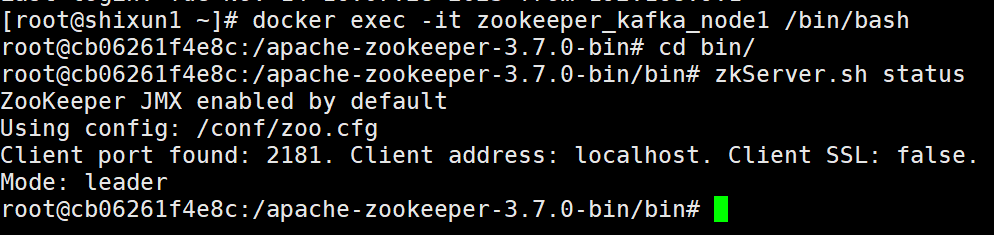
命令：docker exec -it 容器名 /bin/bash

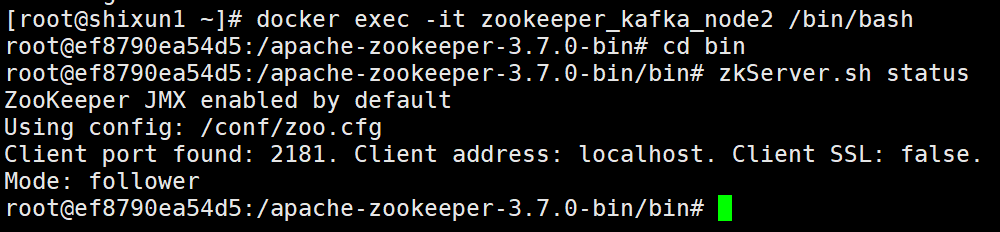
cd bin

zkServer.sh status

可以看到Mode参数的状态。Follower表示从节点，leader表示主节点。







## kafka容器集群

### 节点一

容器启动命令：

docker run -d --name kafka\_broker0 --network my-macvlan-net -p 9096:9096 \

-e KAFKA\_BROKER\_ID=0 \

-e KAFKA\_ZOOKEEPER\_CONNECT=\

192.168.9.12:2181,192.168.9.13:2181,192.168.9.14:2181 \

-e KAFKA\_ADVERTISED\_LISTENERS=PLAINTEXT://192.168.9.15:9096 \

-e KAFKA\_LISTENERS=PLAINTEXT://0.0.0.0:9096 \

--ip 192.168.9.15 \

wurstmeister/kafka

### 节点二

容器启动命令：

docker run -d --name kafka\_broker1 --network my-macvlan-net -p 9097:9096 \

-e KAFKA\_BROKER\_ID=1 \

-e KAFKA\_ZOOKEEPER\_CONNECT=\

192.168.9.12:2181,192.168.9.13:2181,192.168.9.14:2181 \

-e KAFKA\_ADVERTISED\_LISTENERS=PLAINTEXT://192.168.9.16:9096 \

-e KAFKA\_LISTENERS=PLAINTEXT://0.0.0.0:9096 \

--ip 192.168.9.16 \

wurstmeister/kafka

### 节点三

容器启动命令：

docker run -d --name kafka\_broker2 --network my-macvlan-net -p 9098:9096 \

-e KAFKA\_BROKER\_ID=2 \

-e KAFKA\_ZOOKEEPER\_CONNECT=\

192.168.9.12:2181,192.168.9.13:2181,192.168.9.14:2181 \

-e KAFKA\_ADVERTISED\_LISTENERS=PLAINTEXT://192.168.9.17:9096 \

-e KAFKA\_LISTENERS=PLAINTEXT://0.0.0.0:9096 \

--ip 192.168.9.17 \

wurstmeister/kafka

参数解释：

·KAFKA\_BROKER\_ID: 设置 Kafka Broker 的 ID 。每个 Broker 都需要有一个唯一的 ID。

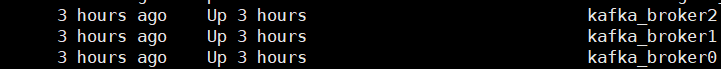
·-e KAFKA\_ZOOKEEPER\_CONNECT=指定连接到 ZooKeeper 的地址和端口。这里使用了一个逗号分隔的 ZooKeeper 地址列表。

