**K8s学习**

# K8s发展历史

## 资源管理器对比

1. Mesos

采用apache开源协议，开源的分布式的资源管理框架，在2019.5不再使用，转向k8s

1. Docker swarm

分布式资源管理框架，针对docker容器化，在2019.7阿里云不再使用docker swarm

1. K8s

是Google旗下采用go语言基于borg系统开发的一款资源管理器kubernets

# K8s组件说明

## borg系统架构

|  |
| --- |
| 1575710933(1) |

Borg系统架构组件说明

1. BorgMaster：负责请求分发，为了防止BorgMaster挂了，borgMaster集群，节点数量 为单数
2. Borglet：真正执行请求
3. 请求方式：web browsers(浏览器)、command-line tools(命令行)、borgcfg(文件的读取)

当通过这些方式去发起请求的时候，会先经过BorgMaster进行请求分发，然后到Borglet 去执行请求

1. Scheduler(调度器)：scheduler讲数据写入paxos数据库(google的一个键值对数据库)， borglet会实时的去paxos数据库里面进行监听，如果发现有对应的请求，就会获取并进 行执行

## K8s架构

|  |
| --- |
| 1575711693(1) |

k8s架构组件说明

1. scheduler(调度器)：负责接受任务，选择合适的节点进行分配任务
2. kube-controller-manager

Node controller：节点控制器，负责在节点出现故障时进行通知和响应

replication controller：副本控制器，负责为系统中的每个副本控制器对象维护正确数量的pod

Endpoint controller：端点控制器

1. api server：

所有服务访问的统一入口，对外暴露k8s的api接口，是外界进行资源操作的唯一入口

提供认证、授权、访问控制、API注册和发现等机制

1. etcd：一个可信赖的(不会存在单点故障，天生支持集群化)分布式键值(kv)存储服务， 它能够为整个分布式集群存储一些关键数据，协助分布式集群的正常运转。

k8s使用etcd作为持久化方案(v2,v3两个版本)，v2版本已在k8s版本v1.11中弃用

|  |
| --- |
| 1575712254(1) |

etcd内部架构图

1. 采用http协议，k8s也是采用http协议，因为http协议支持多种操作方式
2. Ralt：存放所有读写信息，并且为了防止信息出现损坏，会将日志信息写入本地磁 盘中(store)
3. node节点：node节点需要安装三个组件(kubelet：和docker进行交互，操作docker 创建对应的容器即实现容器的生命周期管理、kube proxy：负责写入规则至iptables或者ipvs来实现服务映射访问、docker)

## K8s其它插件说明

1. CoreDNS：可以为集群中的svc创建一个域名ip的对应关系解析
2. Dashboard：给k8s提供一个B/S结构的访问体系
3. Ingress Controller：官方只能实现4层代理，Ingress可以实现7层代理
4. Federation：提供一个可以跨集群中心多k8s统一管理功能
5. Prometheus(普罗米修斯)：提供一个k8s集群的监控能力
6. Elk：提供k8s集群日志统一分析接入平台

# k8s基础概念

## Volume

声明在pod容器中可访问的文件目录

可以被挂载在pod中一个或多个容器指定路径下

支持多种后端存储抽象(本地存储、分布式存储、云存储)

## 网络通信方式

# mall中k8s使用

# k8s集群安装部署(1个master节点、2个node节点)

## 前置要求(服务器)

2GB或更多的RAM，2个CPU或者更多、30G磁盘或者更多

可以访问外网、需要拉取镜像

禁止swap分区

## 部署步骤

在所有节点上安装docker和kubeadm

部署kubernetes master节点

部署容器网络插件

部署kubernetes node节点，将节点加入kubernetes集群中

部署dashboard web页面，可视化查看kubernetes资源

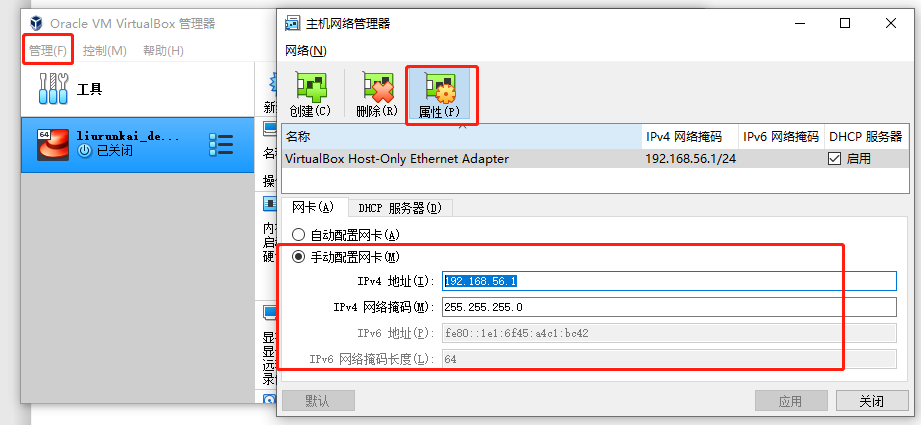
## 环境准备

### 设置virtualBox环境

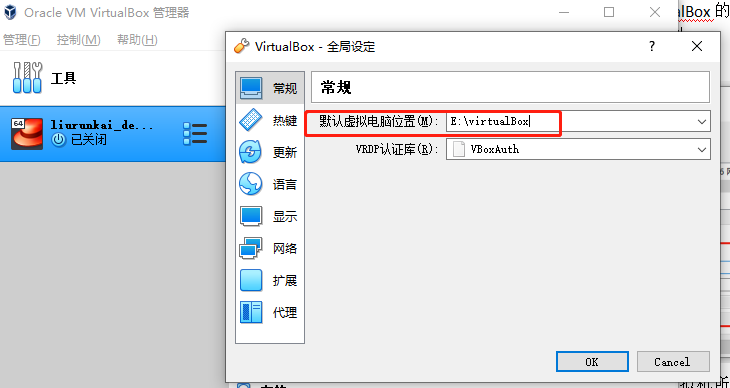
使用vagrant快速创建3个虚拟机，虚拟机启动前先设置virtualBox的主机网络

现全部统一为192.168.56.1，以后所有虚拟机都是56.x的ip地址

选中管理------->主机网络管理器

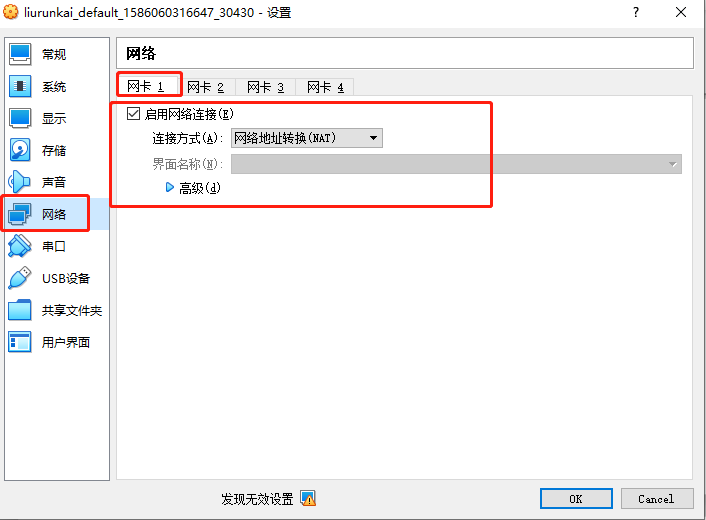


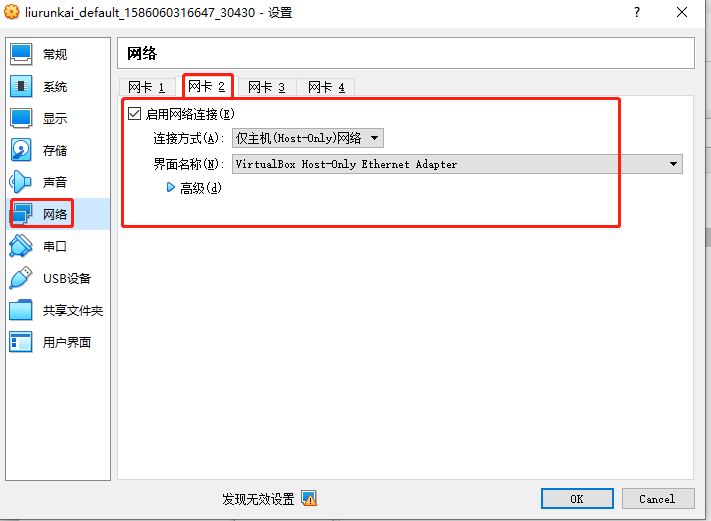
选中管理---------->全局设定，设置默认虚拟电脑位置用来存储虚拟机所有文件



选中安装好的虚拟机，设置网络里面的网卡1和网卡2







### 1.3.2 Pod

K8s使用pod来组织一组容器

一个pod中的所有容器共享同一网络

Pod是k8s中的最小部署单元

## K8s的特点及优势

1. 轻量级：消耗资源小
2. 开源
3. 弹性(水平)伸缩：服务器动态增减而不需要重启服务
4. 服务发现和负载均衡：采用IPVS
5. 自动部署和回滚
6. 自动编排：可以将应用部署到合适的服务器
7. 自动修复：如果一个服务器不能响应，k8s会将该服务器上的应用重新部署到其它服务器，从而可以保证应用正常工作

### 使用vagrant创建3台虚拟机

Vagrantfile：非中文、无空格的文件夹下

|  |
| --- |
| Vagrant.configure("2") do |config|  (1..3).each do |i|  config.vm.define "k8s-node#{i}" do |node|  # 设置虚拟机的Box  node.vm.box = "centos/7"  # 设置虚拟机的主机名  node.vm.hostname="k8s-node#{i}"  # 设置虚拟机的IP  node.vm.network "private\_network", ip: "192.168.56.#{99+i}", netmask: "255.255.255.0"  # 设置主机与虚拟机的共享目录  # node.vm.synced\_folder "~/Documents/vagrant/share", "/home/vagrant/share"  # VirtaulBox相关配置  node.vm.provider "virtualbox" do |v|  # 设置虚拟机的名称  v.name = "k8s-node#{i}"  # 设置虚拟机的内存大小  v.memory = 4096  # 设置虚拟机的CPU个数  v.cpus = 4  end  end  end  end |



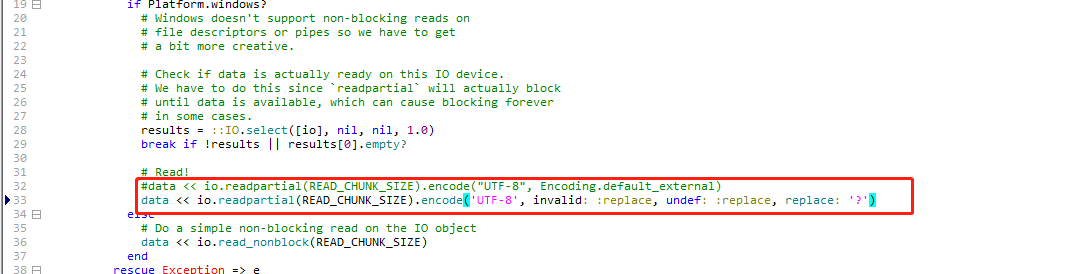
运行vagrant up命令，输出k8s-node1: Rsyncing folder: /cygdrive/d/vmware/ => /vagrant，说明k8s-node1安装成功

安装过程中出现的问题：

io.rb:32:in `encode': "\xA3\x85" from GBK to UTF-8 (Encoding::UndefinedConversionError)

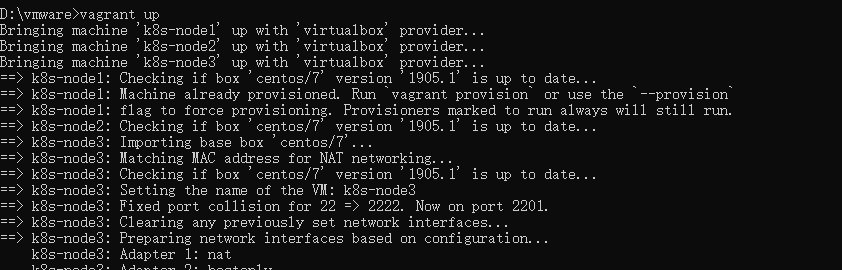
解决方式：将io.rb中第32行注释，添加如下内容：

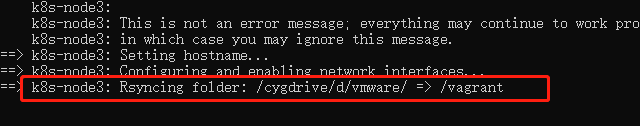
|  |
| --- |
| data << io.readpartial(READ\_CHUNK\_SIZE).encode('UTF-8', invalid: :replace, undef: :replace, replace: '?') |



关闭cmd命令窗口，重新打开继续运行vagrant up命令，输出k8s-node2: Rsyncing folder: /cygdrive/d/vmware/ => /vagrant，说明k8s-node2安装成功

关闭cmd命令窗口，重新打开继续运行vagrant up命令，输出k8s-node3: Rsyncing folder: /cygdrive/d/vmware/ => /vagrant，说明k8s-node3安装成功



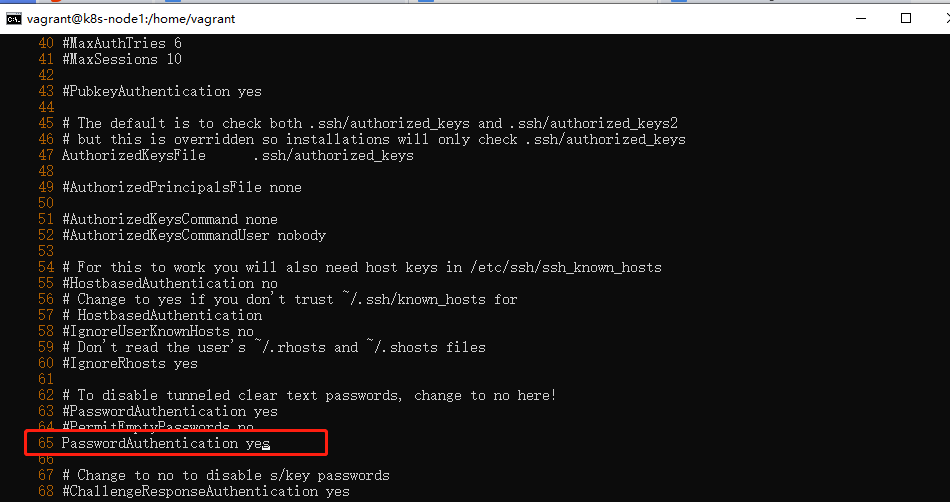




### 虚拟机默认ssh连接，修改配置允许账号密码访问

在Vagrantfile所在目录运行cmd命令，vagrant ssh k8s-node1/2/3进行连接

切换root用户，密码默认vagrant，修改配置文件/etc/ssh/sshd\_config允许账号密码访问，将65行的no改为yes，重启sshd服务：service sshd restart，使用exit命令退出



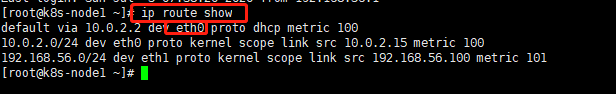


### 使用xshell连接k8s-node1/2/3

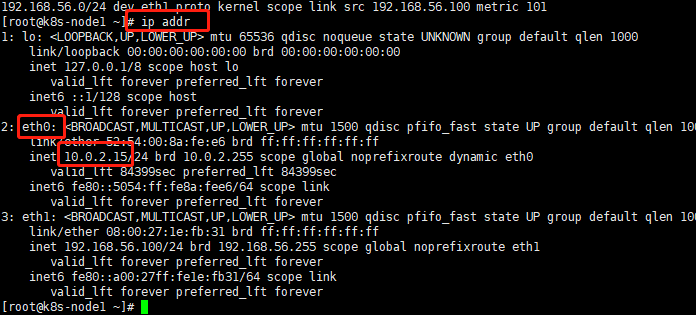
ip分别为192.168.56.100、192.168.56.101、192.168.56.102

