# kubeadm 安装 k8s1.25 高可用集群

**k8s 环境规划：**

podSubnet（pod 网段） 10.244.0.0/16

serviceSubnet（service 网段）: 10.96.0.0/12

**实验环境规划：**

**操作系统**：centos7.9

**配置**： 4Gib 内存/4vCPU/60G 硬盘

**网络**：NAT 模式

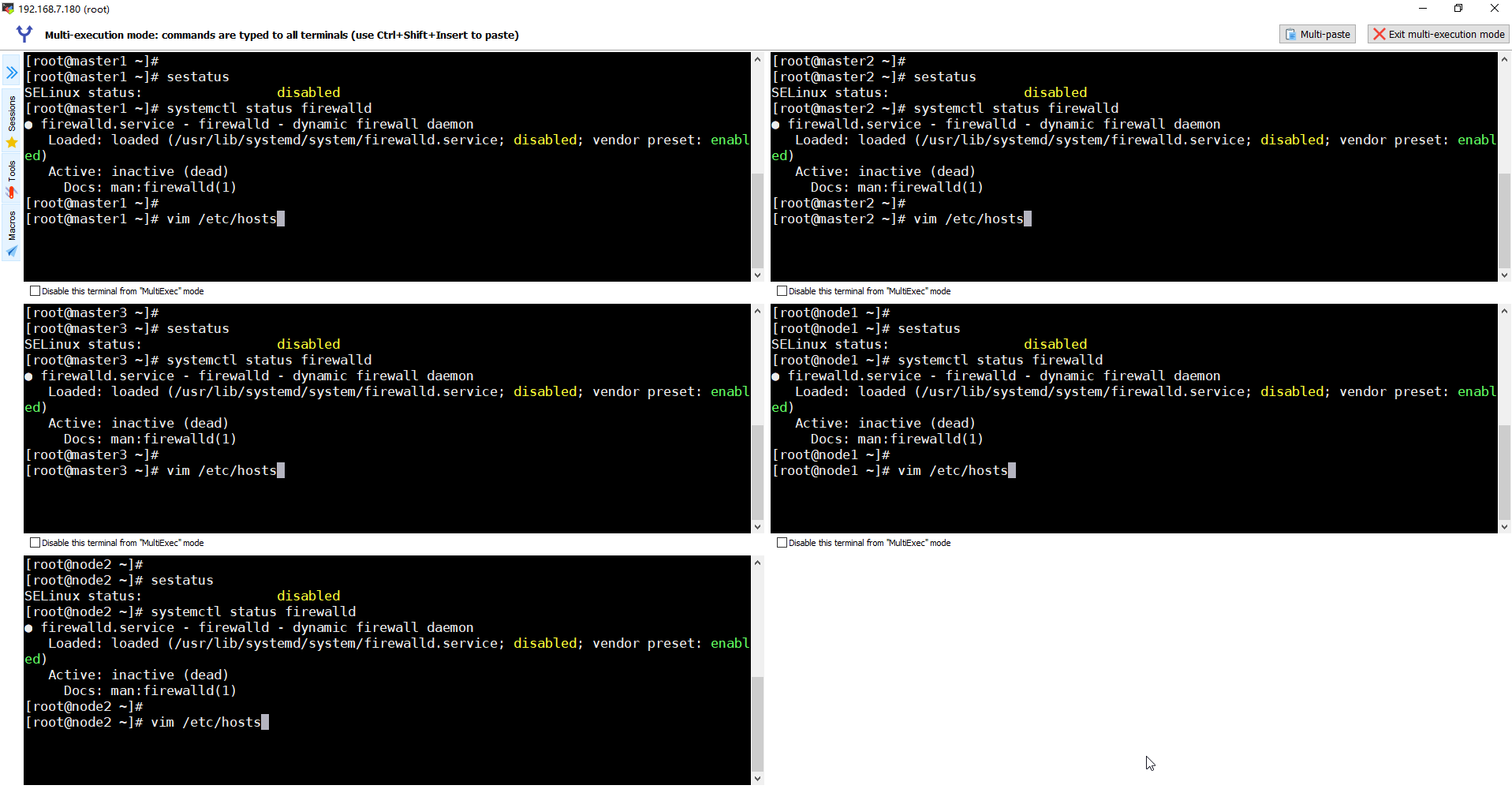
**K8S 集群角色 IP 主机名 安装的组件**

**控制节点 192.168.7.180 master1:** apiserver、controller-manager、schedule、kubelet、etcd、kube-proxy、容器运行时、calico、keepalived、nginx