·-e KAFKA\_ADVERTISED\_LISTENERS: 设置 Kafka Broker 的广告监听器，用于向客户端公布 Broker 的访问地址。

·-e KAFKA\_LISTENERS=PLAINTEXT://0.0.0.0:9096: 设置 Kafka Broker 监听的地址和端口。PLAINTEXT://0.0.0.0:9096 表示 Kafka Broker 将监听所有网络接口上的 9096 端口。

### 检查集群状态

命令：docker ps -a



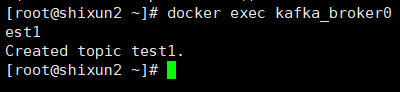
容器状态为UP表示容器启动成功

# 测试：

## 创建Topic

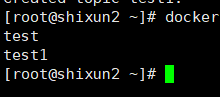
使用其中一个容器创建一个topic：

命令：docker exec kafka\_broker0 kafka-topics.sh --create --zookeeper 192.168.9.12:2181,192.168.9.13:2181,192.168.9.14:2181 --replication-factor 3 --partitions 3 --topic test



查看创建的topic

命令：docker exec kafka\_broker0 kafka-topics.sh --list --zookeeper 192.168.9.12:2181,192.168.9.13:2181,192.168.9.14:2181

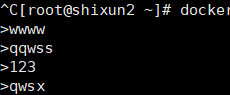


## 生产者与消费者创建

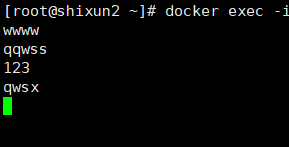
创建生产者命令：docker exec -it kafka\_broker1 kafka-console-producer.sh --broker-list 192.168.9.15:9096,192.168.9.16:9096,192.168.9.17:9096 --topic test1

创建消费者命令：docker exec -it kafka\_broker1 kafka-console-consumer.sh --bootstrap-server 192.168.9.15:9096,192.168.9.16:9096,192.168.9.17:9096 --topic test1 --from-beginning

打开生产者输入几条测试数据



打开另一个窗口，查看消费者记录是否同步在生产者发送的消息。



# 集群的kafka-manage部署

创建容器

命令：docker run -itd --name=kafka-manager-jiqun -p 9500:9000 \

-e ZK\_HOSTS="192.168.9.12:2181,192.168.9.13:2181,192.168.9.14:2181" \

-v /usr/local/kafka-manager/data/:/var/lib/zookeeper \

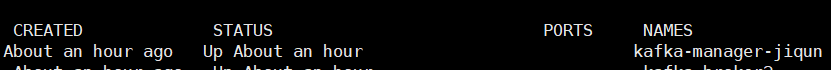
-v /usr/local/kafka-manager/logs:/kafka/kafka-logs \

-e APPLICATION\_SECRET=letmein \

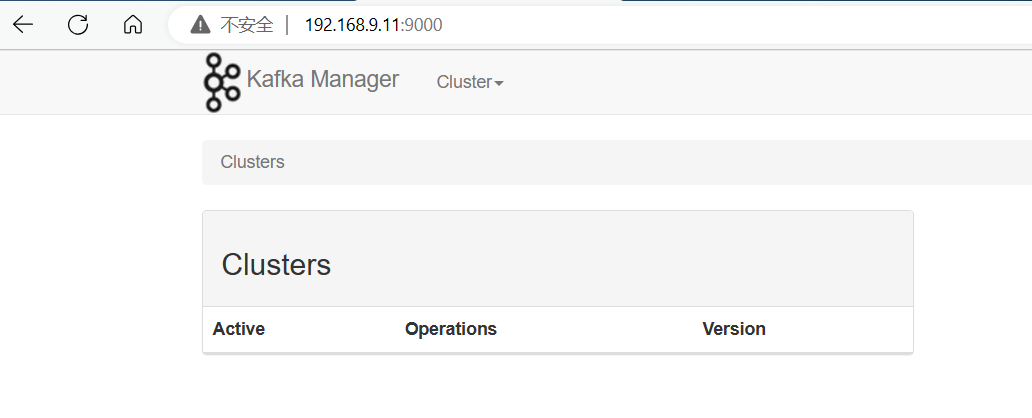
--network my-macvlan-net --ip 192.168.9.18 \