账号密码为root、vagrant

使用ip route show 查看默认网卡，默认为eth0

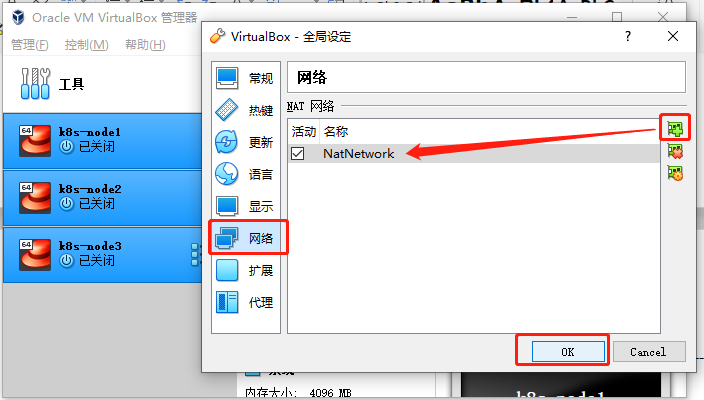


查看eth0网卡的ip地址：ip addr，都是10.0.2.15

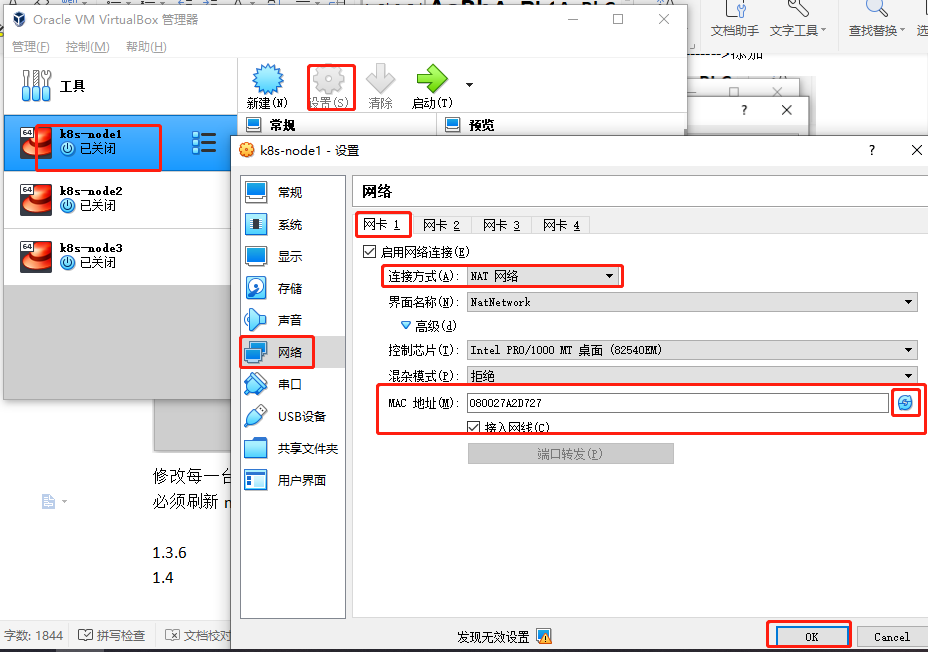


### 设置好net网络，修改k8s-node1/2/3默认eth0的ip地址

先将3台虚拟机全部关闭电源，选中管理---------->全局设定---------->网络--------->添加

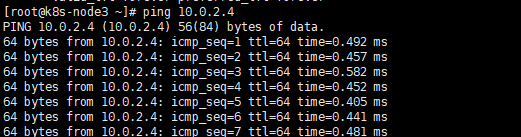


修改每一台虚拟机的网卡1的网络设置，由网络地址转换改为NAT网络，点开高级，必须刷新mac地址



重新查看默认网卡和网卡的ip地址：ip route show、ip addr

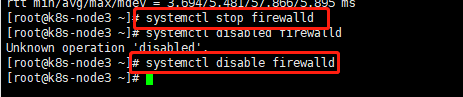
可以看到默认网卡eth0的ip地址变了，查看3台虚拟机是否连通ping，查看是否连通外网ping www.baidu.com



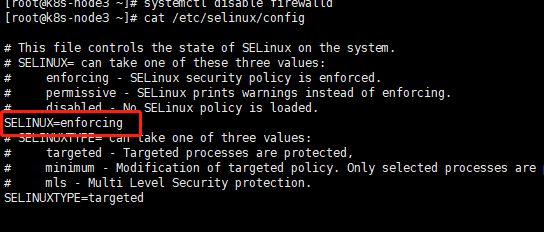
### 关闭防火墙

systemctl stop firewalld

systemctl disabled firewalld



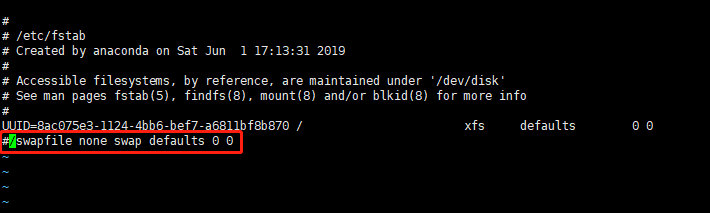
### 修改selinux的安全规则



### 关闭swap

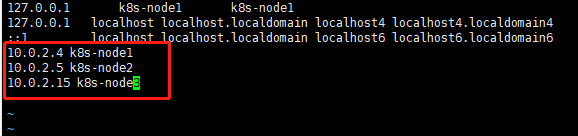
临时：swapoff -a

永久：将/etc/fstab最后一行注释掉



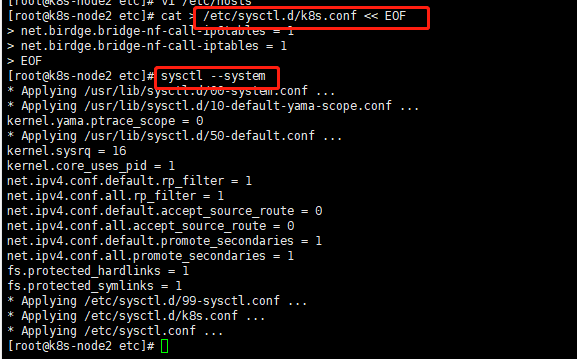
### 添加主机名与默认网卡eth0的ip对应关系

|  |
| --- |
| 10.0.2.4 k8s-node1  10.0.2.5 k8s-node2  10.0.2.15 k8s-node3 |



### 将桥接的IPV4流量传递到iptables的链

|  |
| --- |
| cat > /etc/sysctl.d/k8s.conf << EOF  net.birdge.bridge-nf-call-ip6tables = 1  net.birdge.bridge-nf-call-iptables = 1  EOF |
| sysctl --system |

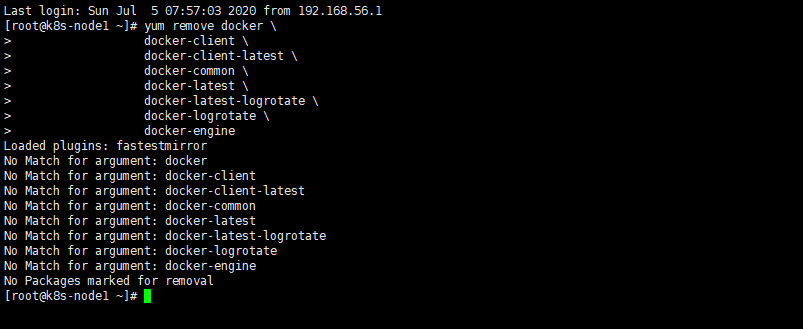


## 所有节点都安装docker、kubeadm、kubelet、kubectl

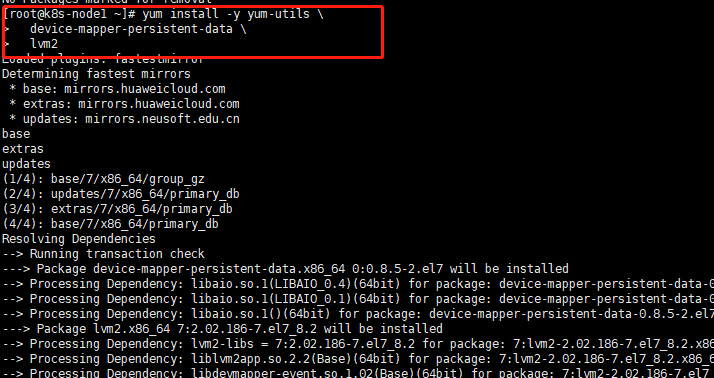
### 安装docker

使用docker作为基础镜像

1. 先删除老版本docker



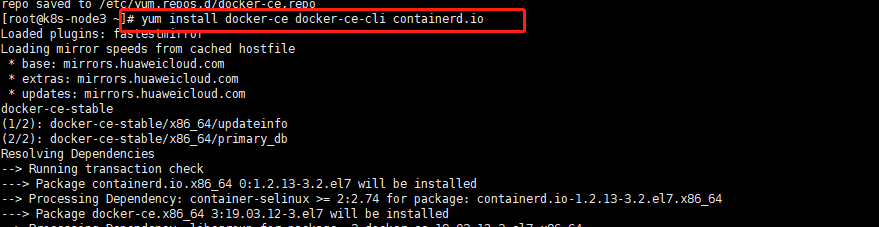
1. 安装docker-ce



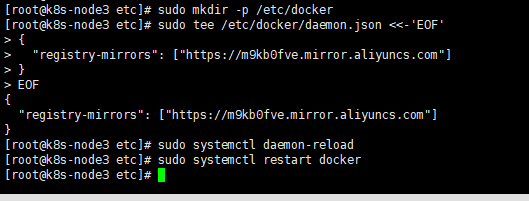
1. 设置docker repo的yum位置



1. 安装docker



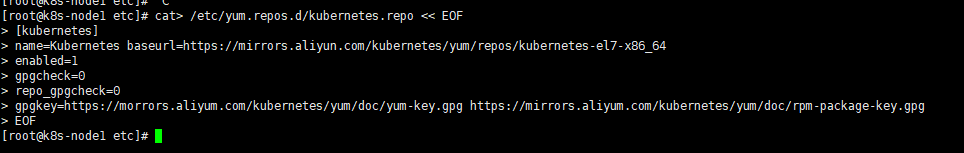
1. 配置docker阿里云镜像加速



1. 设置docker开机自启



### 添加阿里云yum源



|  |
| --- |
| cat> /etc/yum.repos.d/kubernetes.repo << EOF  [kubernetes]  name=Kubernetes baseurl=https://mirrors.aliyun.com/kubernetes/yum/repos/kubernetes-el7-x86\_64  enabled=1  gpgcheck=0  repo\_gpgcheck=0  gpgkey=https://morrors.aliyum.com/kubernetes/yum/doc/yum-key.gpg https://mirrors.aliyum.com/kubernetes/yum/doc/rpm-package-key.gpg  EOF  setenforce 0 |
| #这个下面的不执行，跟上面是一样的，这个是从网上找的，因为最初执行完上面之后install报错，可能是没加setenforce 0这一句的原因  #cat <<EOF > /etc/yum.repos.d/kubernetes.repo  #[kubernetes]  #name=Kubernetes  #baseurl=https://mirrors.aliyun.com/kubernetes/yum/repos/kubernetes-el7-x86\_64/  #enabled=1  #gpgcheck=1  #repo\_gpgcheck=1  #gpgkey=https://mirrors.aliyun.com/kubernetes/yum/doc/yum-key.gpg #https://mirrors.aliyun.com/kubernetes/yum/doc/rpm-package-key.gpg  #EOF  #setenforce 0  # 默认安装最新版kubelet kubeadm kubectl  #yum install -y kubelet kubeadm kubectl  # 设置kubelet开机启动  #systemctl enable kubelet  # 启动kubelet  #systemctl start kubelet |

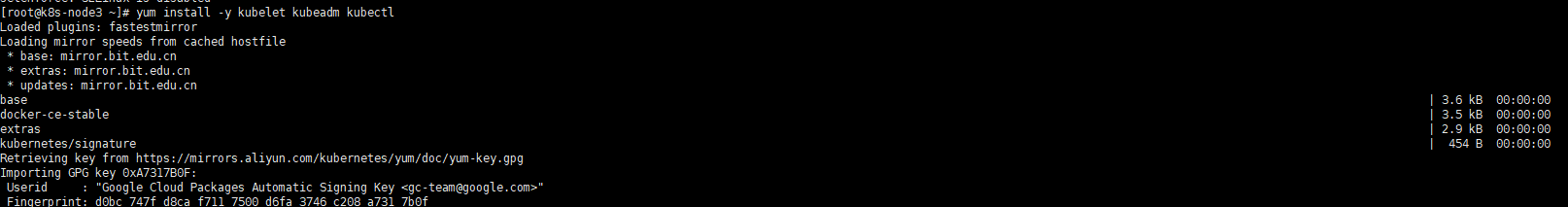
### 安装kubeadm、kubelet、kubectl

使用kubeadm安装k8s的master和node节点

Kubelet管理pod

Kubectl命令行

|  |
| --- |
| yum install -y kubelet-1.17.3 kubeadm-1.17.3 kubectl-1.17.3 |



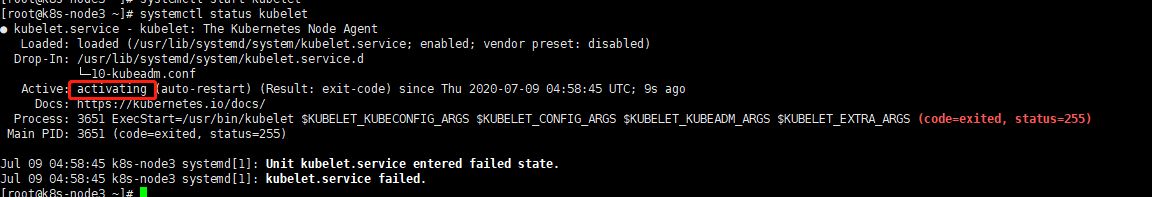
### 设置kubelet开启启动和启动kubelet

|  |
| --- |
| systemctl enable kubelet  systemctl start kubelet |



### 查看kubelet启动状态，现在是启动不起来的activating

|  |
| --- |
| systemctl status kubelet |



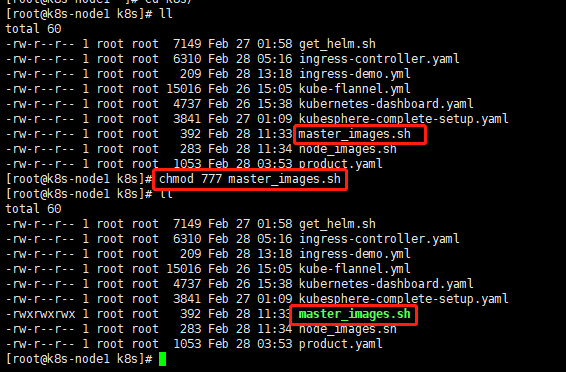
## 安装master和node节点

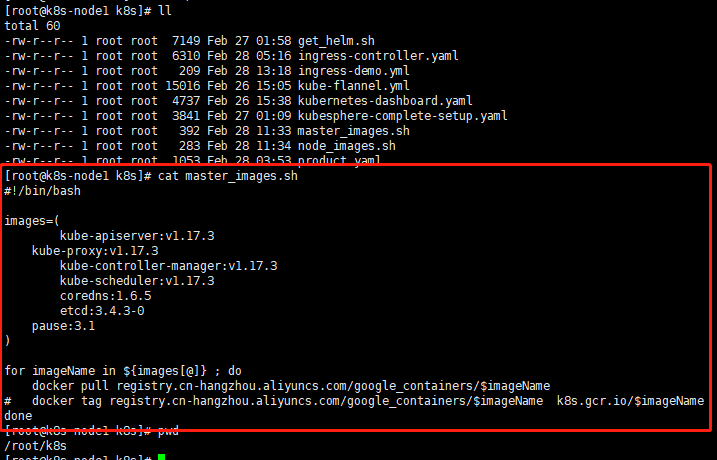
### 初始化master节点

K8s-node1作为master节点，K8s-node2和k8s-node3作为node节点，

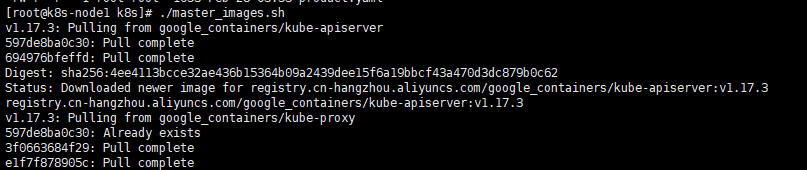
|  |
| --- |
| 因为k8s默认下载镜像地址为k8s.gcr.io，国内无法访问，指定阿里云镜像仓库地址，可以先执行master\_images.sh下载镜像，然后执行初始master节点的命令 |

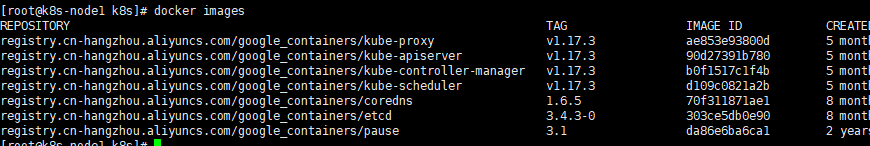
1. 在k8s-node1上传master\_images.sh并授权：chmod 777 master\_images.sh





1. 执行./master\_images.sh下载镜像





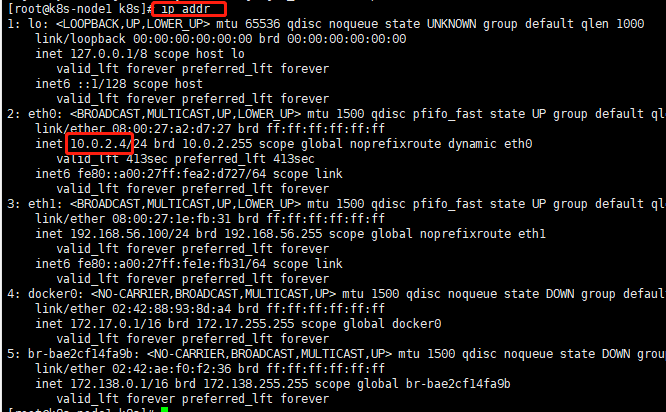
1. master节点初始化

--apiserver-advertise-address=10.0.2.4：这个要写eth0的地址，不能写成eth1的192.168.56.100地址，可通过ip addr查看