**控制节点 192.168.7.181 master2:**apiserver、controller-manager、schedule、kubelet、etcd、kube-proxy、容器运行时、calico、keepalived、nginx

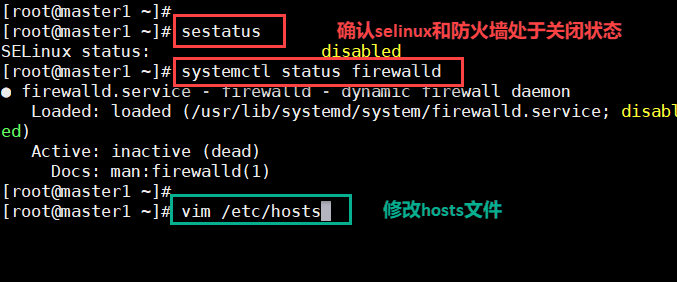
**控制节点 192.168.7.182 master3:** apiserver、controller-manager、schedule、kubelet、etcd、kube-proxy、容器运行时、calico、keepalived、nginx

**工作节点 192.168.7.183 node1:** Kube-proxy、calico、coredns、容器运行时、kubeletVIP 192.168.7.199



## 1关闭防火墙和selinux

## 2修改hosts文件



**配置主机 hosts 文件，相互之间通过主机名互相访问**

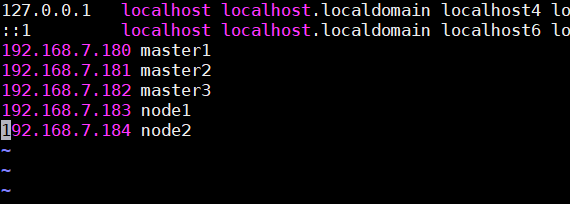
**修改每台机器的/etc/hosts 文件，文件最后增加如下内容：**

**192.168.7.180 master1**

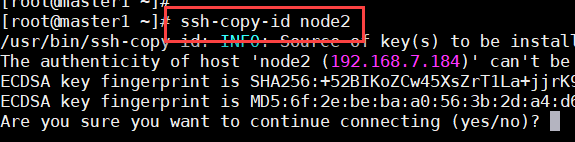
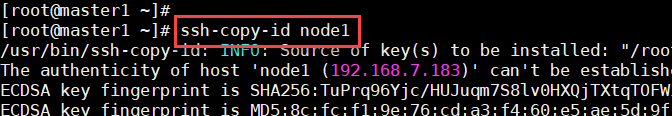
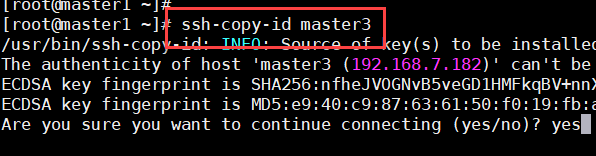
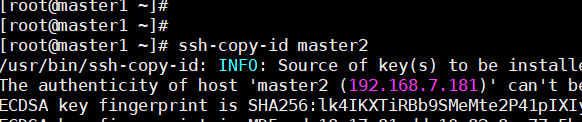
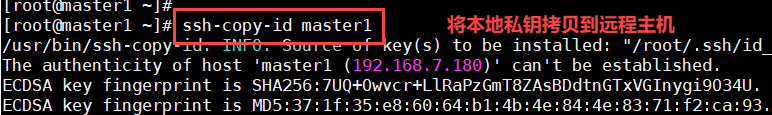
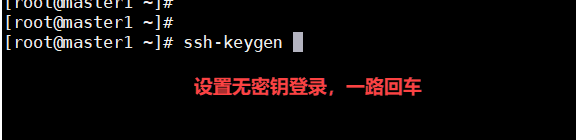
**192.168.7.181 master2**