sheepkiller/kafka-manager

查看容器状态，UP状态为启动成功

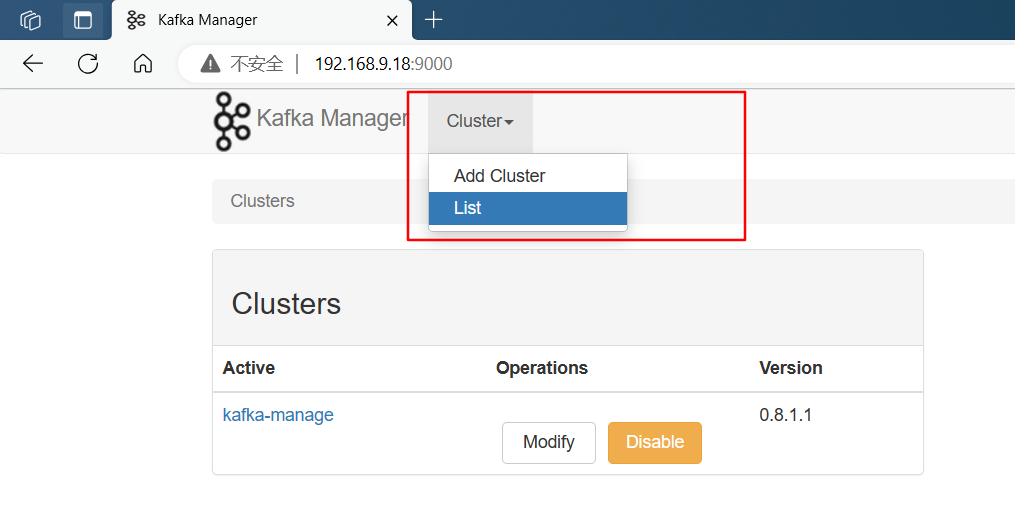


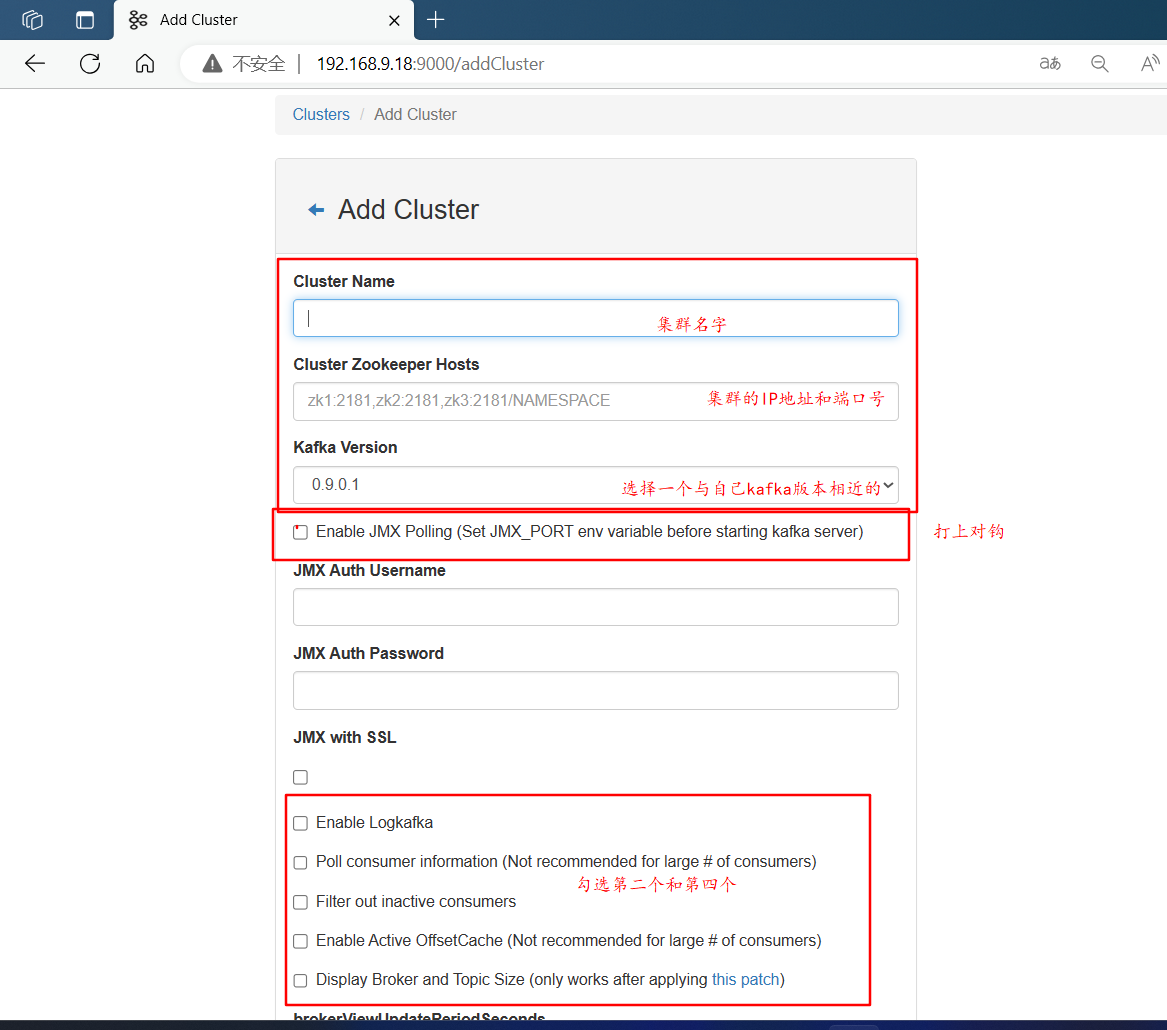
浏览器访问



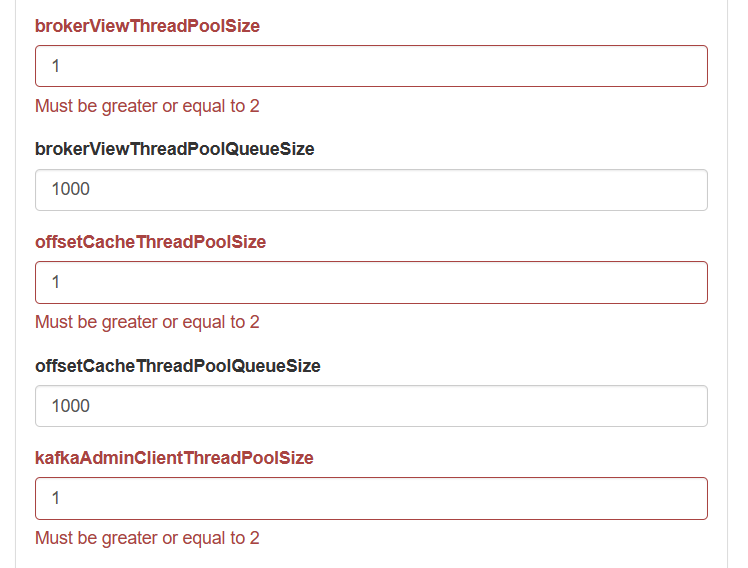