--image-repository registry.aliyuncs.com/google\_containers：镜像下载地址

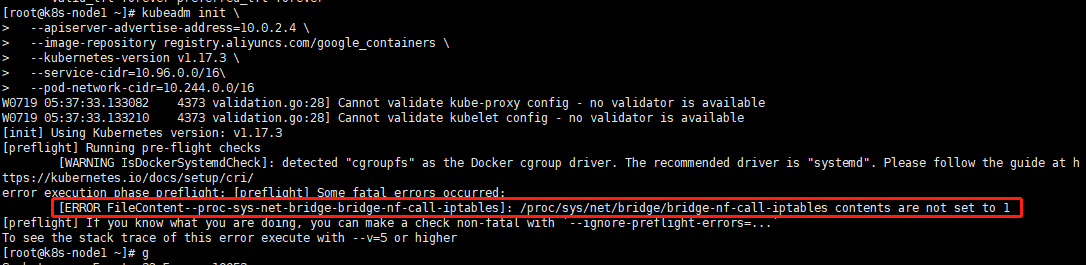
--service-cidr=10.96.0.0/16：service之间的通信地址

--pod-network-cidr=10.244.0.0/16：pod之间的地址



|  |
| --- |
| kubeadm init \  --apiserver-advertise-address=10.0.2.4 \  --image-repository registry.aliyuncs.com/google\_containers \  --kubernetes-version v1.17.3 \  --service-cidr=10.96.0.0/16\  --pod-network-cidr=10.244.0.0/16 |

在执行上面命令的时候，可能会报错如下：

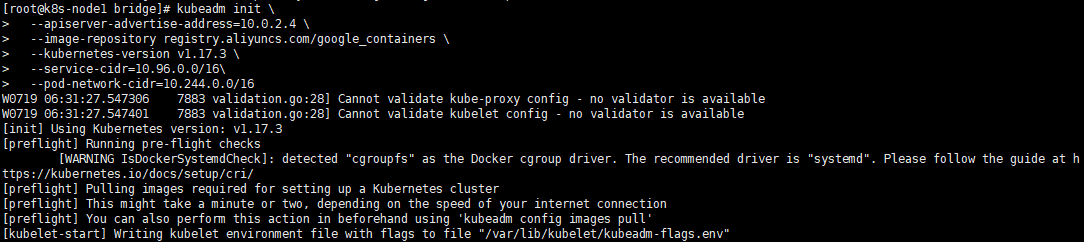


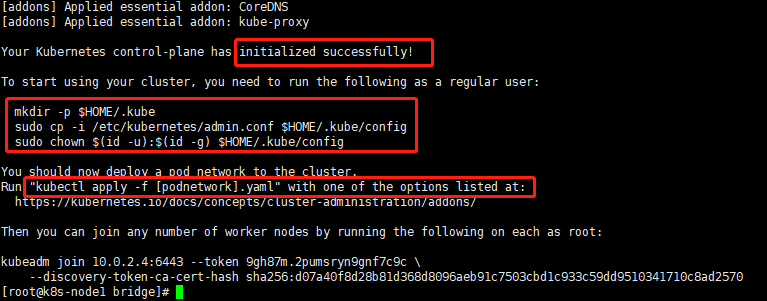
解决方式：

echo "1" >/proc/sys/net/bridge/bridge-nf-call-iptables



初始化成功后：





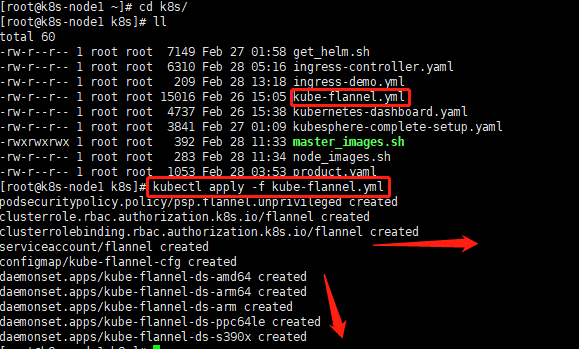
1. 执行初始化成功之后所提示的命令

|  |
| --- |
| mkdir -p $HOME/.kube  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config  sudo chown $(id -u):$(id -g) $HOME/.kube/config |



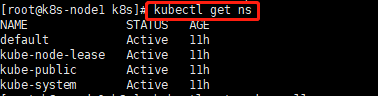
1. 安装flannel网络

kubectl apply -f kube-flannel.yml



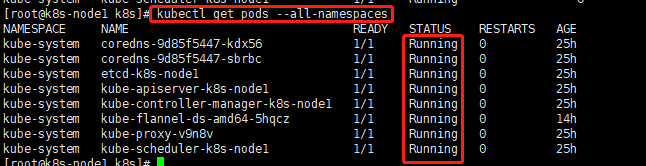
1. 查看名称空间及pods

kubectl get ns



kubectl get pods --all-namespaces

要等所有的status为running状态，flannel网络才算准备好



1. 查看所有节点

kubectl get nodes



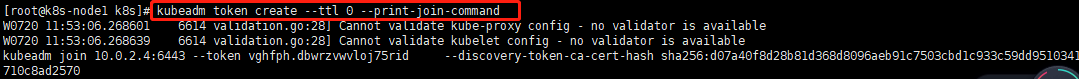
### 1.5.2 初始化nodes节点

（1）在k8s-node2和k8s-node3节点执行初始化完master节点的命令kubeadm join命令



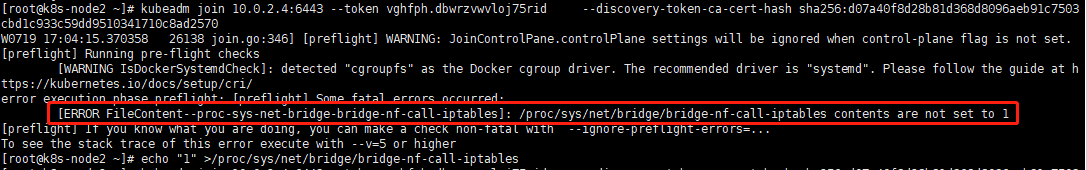
这个命令默认2个小时内有效，如果超过两个小时，需要使用如下命令重新生成

kubeadm token create --ttl 0 --print-join-command：永久有效



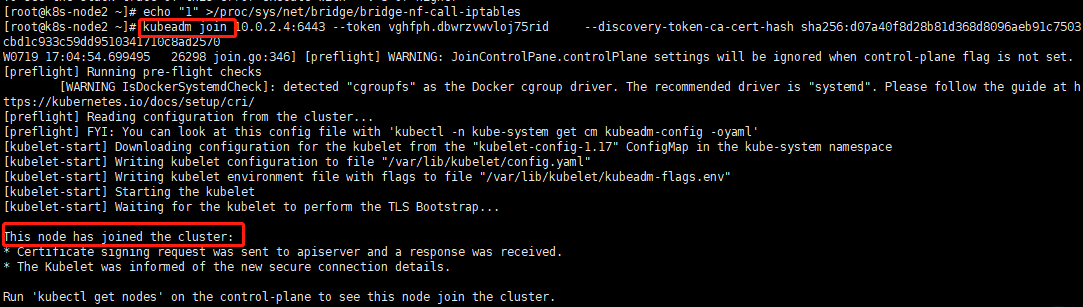
1. 执行kubeadm join命令

在执行kubeadm join的过程中也可能报跟初始化master节点同样的错误：



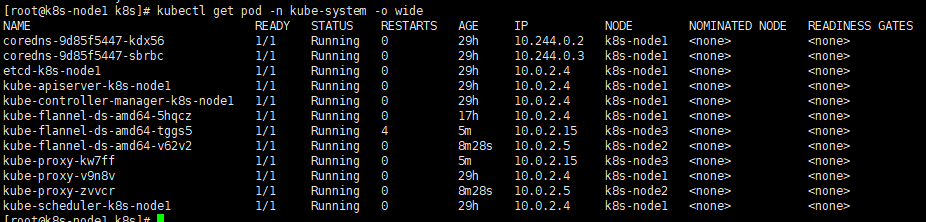
解决方式：echo "1" >/proc/sys/net/bridge/bridge-nf-call-iptables

再次执行kubeadm join命令

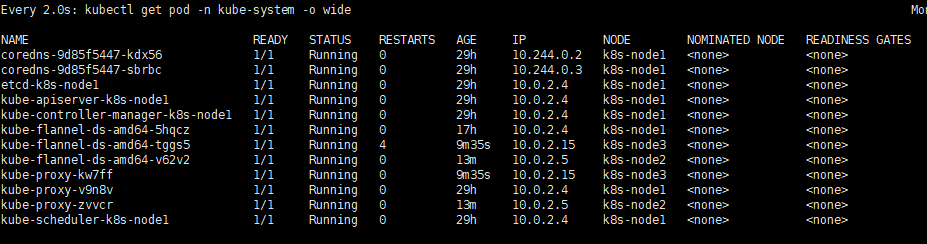


1. 查看pod状态

kubectl get pods -n kube-system -o wide

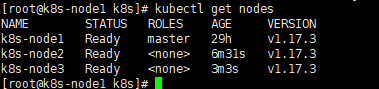


watch kubectl get pods -n kube-system -o wide：监控pod



1. 再次查看nodes节点，直到均变为ready状态

kubectl get nodes/kubectl get no



# K8s入门-基本操作

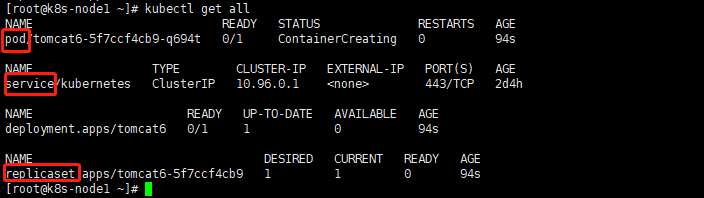
## 2.1 使用k8s部署一个tomcat

（1）kubectl create deployment tomcat6 --image=tomcat:6.0.53-jre8



1. 获取k8s所有资源

kubectl get all



1. 查看pod部署在哪个节点

kubectl get pods -o wide



说明部署在k8s-node3节点，查看k8s-node3节点的镜像及运行的容器



1. 模拟tomcat停了，会重新拉起一个tomcat，保证副本数有一个在运行



1. 模拟k8s-node3宕机(将k8s-node3机器关机)，k8s会在k8s-node2节点重新拉起一个tomcat，保证副本数有一个在运行

## 2.2 暴露tomcat6服务

（1）暴露服务

kubectl expose deployment tomcat6 --port=80 --target-port=8080 --type=NodePort

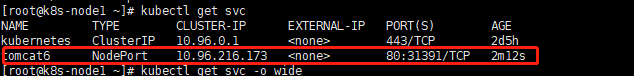
--port=80：pod的端口

--target-port=8080：pod内容器的端口，即tomcat6的端口



1. 查看服务

kubectl get svc



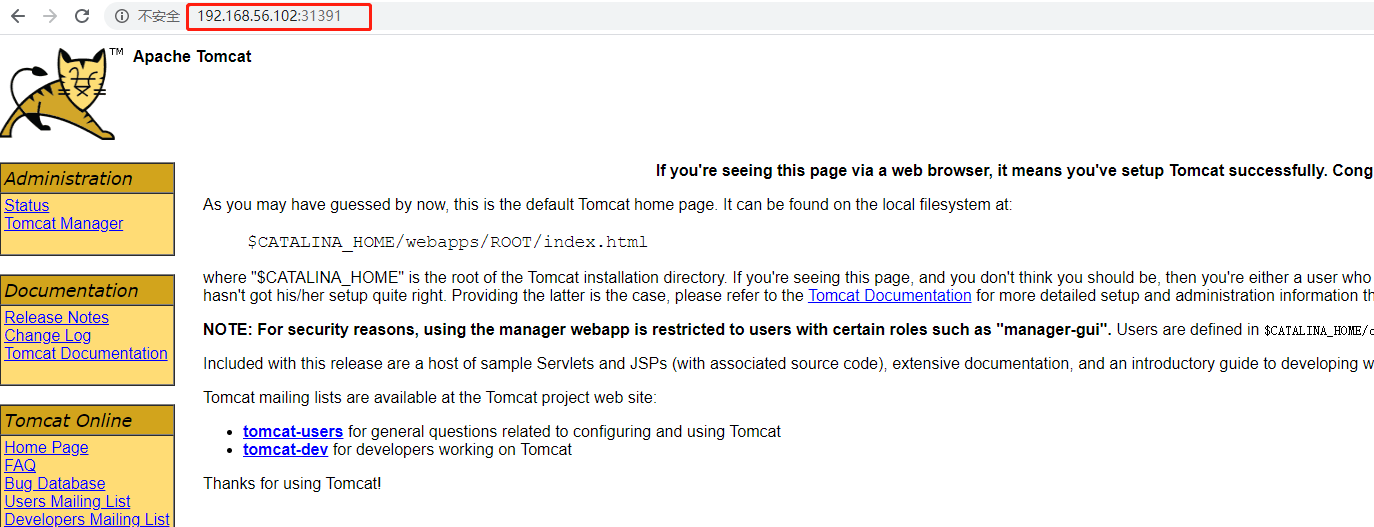
可以看到tomcat6已经暴露服务，端口为31391

1. 访问tomcat服务

<http://192.168.56.100:31391/>

[http://192.168.56.101:31391/](http://192.168.56.100:31391/)

[http://192.168.56.102:31391/](http://192.168.56.100:31391/)



## 2.3 动态扩容

（1）扩容、缩容

kubectl scale --replicas=3 deployment tomcat6.

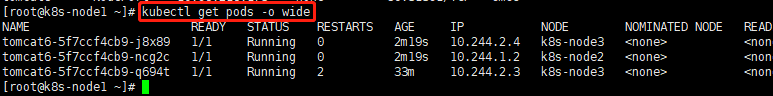


1. 查看pod部署情况

kubectl get pods 查看默认名称空间default下的pod

kubectl get pods --all-namespaces 查看所有名称空间下的pod

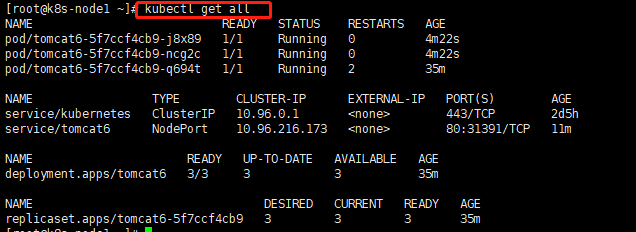
kubectl get pods -o wide



## 2.4 删除pod、deployment、service

（1）先查看k8s所有资源

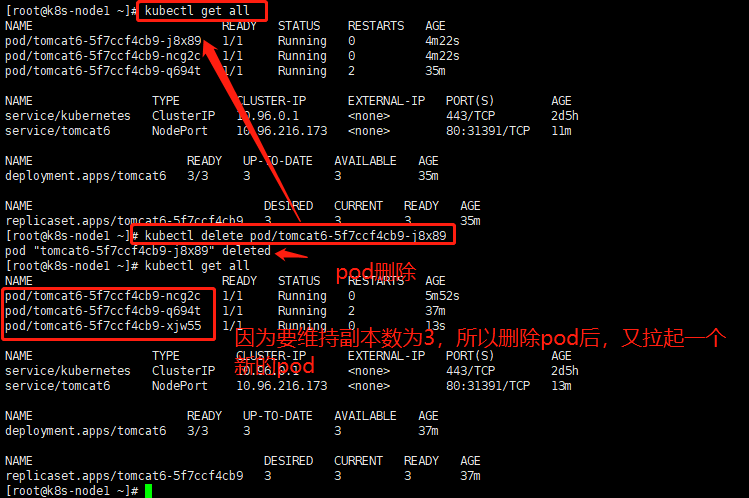
kubectl get all



1. 删除pod

kubectl delete pod/tomcat6-5f7ccf4cb9-j8x89

kubectl delete pod名字



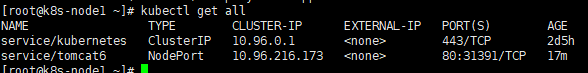
1. 删除deployment

删除deployment会将pod也删除，只剩下service

kubectl delete deployment.apps/tomcat6



kubectl delete deployment名字



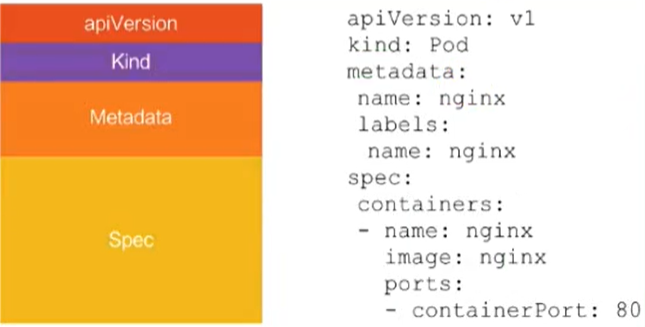
1. 删除service

kubectl delete service/tomcat6



## 2.5 yaml基本使用

（1）yaml模板



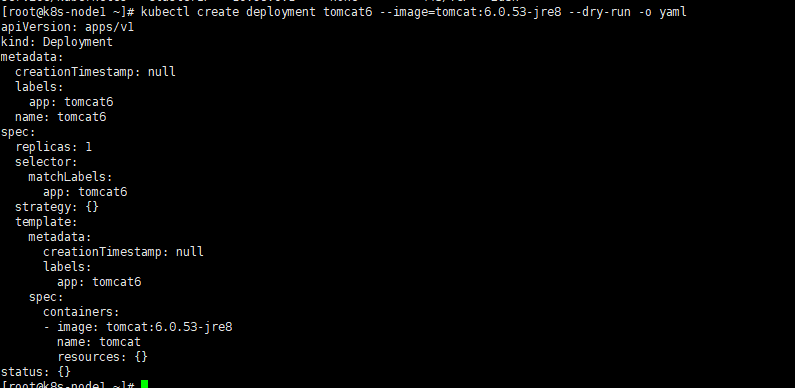
（2）查看创建tomcat6的yaml模板

--dry-run：试运行

-o yaml：输出yaml文件

kubectl create deployment tomcat6 --image=tomcat:6.0.53-jre8 --dry-run -o yaml

|  |
| --- |
| apiVersion: apps/v1  kind: Deployment  metadata:  creationTimestamp: null  labels:  app: tomcat6  name: tomcat6  spec:  replicas: 1  selector:  matchLabels:  app: tomcat6  strategy: {}  template:  metadata:  creationTimestamp: null  labels:  app: tomcat6  spec:  containers:  - image: tomcat:6.0.53-jre8  name: tomcat  resources: {}  status: {} |