**192.168.7.182 master3**

**192.168.7.183 node1**

**192.168.7.184** node2 

## 3 所有节点配置主机之间无密码登录，把本地生成的密钥文件和私钥文件拷贝到远程主机



**配置 master1 到其他机器免密登录**

**[root@master1 ~]# ssh-keygen #一路回车，不输入密码**

**把本地生成的密钥文件和私钥文件拷贝到远程主机**

**[root@master1 ~]# ssh-copy-id master1**

**[root@master1 ~]# ssh-copy-id master2**

**[root@master1 ~]# ssh-copy-id master3**

**[root@master1 ~]# ssh-copy-id node1**

**[root@master1 ~]# ssh-copy-id node2**

**配置 master2 到其他机器免密登录**

**[root@master2 ~]# ssh-keygen #一路回车，不输入密码**

**把本地生成的密钥文件和私钥文件拷贝到远程主机**

**[root@master2 ~]# ssh-copy-id master1**

**[root@master2 ~]# ssh-copy-id master2**

**[root@master2 ~]# ssh-copy-id master3**

**[root@master2 ~]# ssh-copy-id node1**

**[root@master2 ~]# ssh-copy-id node2**

**配置 master3 到其他机器免密登录**

**[root@master3 ~]# ssh-keygen #一路回车，不输入密码**

**把本地生成的密钥文件和私钥文件拷贝到远程主机**

**[root@master3 ~]# ssh-copy-id master1**

**[root@master3 ~]# ssh-copy-id master2**

**[root@master3~]# ssh-copy-id master3**

**[root@master3 ~]# ssh-copy-id node1**

**[root@master3 ~]# ssh-copy-id node2**

**配置 node1 到其他机器免密登录**

**[root@node1 ~]# ssh-keygen #一路回车，不输入密码**

**把本地生成的密钥文件和私钥文件拷贝到远程主机**

**[root@node1 ~]# ssh-copy-id master1**

**[root@node1 ~]# ssh-copy-id master2**

**[root@node1 ~]# ssh-copy-id master3**

**[root@node1 ~]# ssh-copy-id node1**

**[root@node1 ~]# ssh-copy-id node2**

**root@node2 ~]# ssh-keygen #一路回车，不输入密码**

**把本地生成的密钥文件和私钥文件拷贝到远程主机**

**[root@node2 ~]# ssh-copy-id master1**

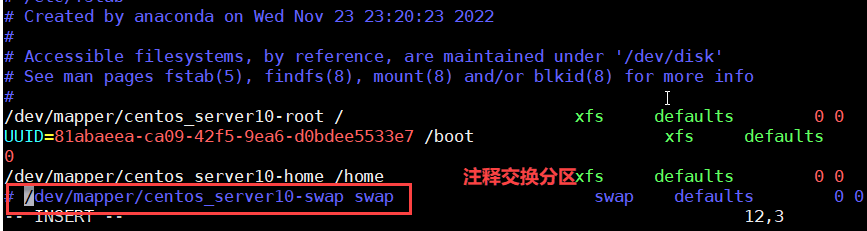
**[root@node2 ~]# ssh-copy-id master2**

**[root@node2 ~]# ssh-copy-id master3**

**[root@node2 ~]# ssh-copy-id node1**

**[root@node2 ~]# ssh-copy-id node2**

## 4.所有节点永久关闭交换分区，提升性能



问题 1：为什么要关闭 swap 交换分区？

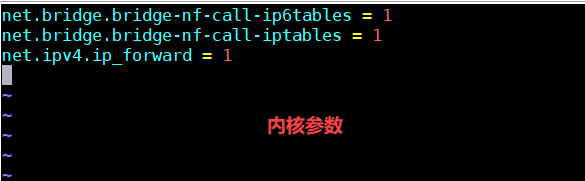
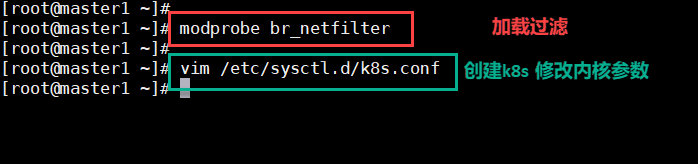
Swap 是交换分区，如果机器内存不够，会使用 swap 分区，但是 swap 分区的性能较低，k8s 设

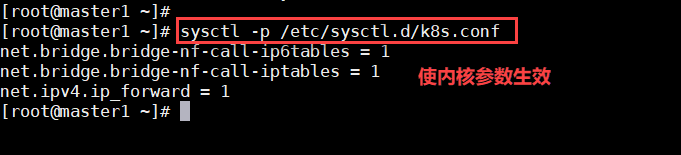
计的时候为了能提升性能，默认是不允许使用交换分区的。Kubeadm 初始化的时候会检测 swap 是否