### 添加集群、修改集群配置

点击Cluster,再点击ADD Cluster

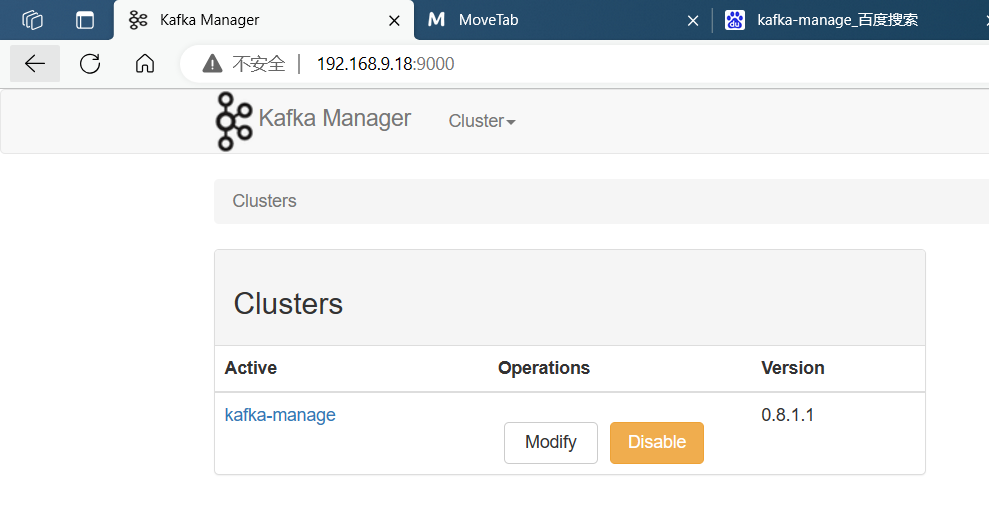




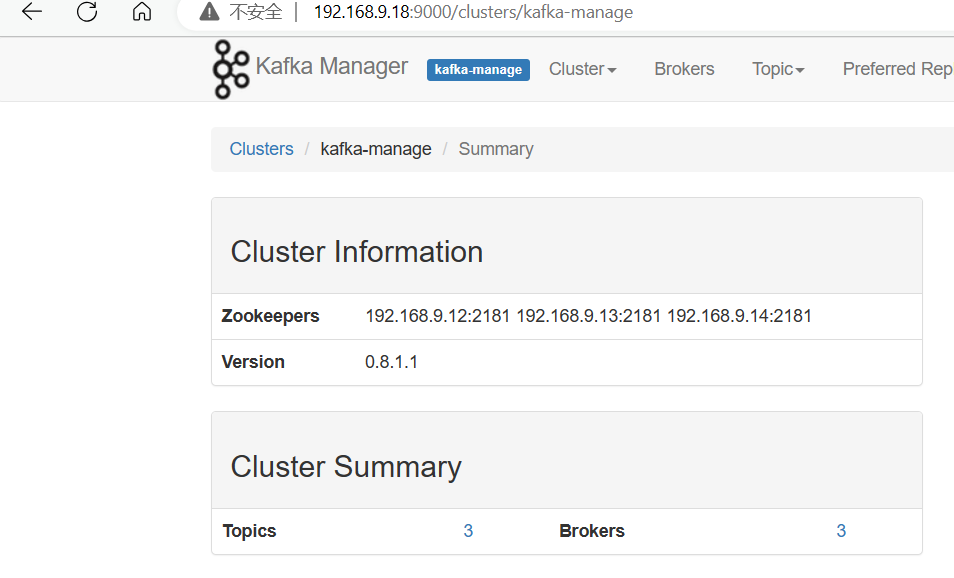
下面的选项要求参数必须大于等于2

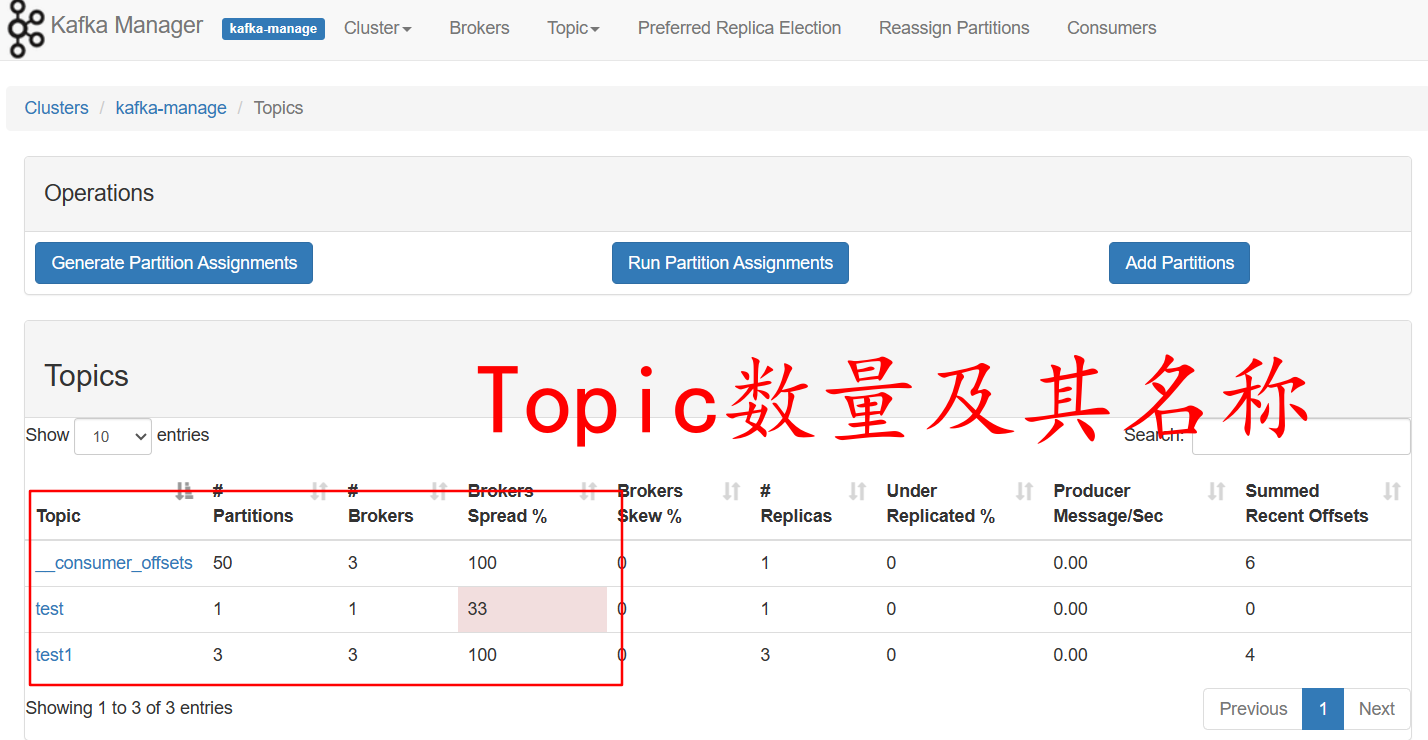