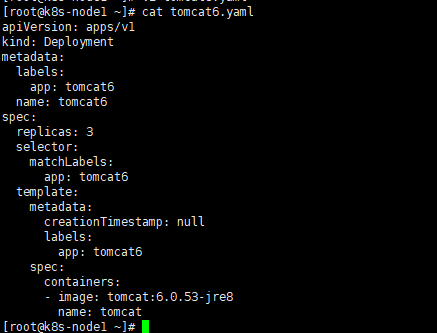


将yaml文件用>输出到tomcat6.yaml

kubectl create deployment tomcat6 --image=tomcat:6.0.53-jre8 --dry-run -o yaml > tomcat6.yaml



修改tomcat6.yaml文件



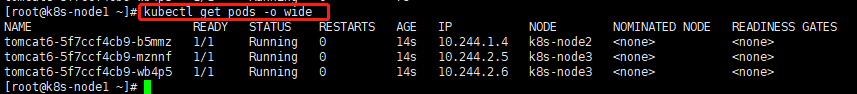
1. 应用yaml文件

kubectl apply -f tomcat6.yaml



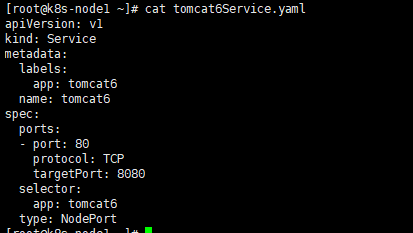
1. 查看pods，有3个pod

kubectl get nodes -o wide

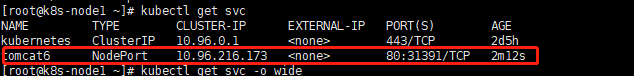


1. 使用yaml暴露tomcat6服务

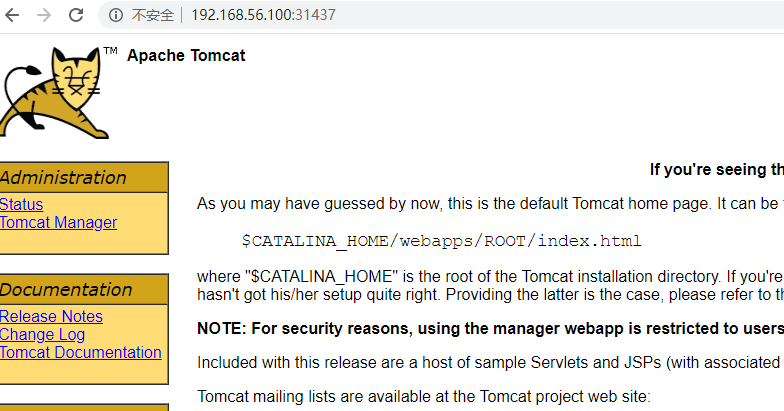
kubectl expose deployment tomcat6 --port=80 --target-port=8080 --type=NodePort --dry-run -o yaml



kubtctl apply -f tomcat6Service.yaml



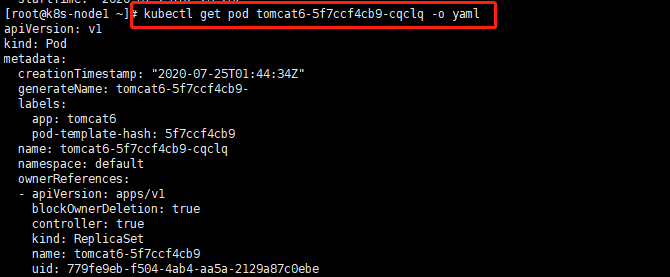
1. 访问测试

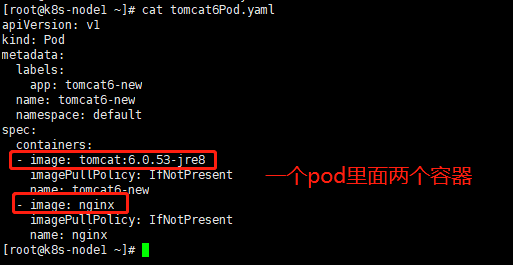


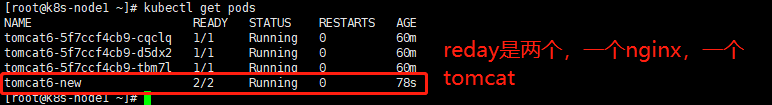
1. 使用yaml暴露pod

kubectl get pod pod名称 -o yaml

kubectl get pod tomcat6-5f7ccf4cb9-cqclq -o yaml



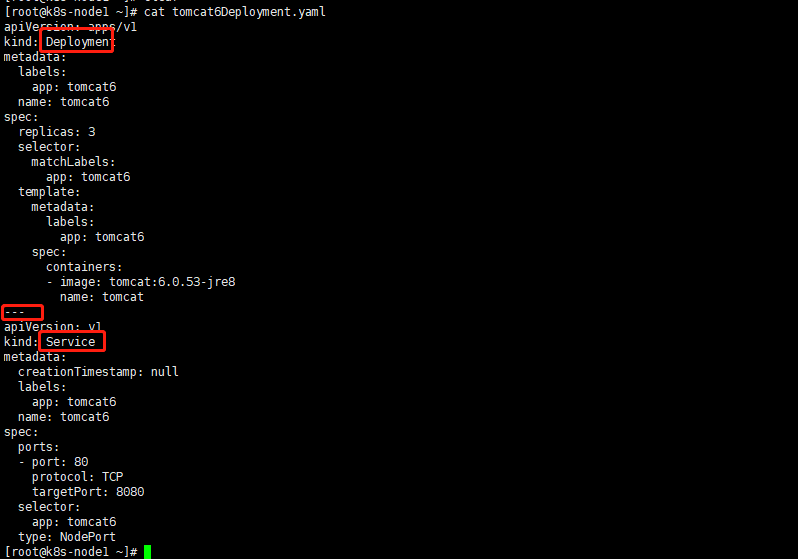


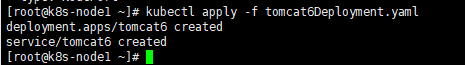


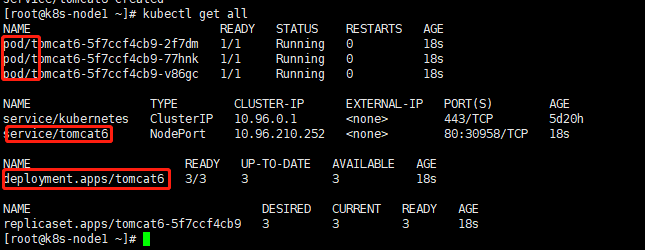
## 2.6 控制器

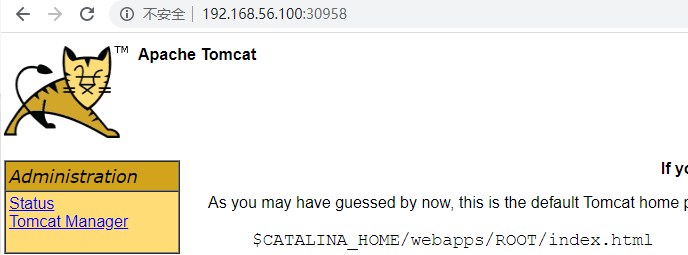
## 2.7 pod、depolyment、service关系

### 2.7.1 创建一个部署并暴露服务









### 2.7.2 访问测试

[http://192.168.56.100:30958/](http://192.168.56.101:30958/)

<http://192.168.56.101:30958/>

[http://192.168.56.102:30958/](http://192.168.56.101:30958/)

## 2.8 ingress

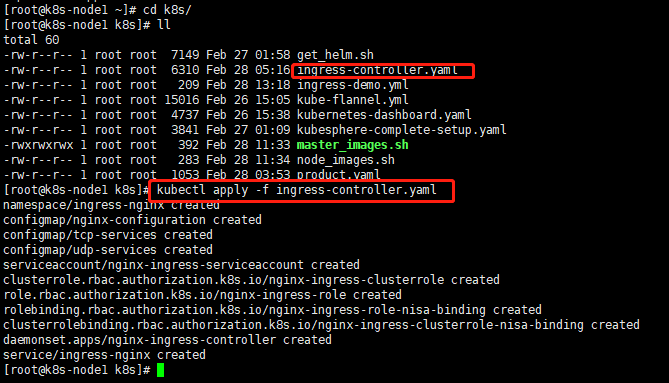
### 2.8.1 ingress简介

刚刚部署的tomcat6暴露了service，我们访问需要通过ip：port来进行访问，使用ingress我们可以使用域名来进行访问一个或多个service，并且会负载均衡到service的每个pod上面。

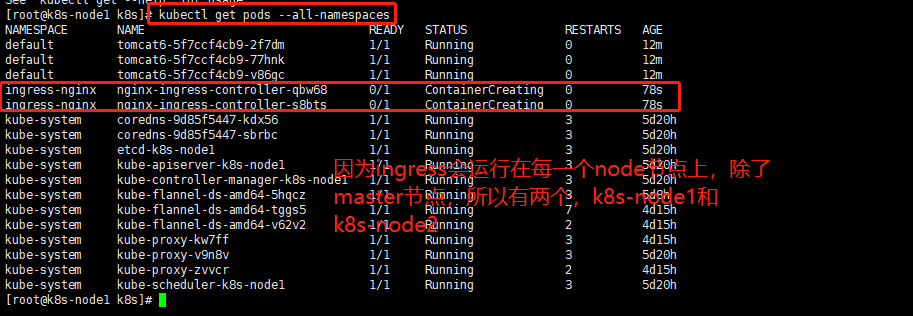
Ingress会运行在每一个node节点上

### 2.8.2 ingress安装

（1）kubectl apply -f ingress-controller.yaml

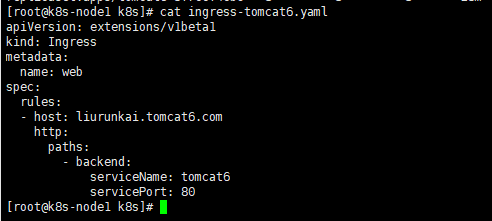


（2）kubectl get pods --all-namespaces 至全变成running状态



1. 创建ingress规则

|  |
| --- |
| apiVersion: extensions/v1beta1  kind: Ingress  metadata:  name: web  spec:  rules:  - host: liurunkai.tomcat6.com  http:  paths:  - backend:  serviceName: tomcat6  servicePort: 80 |



kubectl apply -f ingress-tomcat6.yaml

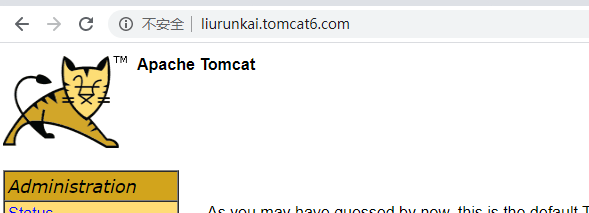


### 2.8.3 测试

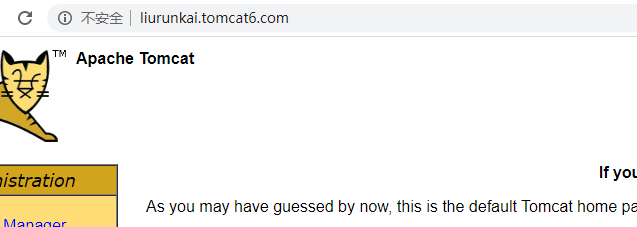
配置hosts文件



使用域名访问：<http://liurunkai.tomcat6.com/>



将192.168.56.101服务停了，重新访问

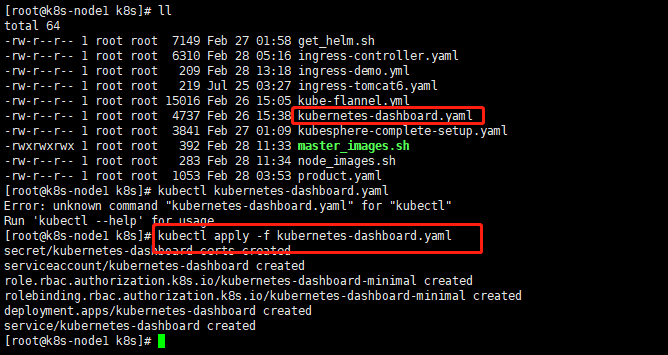


# k8s可视化界面

## 3.1 dashboard

（1）安装

kubectl apply -f kubernetes-dashboard.yaml



## 3.2 Kubesphere

<https://kubesphere.com.cn/docs/zh-CN/installation/install-on-k8s/>

### 3.2.1 前提条件-helm、tiller

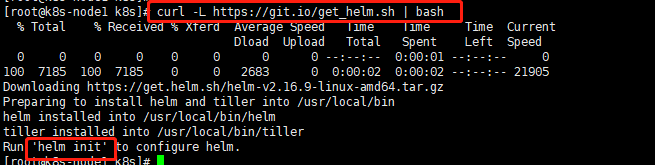
1. master节点安装helm



修改/etc/hosts文件，添加

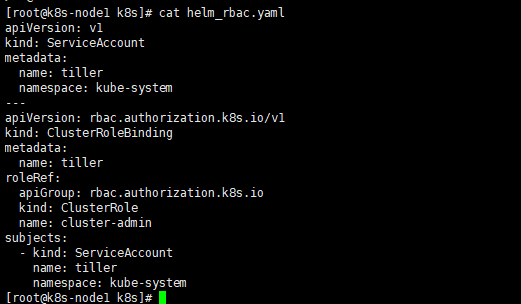
199.232.68.133 raw.githubusercontent.com

执行安装命令：curl -L https://git.io/get\_helm.sh | bash



1. 编写helm权限文件

|  |
| --- |
| apiVersion: v1  kind: ServiceAccount  metadata:  name: tiller  namespace: kube-system  ---  apiVersion: rbac.authorization.k8s.io/v1  kind: ClusterRoleBinding  metadata:  name: tiller  roleRef:  apiGroup: rbac.authorization.k8s.io  kind: ClusterRole  name: cluster-admin  subjects:  - kind: ServiceAccount  name: tiller  namespace: kube-system |

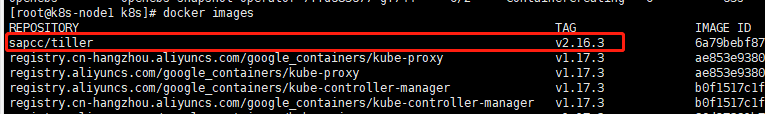


kubectl apply -f helm\_rbac.yaml



1. 初始化

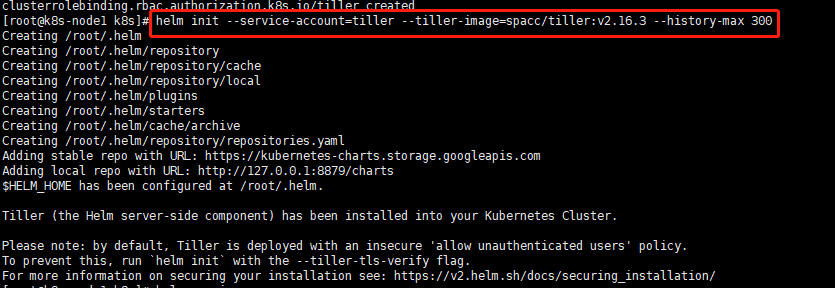
初始化之前可以下下载docker镜像：docker pull sapcc/tiller:v2.16.3



helm init --service-account=tiller --tiller-image=spacc/tiller:v2.16.3 --history-max 300

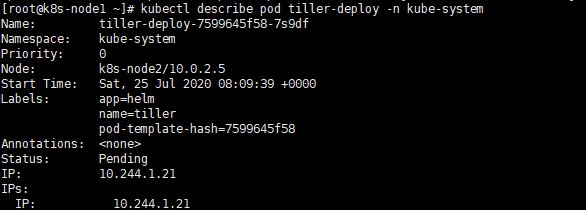
如果报错：不能下载tiller镜像，则使用如下命令

helm init --service-account tiller --tiller-image registry.cn-hangzhou.aliyuncs.com/google\_containers/tiller:v2.12.1