关闭，如果没关闭，那就初始化失败。如果不想要关闭交换分区，安装 k8s 的时候可以指定--

ignore-preflight-errors=Swap 来解决。

## 5.修改所有节点内核参数





**问题 1：sysctl 是做什么的？**

**在运行时配置内核参数**

**-p 从指定的文件加载系统参数，如不指定即从/etc/sysctl.conf 中加载**

**问题 2：为什么要执行 modprobe br\_netfilter？**

**修改/etc/sysctl.d/k8s.conf 文件，增加如下三行参数：**

**net.bridge.bridge-nf-call-ip6tables = 1**

**net.bridge.bridge-nf-call-iptables = 1**

**net.ipv4.ip\_forward = 1**

**sysctl -p /etc/sysctl.d/k8s.conf 出现报错：**

**sysctl: cannot stat /proc/sys/net/bridge/bridge-nf-call-ip6tables: No such file or**

**directory**

**sysctl: cannot stat /proc/sys/net/bridge/bridge-nf-call-iptables: No such file or**

**directory**

**解决方法：**

**modprobe br\_netfilter**

**问题 3：为什么开启 net.bridge.bridge-nf-call-iptables 内核参数？**

**在 centos 下安装 docker，执行 docker info 出现如下警告：**

**WARNING: bridge-nf-call-iptables is disabled**

**WARNING: bridge-nf-call-ip6tables is disabled**

**解决办法：**

**vim /etc/sysctl.d/k8s.conf**

**net.bridge.bridge-nf-call-ip6tables = 1**

**net.bridge.bridge-nf-call-iptables = 1**

**问题 4：为什么要开启 net.ipv4.ip\_forward = 1 参数？**

**kubeadm 初始化 k8s 如果报错：**

**就表示没有开启 ip\_forward，需要开启。**

**net.ipv4.ip\_forward 是数据包转发：**

**出于安全考虑，Linux 系统默认是禁止数据包转发的。所谓转发即当主机拥有多于一块的网卡时，**

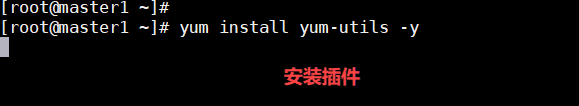
**其中一块收到数据包，根据数据包的目的 ip 地址将数据包发往本机另一块网卡，该网卡根据路由表继续**