修改完后点击save，在主页可以查看已经添加成功的集群

查看集群详细信息

### 查看集群，点击kafka-manage





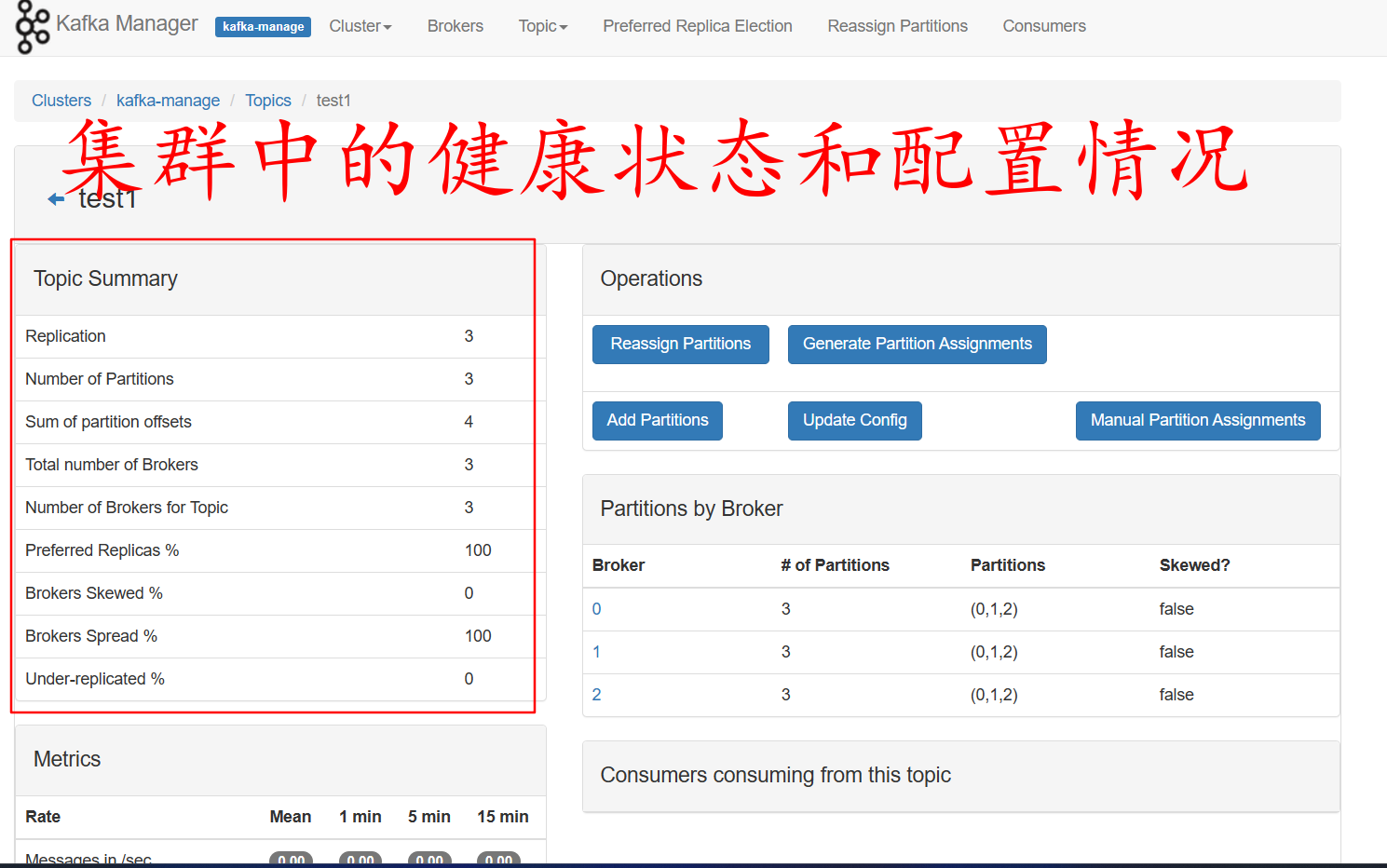
### 点击test可以查看topic的详细信息

·复制因子（Replication）：每个分区的复制因子为 3。这意味着每个分区将在 Kafka 集群中的三个 Broker 上进行复制，以提高数据的冗余和可用性。

·Topic 的 Brokers 数量（Number of Brokers for Topic）：主题使用了 3 个 Broker。这意味着主题的分区在整个集群中均匀地分布在这些 Broker 上。

·Brokers 偏斜百分比（Brokers Skewed %）：Brokers 偏斜的百分比为 0。这表示在 Kafka 集群中，没有 Broker 承担过多的负载，分布相对均衡。

·Brokers 分布百分比（Brokers Spread %）：Brokers 分布的百分比为 100。这表示 Broker 实例在集群中均匀地分布，有助于负载均衡和高可用性。



### 查看Brokers