1. 查看pod事件

kubectl describe pod tiller-deploy -n kube-system

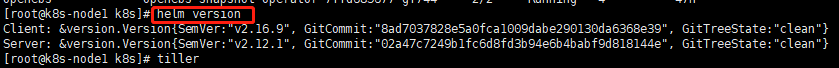


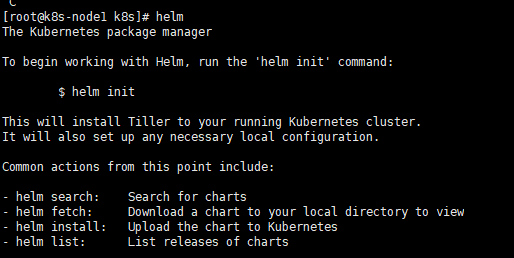
1. 修改deployment

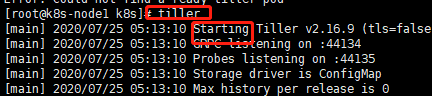
kubectl edit deployment tiller-deploy -n kube-system

1. 查看helm和tiller是否安装成功

helm version

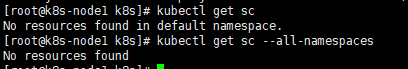






### 3.2.2 前提条件-StorageClass：OpenEBS

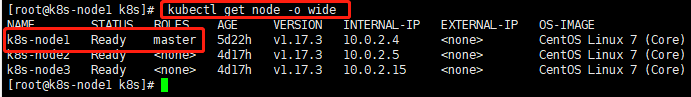
（1）查看集群已有的存储类型(StorageClass)



（2）安装OpenEBS

监控所有节点：kubectl get node -o wide

有一个master节点k8s-node1



查看master节点是否有污点(Taint)

kubectl describe node k8s-node1 | grep Taint



去掉master节点的污点(Taint)

kubectl taint nodes k8s-node1 node-role.kubernetes.io/master:NoSchedule-



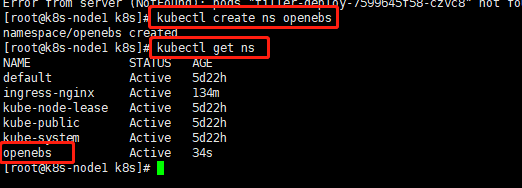
再次查看master节点污点(Taint)

kubectl describe node k8s-node1 | grep Taint



创建OpenEBS的namespace：kubectl create ns openebs

查看已有的namespace：kubectl get ns

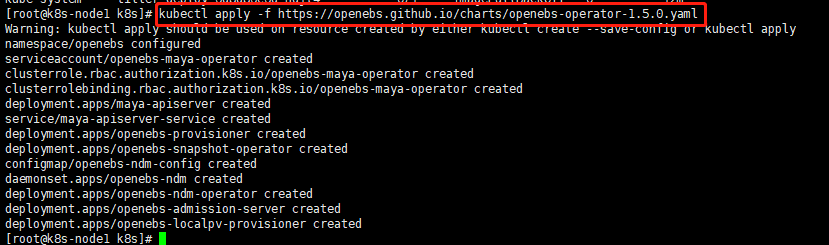


如果已安装helm，则使用helm来安装OpenEBS:

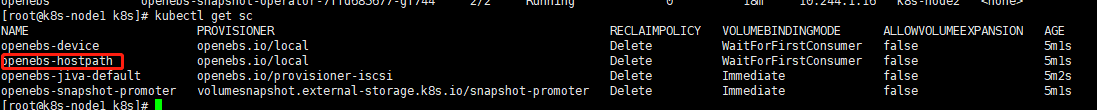
helm install --namespace openebs --name openebs stable/openebs --version 1.5.0

如果没有安装helm，则通过kubectl命令来安装

kubectl apply -f https://openebs.github.io/charts/openebs-operator-1.5.0.yaml



查看创建的sc(StorageClass)：kubectl get sc

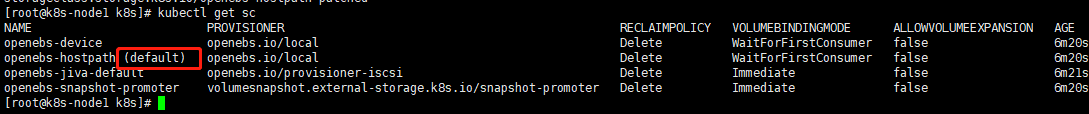


设置OpenEBS默认的StorageClass为openebs-hostpath：

kubectl patch storageclass openebs-hostpath -p '{"metadata": {"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'



查看StorageClass



给master节点添加污点(taint)：不参与调度(NoSchedule)：

kubectl taint nodes k8s-node1 node-role.kubernetes.io=master:NoSchedule

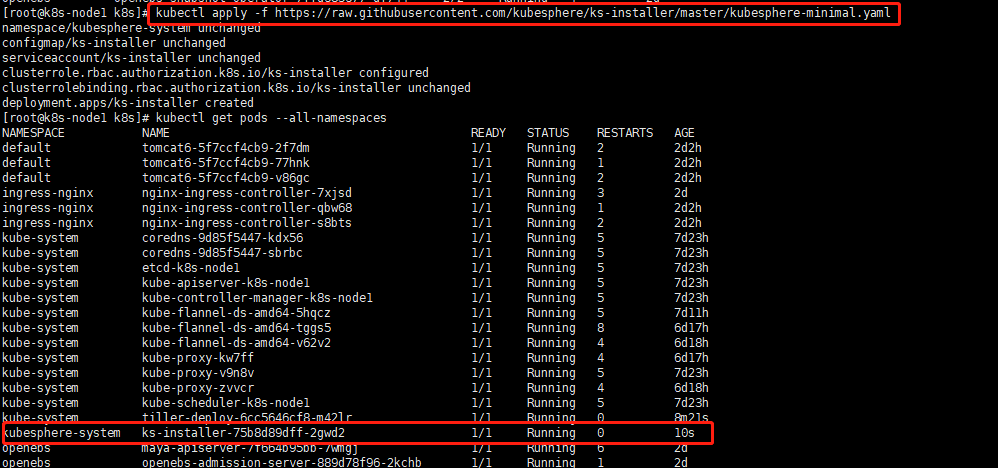


查看master节点污点添加成功：kubectl describe node k8s-node1 | grep Taint



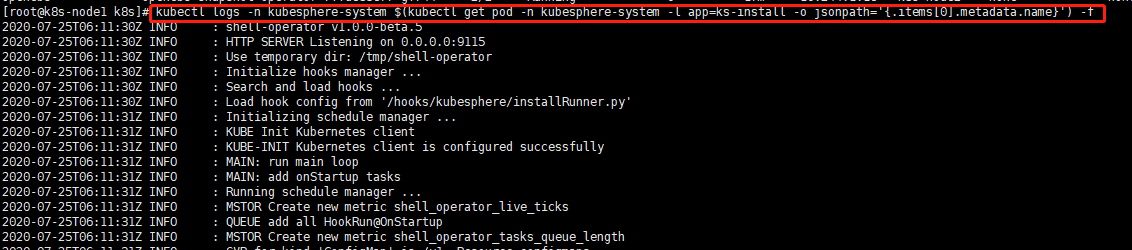
### 3.2.3 最小化安装kubesphere

kubectl apply -f <https://raw.githubusercontent.com/kubesphere/ks-installer/master/kubesphere-minimal.yaml>

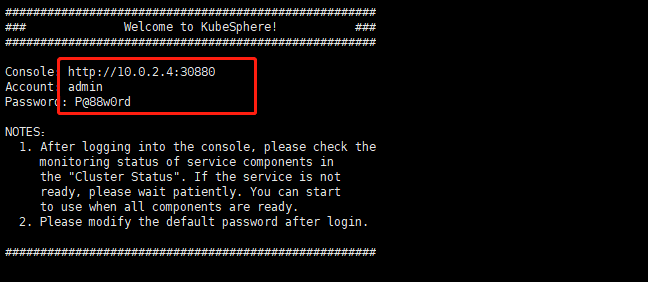


查看安装日志

kubectl logs -n kubesphere-system $(kubectl get pod -n kubesphere-system -l app=ks-install -o jsonpath='{.items[0].metadata.name}') -f

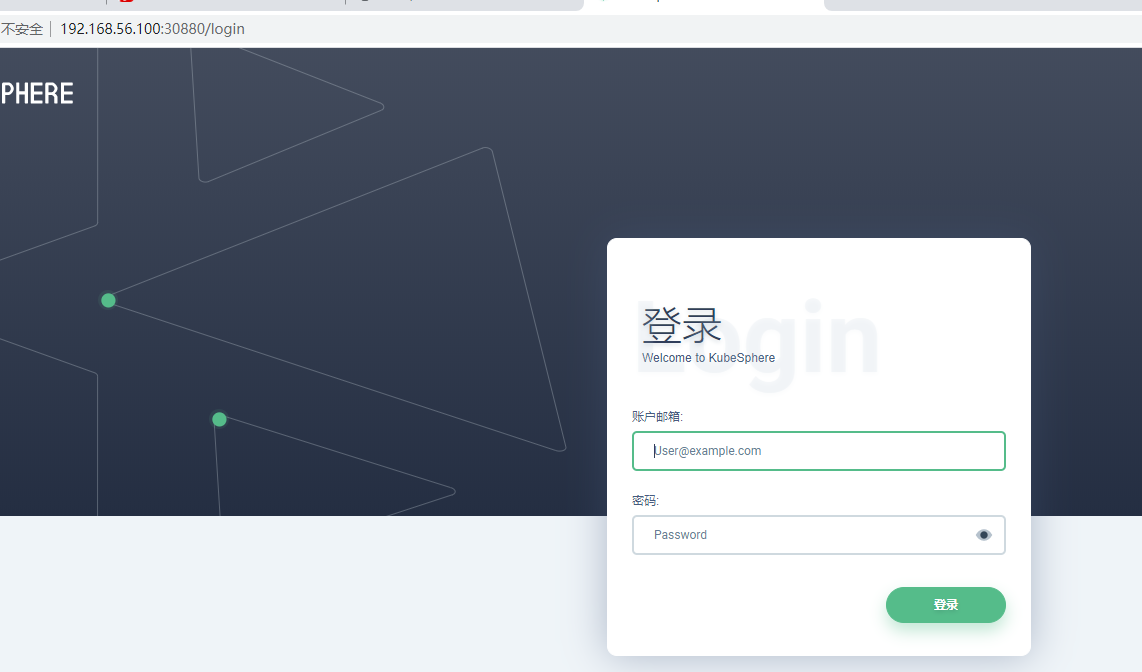


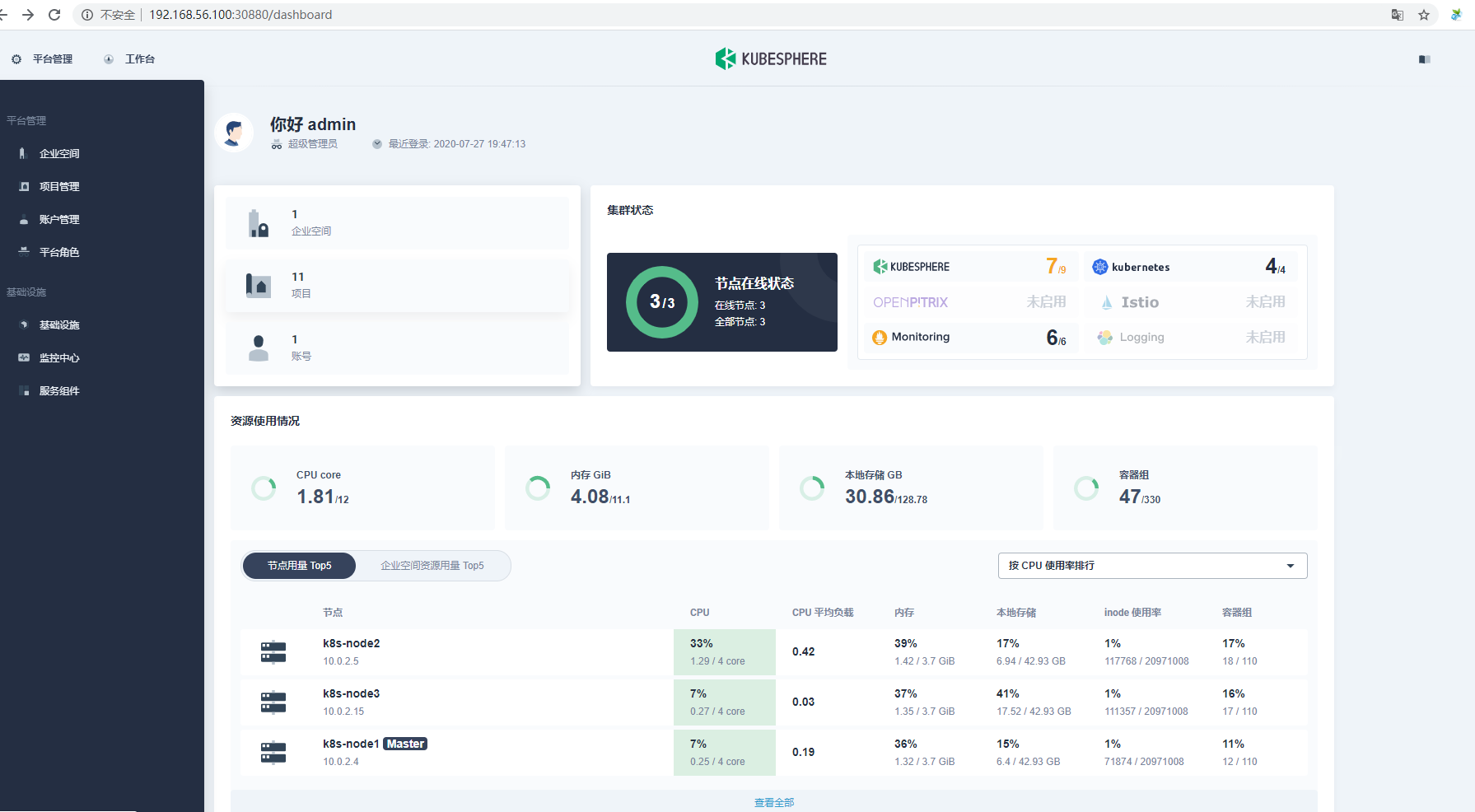
安装成功后，日志会展示访问地址、用户名、密码，访问需要用外网地址访问



### 3.2.4 访问测试

<http://192.168.56.100:30880/>，登录后修改的密码为：Lrk123456



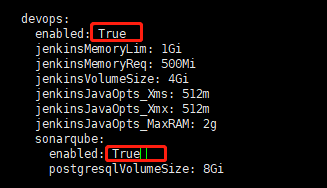


### 3.2.5 定制化安装(可插拔组件)



1. 使用命令：kubectl edit cm -n kubesphere-system ks-installer将所要开启的功能改为True

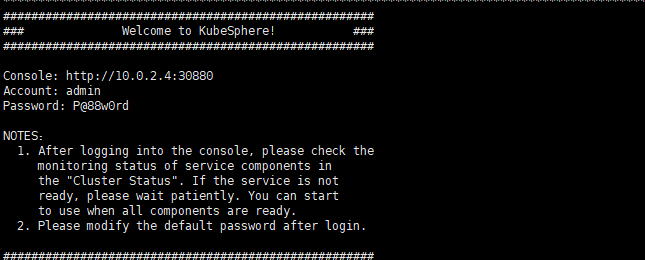
开启devops、sonarqube(代码质量检测)、notification(通知)、alerting(告警)：将false改为true





1. 查看日志

kubectl logs -n kubesphere-system $(kubectl get pod -n kubesphere-system -l app=ks-install -o jsonpath='{.items[0].metadata.name}') -f

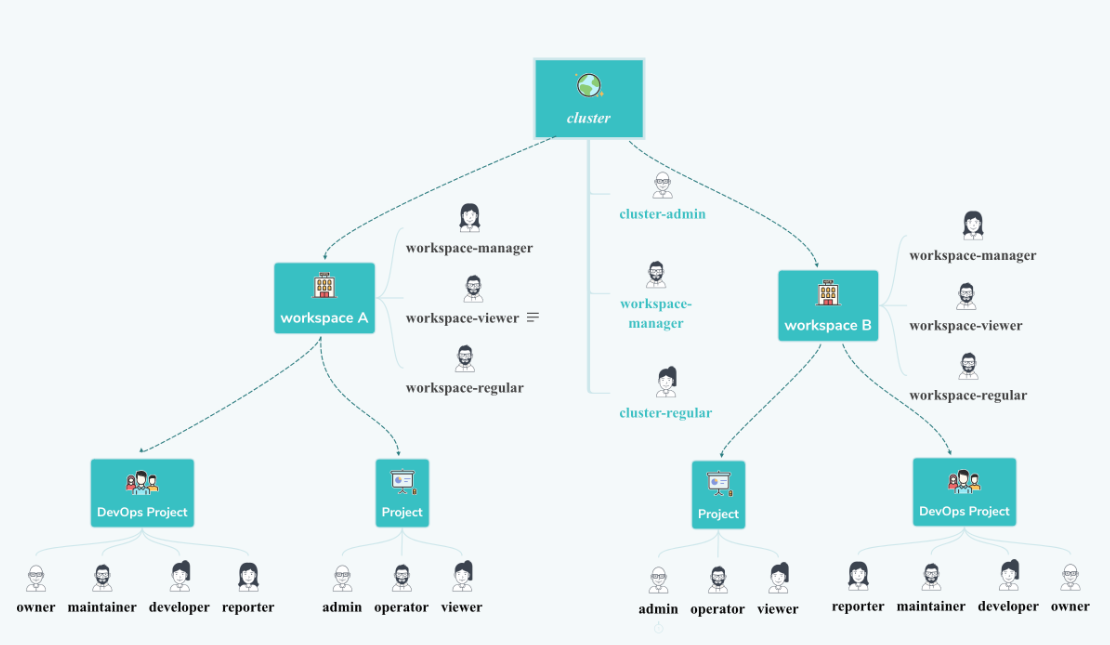


### 3.2.6 kubeshere快速入门

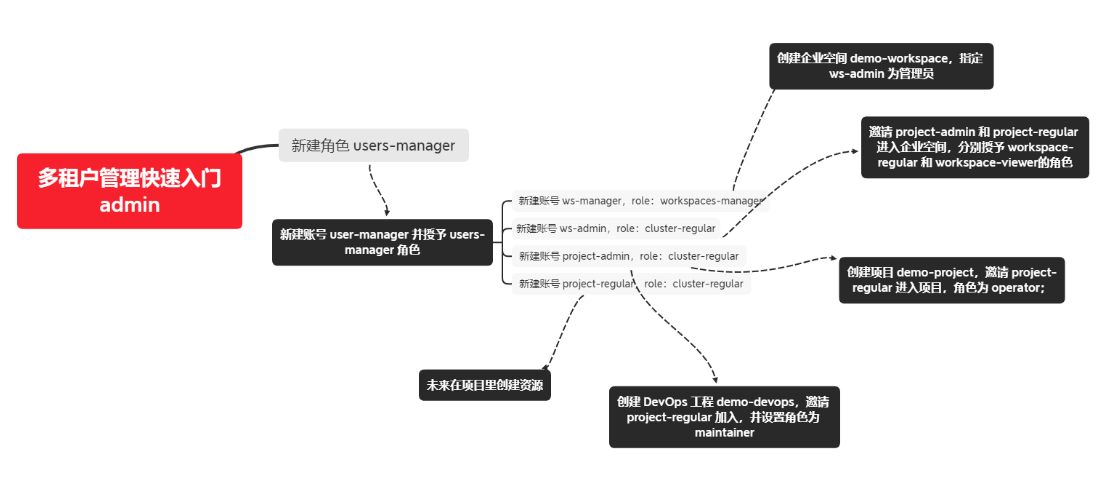
<https://kubesphere.io/docs/zh-CN/quick-start/admin-quick-start/>

1. 多租户管理

平台的资源一共有三个层级，包括 ****集群 (Cluster)、 企业空间 (Workspace)、 项目 (Project) 和 DevOps Project (DevOps 工程)****，层级关系如下图所示，即一个集群中可以创建多个企业空间，而每个企业空间，可以创建多个项目和 DevOps工程，而集群、企业空间、项目和 DevOps工程中，默认有多个不同的内置角色



| **内置角色** | **描述** |
| --- | --- |
| cluster-admin | 集群管理员，可以管理集群中所有的资源。 |
| workspaces-manager | 集群中企业空间管理员，仅可创建、删除企业空间，  维护企业空间中的成员列表。 |
| cluster-regular | 集群中的普通用户，在被邀请加入企业空间之前没有任何资源操作权限。 |



1. 应用路由与服务示例
2. 创建wordPress应用并发布至k8s
3. 密码等信息属于敏感信息，不适合以明文的方式表现在步骤中，因此以创建密钥的方式来代替该环境变量。创建的密钥将在创建容器组设置时作为环境变量写入



1. 创建存储卷



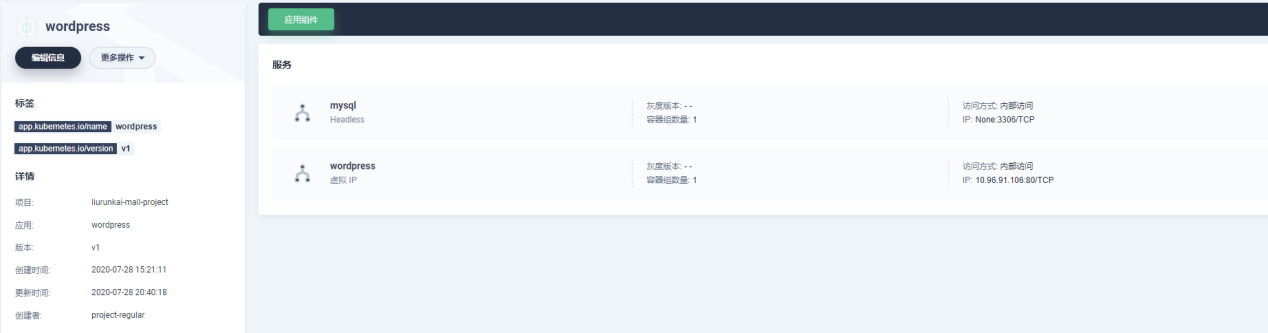
1. 创建应用

-- 添加mysql组件

-- 添加wordpress组件



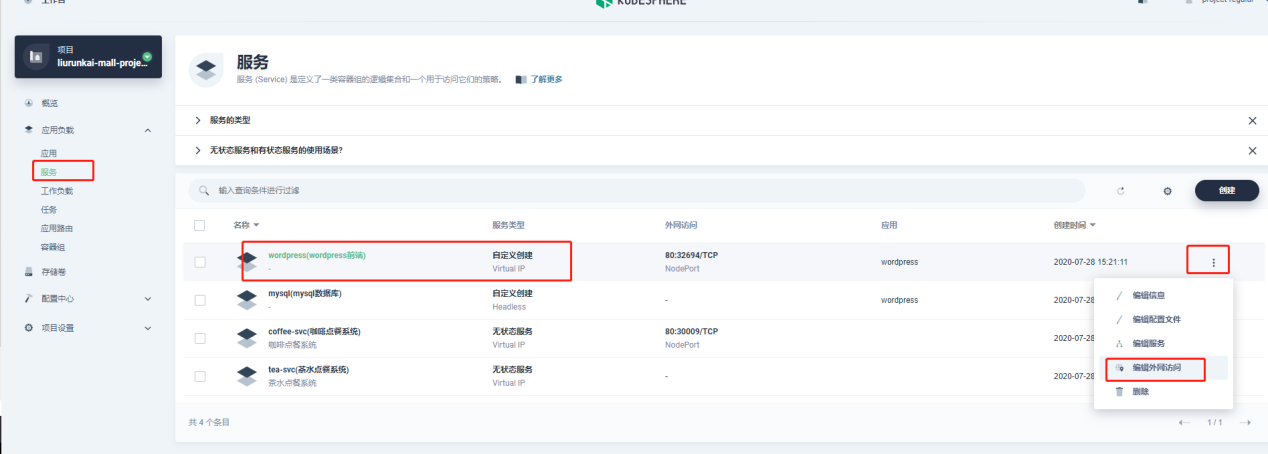
点击wordpress进去可以看到mysql和wordpress组件



1. 查看应用资源(有状态mysql，无状态wordpress)



1. 编辑外网访问

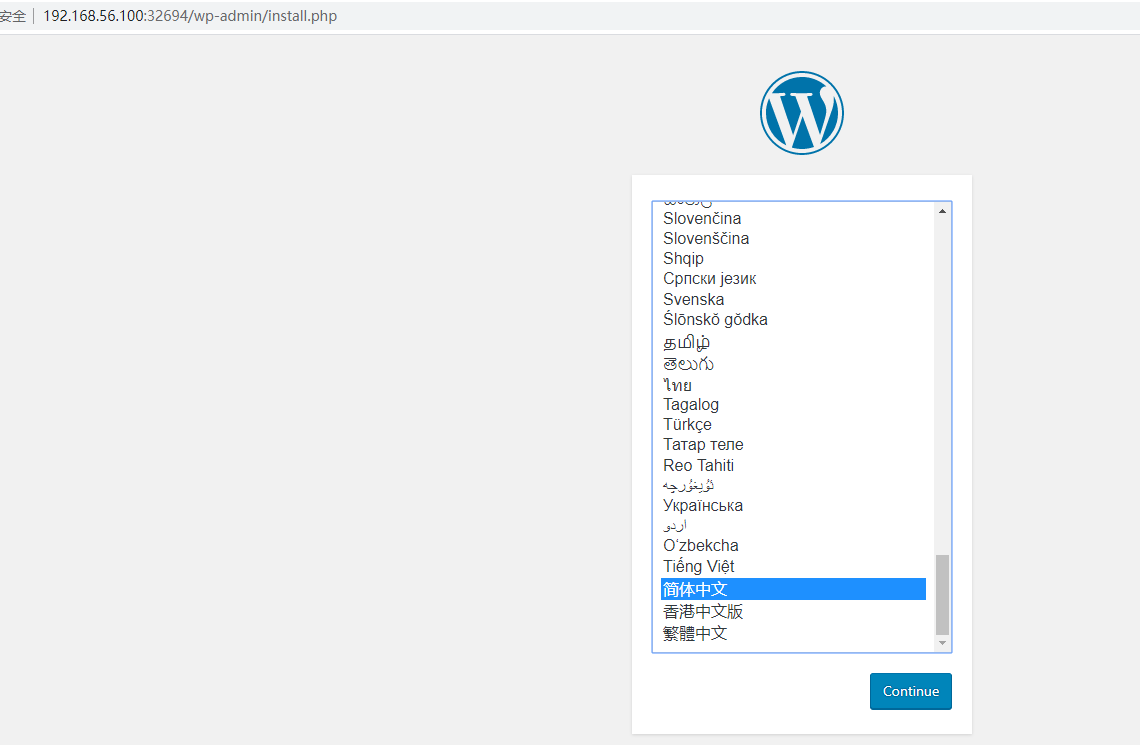


1. 访问测试

<http://192.168.56.100:32694/>

[http://192.168.56.101:32694/](http://192.168.56.100:32694/)

[http://192.168.56.102:32694/](http://192.168.56.100:32694/)



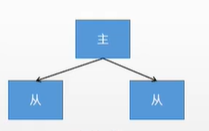
# k8s部署集群

## 4.1 集群形式

（1）主从式

主从复制，同步方式

主从调度，控制方式，eg：k8s集群，由主节点来控制从节点，将请求转发至对应的从节点



1. 分片式

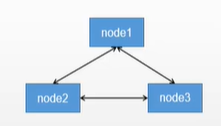
数据分片存储，片区之间备份



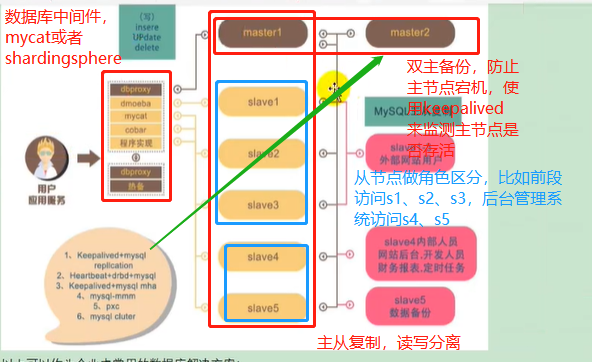
1. 选主式

为了容灾选主

为了调度选主



## 4.2 mysql集群



### 4.2.1 创建master实例并启动

|  |
| --- |
| docker run -p 3307:3306 --name mysql-master \  -v /mydata/mysql/master/log:/var/log/mysql \  -v /mydata/mysql/master/data:/var/lib/mysql \  -v /mydata/mysql/master/conf:/etc/mysql \  -e MYSQL\_ROOT\_PASSWORD=root \  -d mysql:5.7 |

### 4.2.2 创建slave实例并启动

|  |
| --- |
| docker run -p 3317:3306 --name mysql-slave-01 \  -v /mydata/mysql/slave/log:/var/log/mysql \  -v /mydata/mysql/slave/data:/var/lib/mysql \  -v /mydata/mysql/slave/conf:/etc/mysql \  -e MYSQL\_ROOT\_PASSWORD=root \  -d mysql:5.7 |

### 4.2.3 修改master的配置文件

vim /mydata/mysql/master/conf/my.cnf

|  |
| --- |
| [client]  default-character-set=utf8  [mysql]  default-character-set=utf8  [mysqld]  init\_connect=’SET collation\_connection = utf8\_unicode\_ci’  init\_connect=’SET NAMES utf8’  character-set-server=utf8  collation-server=utf8\_unicode\_ci  skip-character-set-client-handshake  skip-name-resolve  sql\_mode=NO\_ENGINE\_SUBSTITUTION,STRICT\_TRANS\_TABLES  # 主从复制部分配置  #设置server-id  server-id=1  #开启二进制日志  log-bin=mysql-bin  #不是只读  read-only=0  #哪些数据库要生成binlog日志  binlog-do-db=lrkmall\_cms  binlog-do-db=lrkmall\_mms  binlog-do-db=lrkmall\_oms  binlog-do-db=lrkmall\_pms  binlog-do-db=lrkmall\_wms  binlog-do-db=lrkmall\_admin  # 复制的时候要忽略的数据库  replicate-ignore-db=mysql  replicate-ignore-db=sys  replicate-ignore-db=information\_schema  replicate-ignore-db=performance\_schema |

### 4.2.4修改slave的配置文件

vim /mydata/mysql/slave/conf/my.cnf

|  |
| --- |
| [client]  default-character-set=utf8  [mysql]  default-character-set=utf8  [mysqld]  init\_connect=’SET collation\_connection = utf8\_unicode\_ci’  init\_connect=’SET NAMES utf8’  character-set-server=utf8  collation-server=utf8\_unicode\_ci  skip-character-set-client-handshake  skip-name-resolve  sql\_mode=NO\_ENGINE\_SUBSTITUTION,STRICT\_TRANS\_TABLES  # 主从复制部分配置  #设置server-id  server-id=2  #开启二进制日志  log-bin=mysql-bin  #不是只读  read-only=1  #哪些数据库要生成binlog日志  binlog-do-db=lrkmall\_cms  binlog-do-db=lrkmall\_mms  binlog-do-db=lrkmall\_oms  binlog-do-db=lrkmall\_pms  binlog-do-db=lrkmall\_wms  binlog-do-db=lrkmall\_admin  # 复制的时候要忽略的数据库  replicate-ignore-db=mysql  replicate-ignore-db=sys  replicate-ignore-db=information\_schema  replicate-ignore-db=performance\_schema |

### 4.2.5 重启master和slave-01

### 4.2.6 为master授权用户来同步数据

使用mysql客户端navicat连接，可以直接从添加用来同步数据的用户开始执行，前面的不用执行

需要关闭主库的防火墙：systemctl stop firewalld，否则会报错如下

Slave I/O for channel '': error connecting to master 'backup@192.168.37.129:3307' - retry-time: 60 retries: 183, Error\_code: 2003



（1）进入master容器：docker exec -it 容器id /bin/bash

1. 登录mysql：mysql -uroot -proot

授权root可以远程访问

GRANT ALL privileges ON \*.\* TO 'root'@'%' identified by 'root' with grant option;

刷新权限

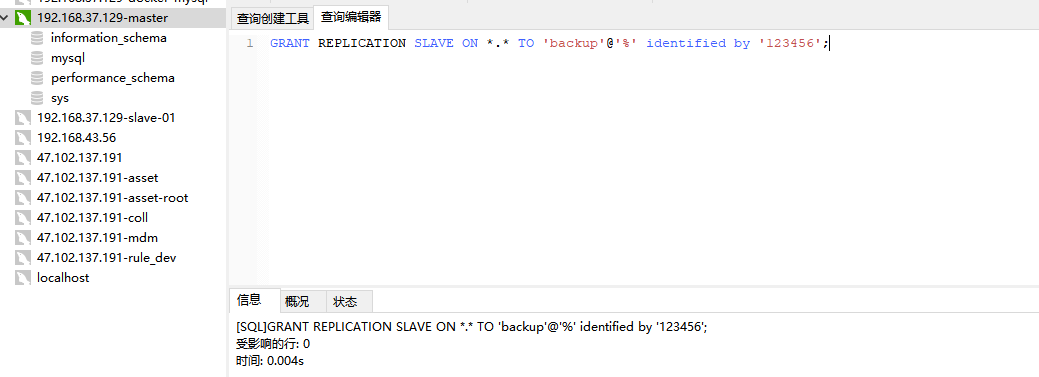
flush privileges;

添加用来同步数据的用户

GRANT REPLICATION SLAVE ON \*.\* TO 'backup'@'%' identified by '123456';

刷新权限

flush privileges;



1. 查看master状态：show master status \G



### 4.2.7 设置slave同步master数据

（1）登录slave数据库

（2）设置主库连接，设置slave需要同步master的信息

change master to master\_host=’192.168.37.129’,master\_user=’backup’,master\_password=’123456’,master\_log\_file=’mysql-bin.000002’,master\_log\_pos=0,master\_port=3307



1. 启动从库同步

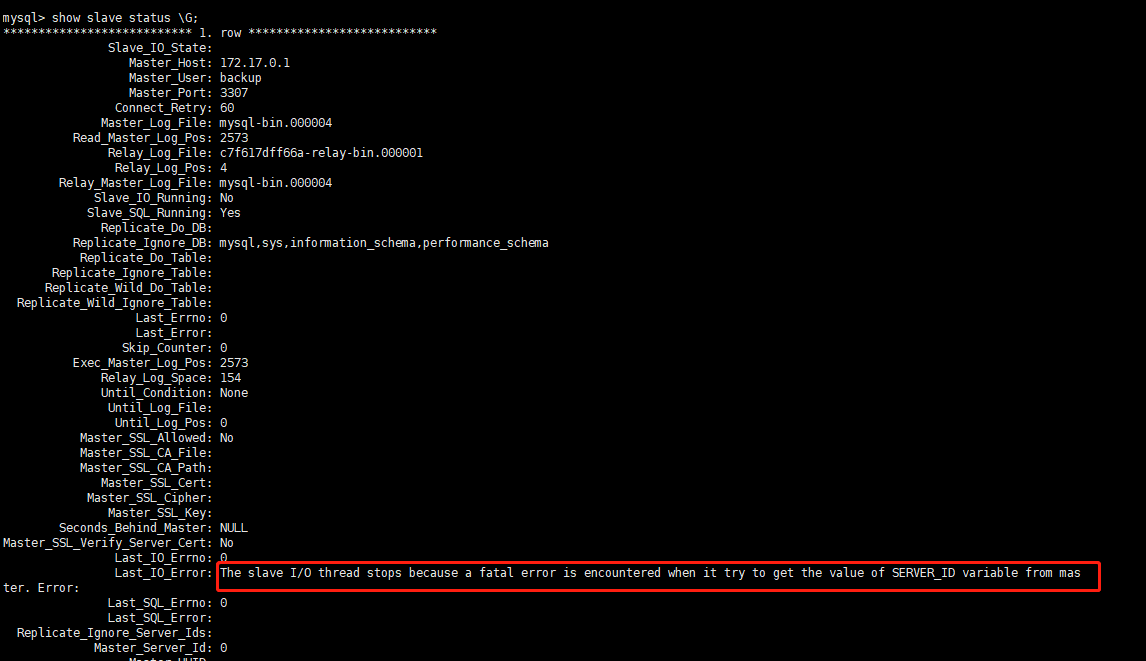
start slave

1. 查看从库状态

show slave status \G;

如果报错如下：

[ERROR] Slave I/O for channel '': The slave I/O thread stops because a fatal error is encountered when it try to get the value of SERVER\_ID variable from master. Error: , Error\_code: 0



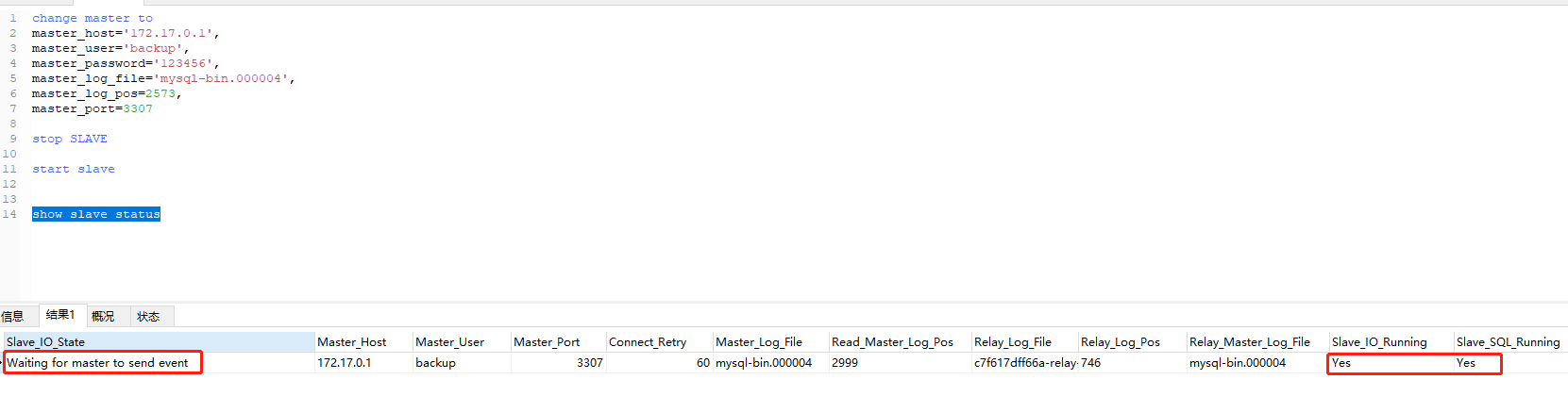
可能是备份的用户权限不够(backup权限不够)，需要重新授权

GRANT ALL PRIVILEGES ON \*.\* TO 'backup'@'%' identified by '123456';

flush privileges;

然后重新执行如下命令：

|  |
| --- |
| stop SLAVE  change master to  master\_host='172.17.0.1',  master\_user='backup',  master\_password='123456',  master\_log\_file='mysql-bin.000004',  master\_log\_pos=2573,  master\_port=3307  start slave  show slave status |



### 4.2.8 测试

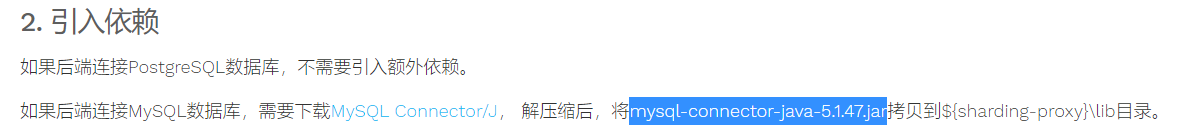
将master创建数据库、表、导入数据，slave可以自动同步同步

实现了mysql主从同步

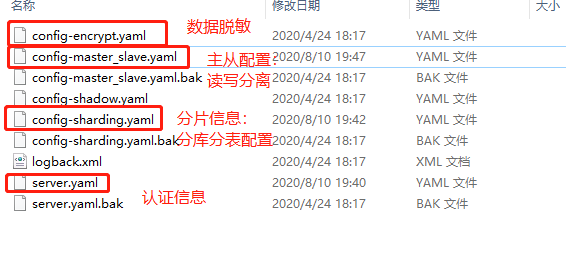
## 4.3 shardingsphere-shardingProxy

### 4.3.1 下载sharding-proxy

### 4.3.2 下载mysql驱动

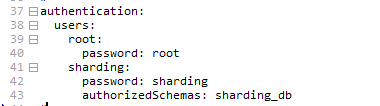


### 4.3.3 配置sharding-proxy

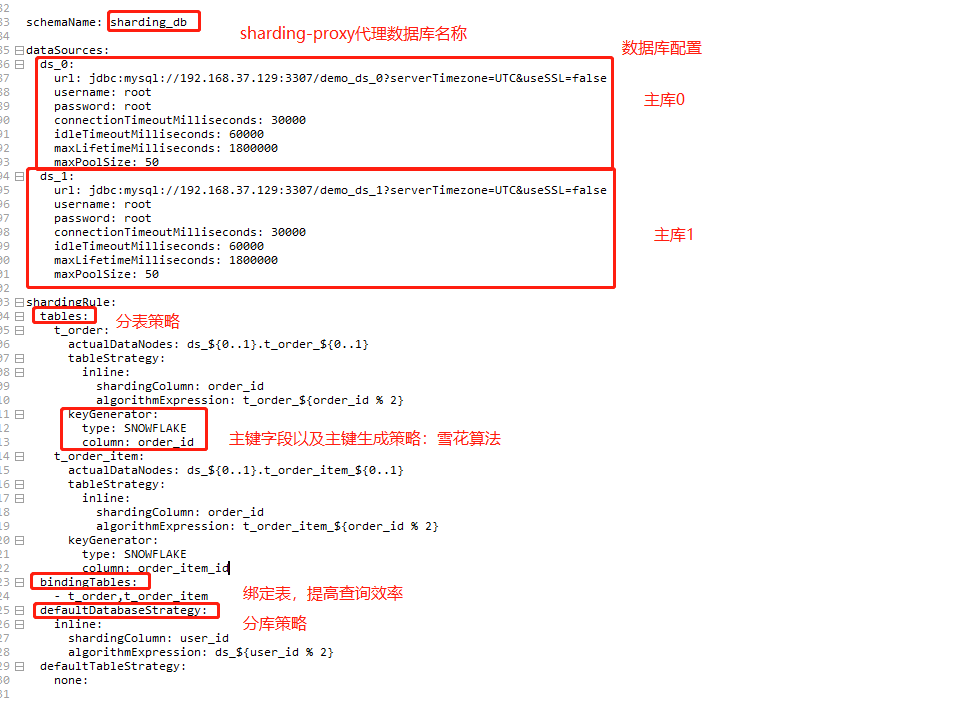


1. 配置认证信息：server.yaml

将下面配置信息打开，默认是注释掉的

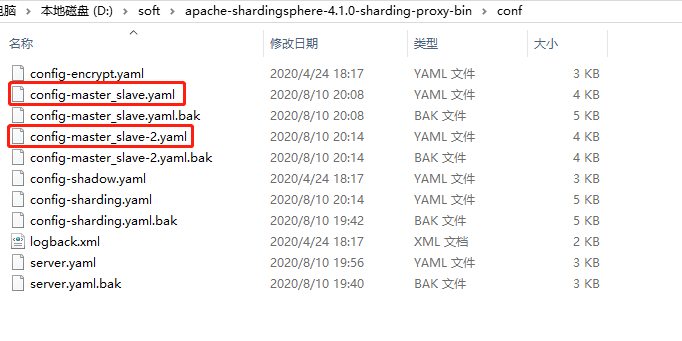


1. 配置分片信息(分库分表配置)：config-sharding.yaml

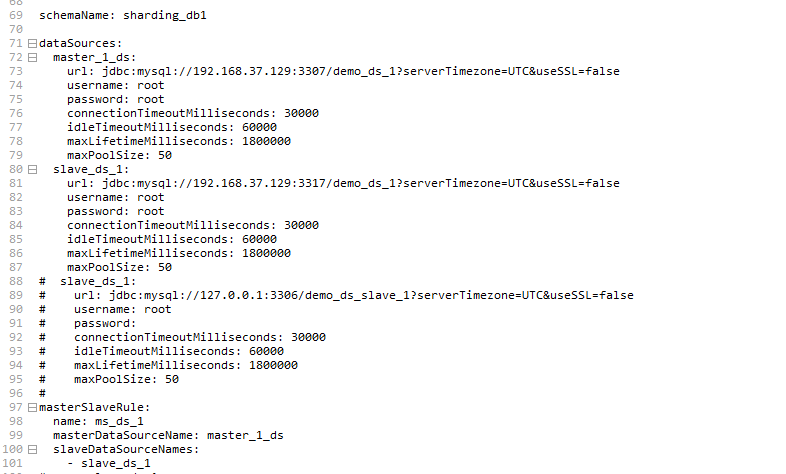


1. 配置主从(读写分离配置)：config-master\_slave.yaml

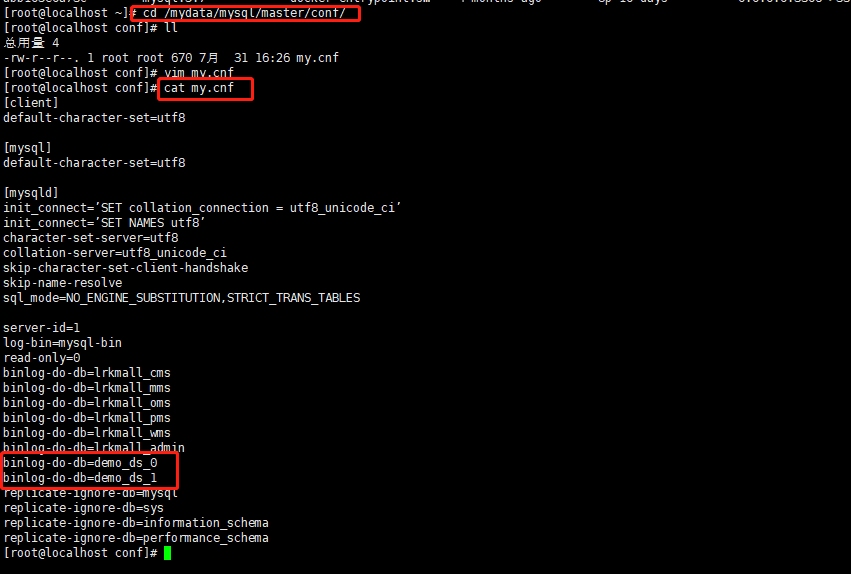
有多少个主从就复制几份config-master\_slave.yaml配置文件，每个配置文件里面配置一份主从

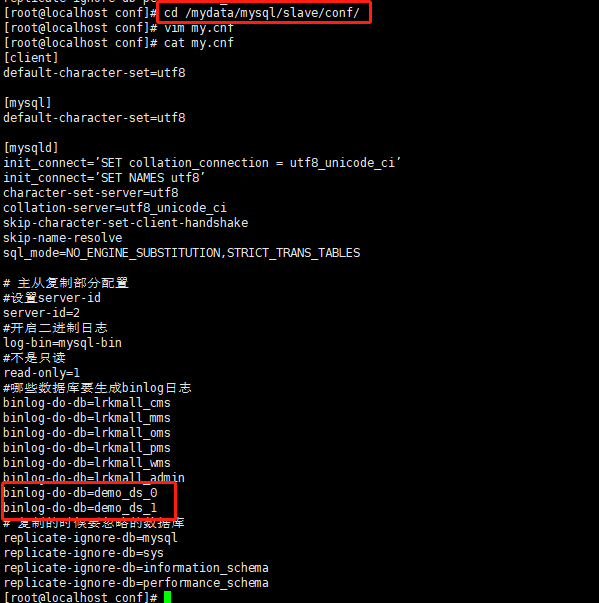




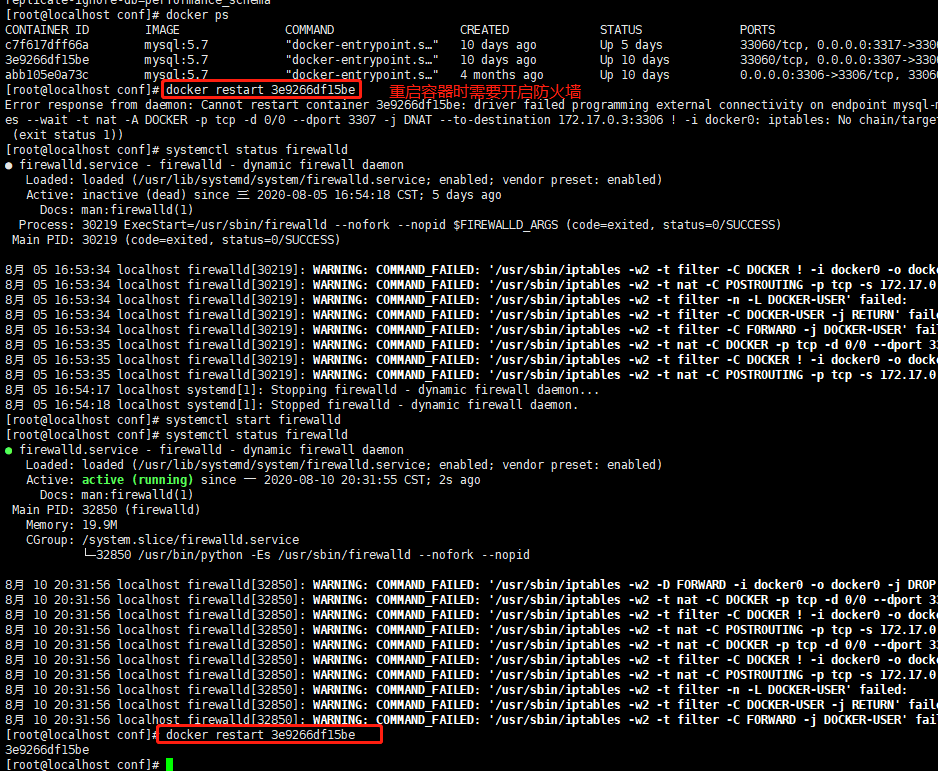


### 4.3.4 修改数据库配置文件，添加要同步的数据库并重启





重启数据库

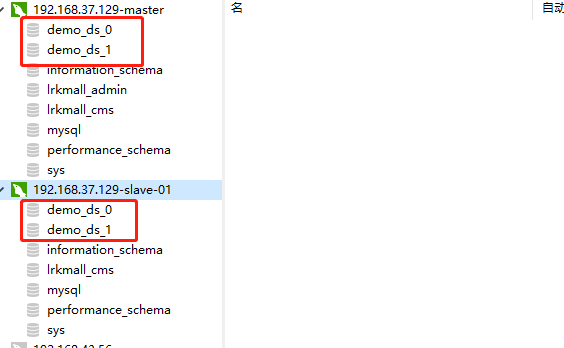


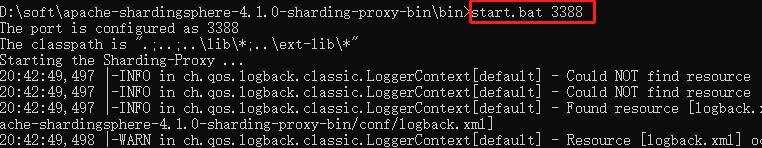
重启完数据库后需要关闭防火墙，否则从库将无法连接主库，或者将主库端口3307和从库端口3317的端口开放

### 4.3.5 自定义端口3388启动sharding-proxy



在主库和从库分别建立对应的数据库

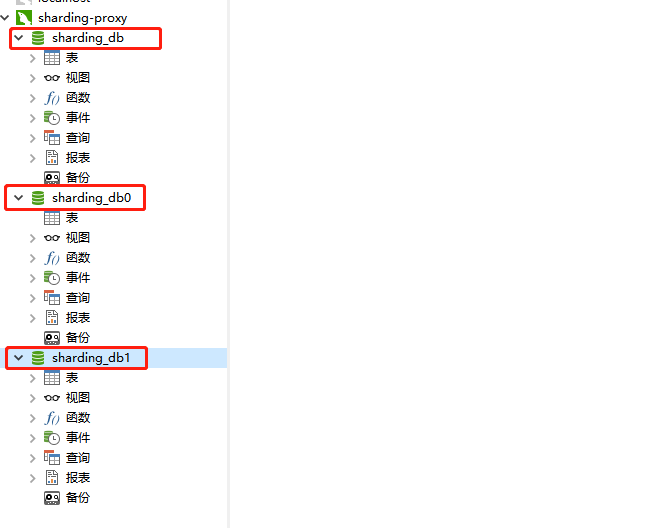




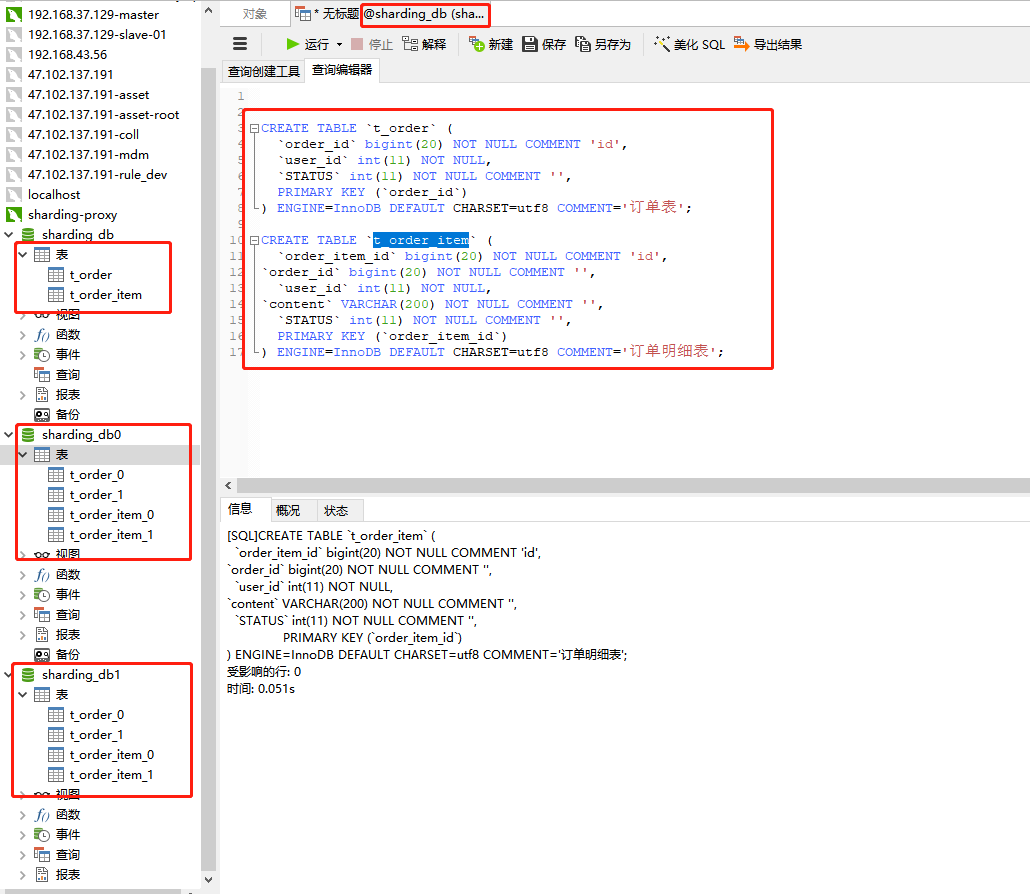
### 4.3.6 测试



存在3个数据库，sharding\_db、sharding\_db0、sharding\_db1



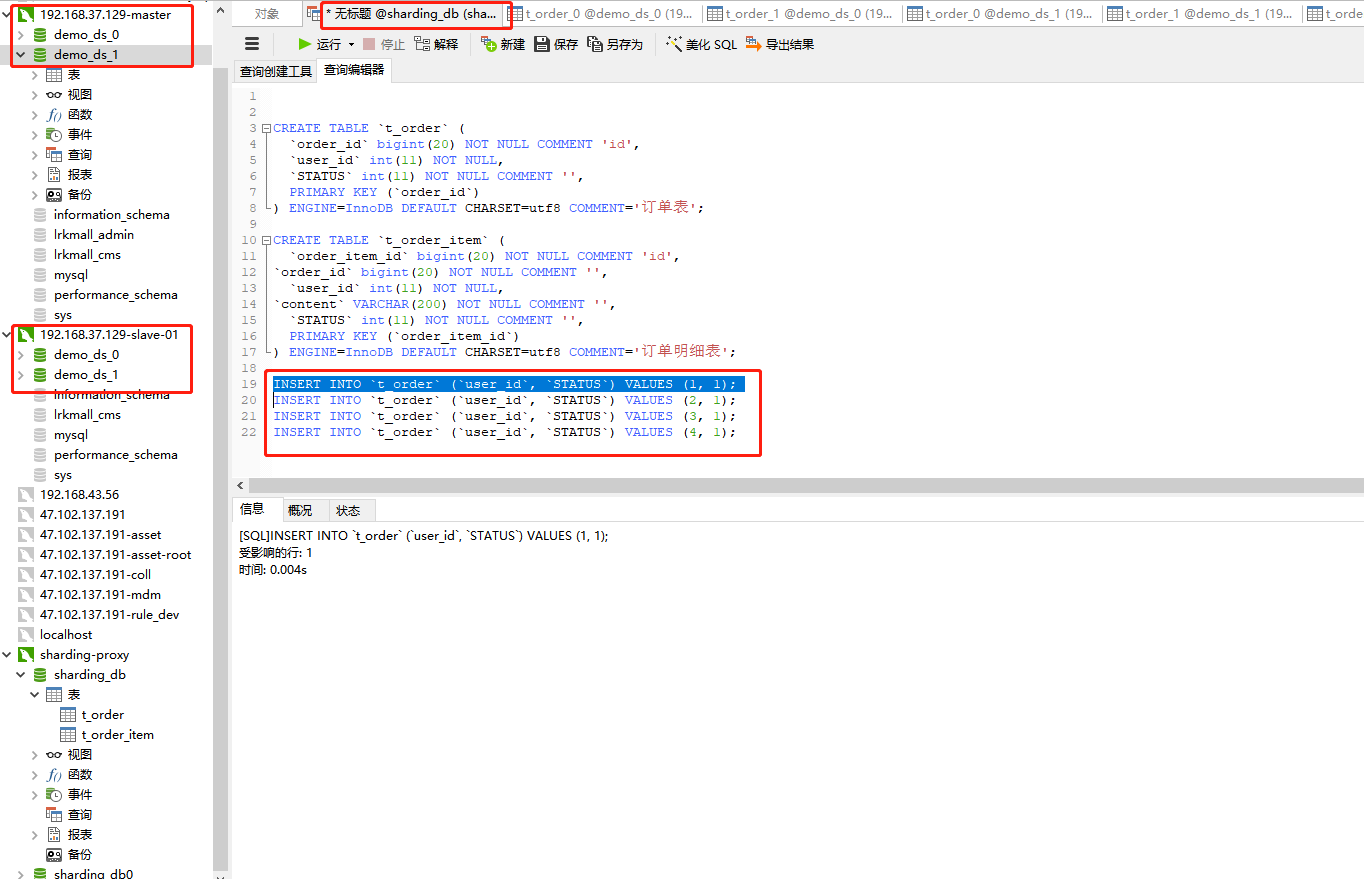
在sharding-db创建表t\_order、t\_order\_item(在sharding\_db0和sharding\_db1中会看到t\_order\_0、t\_order\_1、t\_order\_item\_0、t\_order\_item\_1)



在sharding-db插入数据，可以在主库和从库的t\_order\_0和t\_order\_1表中分别看到数据

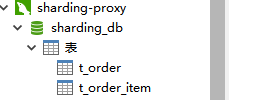
user\_id%2 == 0 数据落到demo\_ds\_0库，否则落到demo\_ds\_1库

Order\_id%2 == 0 数据落到t\_order\_0表，否则落到t\_order\_1表



### 4.3.7 总结

如果使用sharding-proxy需要下载并且在conf配置文件夹下配置分库分表及读写分离策略，程序链接的时候直接连接sharding-proxy即可



如果使用sharding-jdbc不需要下载，但是需要在springboot程序的yml配置文件里面配置分库分表及读写分离策略

|  |
| --- |
| ## mysql 分库分表  #spring.shardingsphere.datasource.names=db0,db1  #  #spring.shardingsphere.datasource.db0.type=com.alibaba.druid.pool.DruidDataSource  #spring.shardingsphere.datasource.db0.driver-class-name=com.mysql.jdbc.Driver  #spring.shardingsphere.datasource.db0.url=jdbc:mysql://192.168.37.128:3306/db0?useUnicode=true&characterEncoding=utf-8&useSSL=false&serverTimezone=GMT  #spring.shardingsphere.datasource.db0.username=root  #spring.shardingsphere.datasource.db0.password=root  #  #spring.shardingsphere.datasource.db1.type=com.alibaba.druid.pool.DruidDataSource  #spring.shardingsphere.datasource.db1.driver-class-name=com.mysql.jdbc.Driver  #spring.shardingsphere.datasource.db1.url=jdbc:mysql://192.168.37.128:3306/db1?useUnicode=true&characterEncoding=utf-8&useSSL=false&serverTimezone=GMT  #spring.shardingsphere.datasource.db1.username=root  #spring.shardingsphere.datasource.db1.password=root  #  ## 分库分表策略  #spring.shardingsphere.sharding.default-database-strategy.inline.sharding-column=id  #spring.shardingsphere.sharding.default-database-strategy.inline.algorithm-expression=db$->{id % 2}  #  #spring.shardingsphere.sharding.tables.user.actual-data-nodes=db$->{0..1}.user\_$->{0..1}  #spring.shardingsphere.sharding.tables.user.table-strategy.inline.sharding-column=age  #spring.shardingsphere.sharding.tables.user.table-strategy.inline.algorithm-expression=user\_$->{age % 2}  ## 主键生成策略：雪花算法  #spring.shardingsphere.sharding.tables.user.key-generator.column=id  #spring.shardingsphere.sharding.tables.user.key-generator.type=SNOWFLAKE  #  ## 打印执行的数据库以及语句  #spring.shardingsphere.props.sql.show=true  #spring.main.allow-bean-definition-overriding=true  # mysql读写分离  #spring.shardingsphere.datasource.names=db0,db1,sdb0,sdb1  #  #spring.shardingsphere.datasource.db0.type=com.alibaba.druid.pool.DruidDataSource  #spring.shardingsphere.datasource.db0.driver-class-name=com.mysql.cj.jdbc.Driver  #spring.shardingsphere.datasource.db0.url=jdbc:mysql://192.168.37.128:3306/db0?useUnicode=true&characterEncoding=utf-8&useSSL=false&serverTimezone=GMT  #spring.shardingsphere.datasource.db0.username=root  #spring.shardingsphere.datasource.db0.password=root  #  #spring.shardingsphere.datasource.db1.type=com.alibaba.druid.pool.DruidDataSource  #spring.shardingsphere.datasource.db1.driver-class-name=com.mysql.cj.jdbc.Driver  #spring.shardingsphere.datasource.db1.url=jdbc:mysql://192.168.37.128:3306/db1?useUnicode=true&characterEncoding=utf-8&useSSL=false&serverTimezone=GMT  #spring.shardingsphere.datasource.db1.username=root  #spring.shardingsphere.datasource.db1.password=root  #  #spring.shardingsphere.datasource.sdb0.type=com.alibaba.druid.pool.DruidDataSource  #spring.shardingsphere.datasource.sdb0.driver-class-name=com.mysql.cj.jdbc.Driver  #spring.shardingsphere.datasource.sdb0.url=jdbc:mysql://192.168.37.128:3306/sdb0?useUnicode=true&characterEncoding=utf-8&useSSL=false&serverTimezone=GMT  #spring.shardingsphere.datasource.sdb0.username=root  #spring.shardingsphere.datasource.sdb0.password=root  #  #spring.shardingsphere.datasource.sdb1.type=com.alibaba.druid.pool.DruidDataSource  #spring.shardingsphere.datasource.sdb1.driver-class-name=com.mysql.cj.jdbc.Driver  #spring.shardingsphere.datasource.sdb1.url=jdbc:mysql://192.168.37.128:3306/sdb1?useUnicode=true&characterEncoding=utf-8&useSSL=false&serverTimezone=GMT  #spring.shardingsphere.datasource.sdb1.username=root  #spring.shardingsphere.datasource.sdb1.password=root  #  #spring.shardingsphere.masterslave.name=ms  ##spring.shardingsphere.masterslave.master-data-source-name=db0  ##spring.shardingsphere.masterslave.slave-data-source-names=sdb0,sdb1  #spring.shardingsphere.sharding.master-slave-rules.db0.master-data-source-name=db0  #spring.shardingsphere.sharding.master-slave-rules.db0.slave-data-source-names=sdb0  #spring.shardingsphere.sharding.master-slave-rules.db1.master-data-source-name=db1  #spring.shardingsphere.sharding.master-slave-rules.db1.slave-data-source-names=sdb1  #  ### 主键生成策略：雪花算法  #spring.shardingsphere.sharding.tables.user.key-generator.column=id  #spring.shardingsphere.sharding.tables.user.key-generator.type=SNOWFLAKE  #  ### 打印执行的数据库以及语句  #spring.shardingsphere.props.sql.show=true  #spring.main.allow-bean-definition-overriding=true  # mysql 同时实现分库分表和读写分离  spring.shardingsphere.datasource.names=db0,db1,sdb0,sdb1  spring.shardingsphere.datasource.db0.type=com.alibaba.druid.pool.DruidDataSource  spring.shardingsphere.datasource.db0.driver-class-name=com.mysql.jdbc.Driver  spring.shardingsphere.datasource.db0.url=jdbc:mysql://192.168.37.128:3306/db0?useUnicode=true&characterEncoding=utf-8&useSSL=false&serverTimezone=GMT  spring.shardingsphere.datasource.db0.username=root  spring.shardingsphere.datasource.db0.password=root  spring.shardingsphere.datasource.db1.type=com.alibaba.druid.pool.DruidDataSource  spring.shardingsphere.datasource.db1.driver-class-name=com.mysql.jdbc.Driver  spring.shardingsphere.datasource.db1.url=jdbc:mysql://192.168.37.128:3306/db1?useUnicode=true&characterEncoding=utf-8&useSSL=false&serverTimezone=GMT  spring.shardingsphere.datasource.db1.username=root  spring.shardingsphere.datasource.db1.password=root  spring.shardingsphere.datasource.sdb0.type=com.alibaba.druid.pool.DruidDataSource  spring.shardingsphere.datasource.sdb0.driver-class-name=com.mysql.jdbc.Driver  spring.shardingsphere.datasource.sdb0.url=jdbc:mysql://192.168.37.128:3306/db0?useUnicode=true&characterEncoding=utf-8&useSSL=false&serverTimezone=GMT  spring.shardingsphere.datasource.sdb0.username=root  spring.shardingsphere.datasource.sdb0.password=root  spring.shardingsphere.datasource.sdb1.type=com.alibaba.druid.pool.DruidDataSource  spring.shardingsphere.datasource.sdb1.driver-class-name=com.mysql.jdbc.Driver  spring.shardingsphere.datasource.sdb1.url=jdbc:mysql://192.168.37.128:3306/db1?useUnicode=true&characterEncoding=utf-8&useSSL=false&serverTimezone=GMT  spring.shardingsphere.datasource.sdb1.username=root  spring.shardingsphere.datasource.sdb1.password=root  # 分库分表策略  spring.shardingsphere.sharding.default-database-strategy.inline.sharding-column=id  spring.shardingsphere.sharding.default-database-strategy.inline.algorithm-expression=db$->{id % 2}  spring.shardingsphere.sharding.tables.user.actual-data-nodes=db$->{0..1}.user\_$->{0..1}  spring.shardingsphere.sharding.tables.user.table-strategy.inline.sharding-column=age  spring.shardingsphere.sharding.tables.user.table-strategy.inline.algorithm-expression=user\_$->{age % 2}  # 读写分离策略  spring.shardingsphere.masterslave.load-balance-algorithm-type=round\_robin  spring.shardingsphere.masterslave.name=ms  spring.shardingsphere.sharding.master-slave-rules.db0.master-data-source-name=db0  spring.shardingsphere.sharding.master-slave-rules.db0.slave-data-source-names=sdb0  spring.shardingsphere.sharding.master-slave-rules.db1.master-data-source-name=db1  spring.shardingsphere.sharding.master-slave-rules.db1.slave-data-source-names=sdb1  # 主键生成策略：雪花算法  spring.shardingsphere.sharding.tables.user.key-generator.column=id  spring.shardingsphere.sharding.tables.user.key-generator.type=SNOWFLAKE  # 打印执行的数据库以及语句  spring.shardingsphere.props.sql.show=true  spring.main.allow-bean-definition-overriding=true |