**发送数据包。这通常是路由器所要实现的功能。**

**要让 Linux 系统具有路由转发功能，需要配置一个 Linux 的内核参数 net.ipv4.ip\_forward。这个**

**参数指定了 Linux 系统当前对路由转发功能的支持情况；其值为 0 时表示禁止进行 IP 转发；如果是 1,**

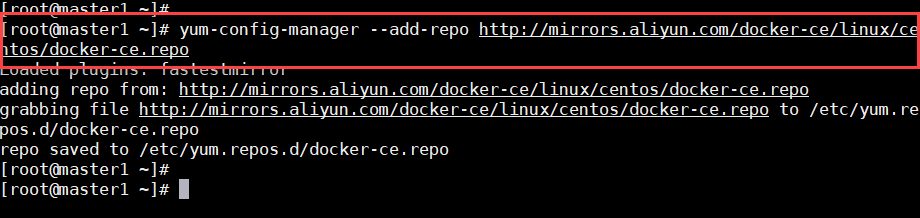
**则说明 IP 转发功能已经打开。**



## 6.在所有节点配置阿里云的repo源

#配置国内安装docker和containerd的阿里云的repo源

**yum-config-manager --add-repo http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo**



## 7.配置安装K8S组件需要的阿里云的repo源

**cat <<EOF > /etc/yum.repos.d/kubernetes.repo**

**[kubernetes]**

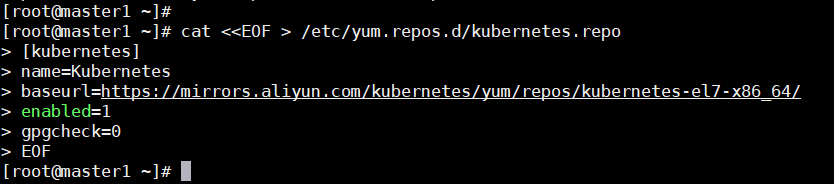
**name=Kubernetes**

**baseurl=https://mirrors.aliyun.com/kubernetes/yum/repos/kubernetes-el7-x86\_64/**

**enabled=1**

**gpgcheck=0**

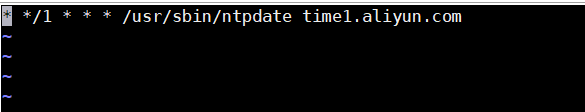
**EOF**



## 8.所有节点配置时间同步

## 9.所有节点安装基础软件包

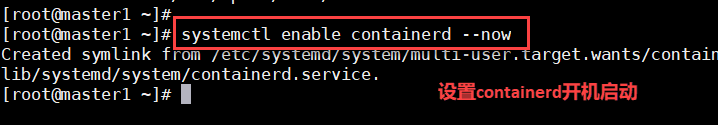
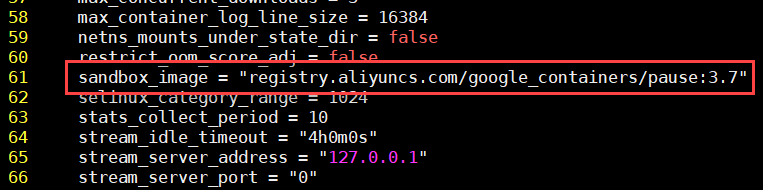
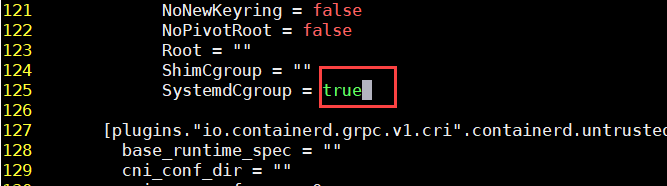
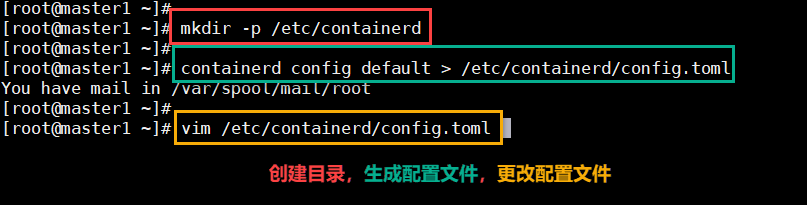
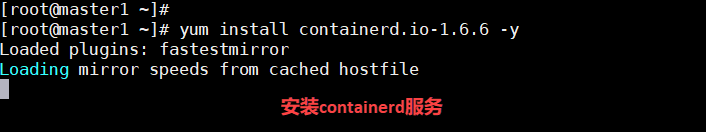




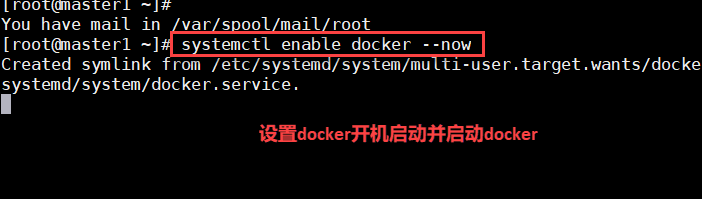
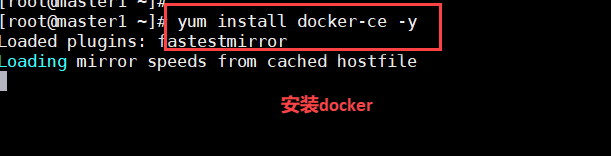
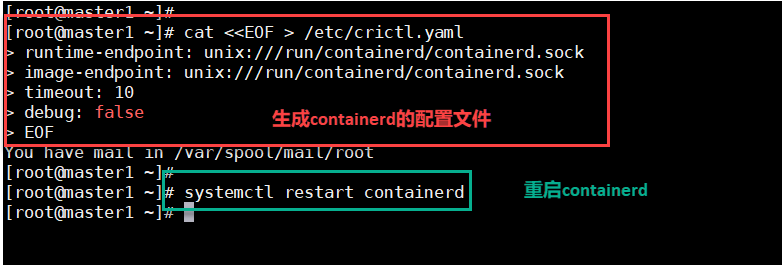
**yum install -y device-mapper-persistent-date lvm2 wget net-tools nfs-utils lrzsz gcc gcc-c++ make cmake libxml2-devel openssl-devel curl curl-devel unzip sudo ntp libaio-devel wget vim ncurses-devel autoconf automake zlib-devel python-devel epel-release openssh-server socat lpvsadm conntrack telnet ipvsadm**

## 10.所有节点安装 containerd 服务

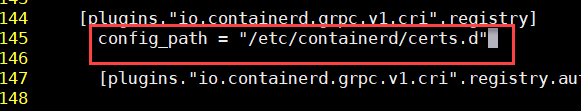
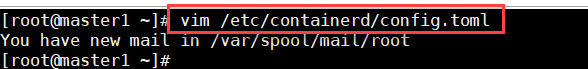
由于自 1.24 版起，Dockershim 已从 Kubernetes 项目中移除，所以我们的 \*\*容器运行时（容器运行时负责运行容器的软件）\*\* 已不在是docker。本文将采用containerd作为 \*\*容器运行时\*\*。



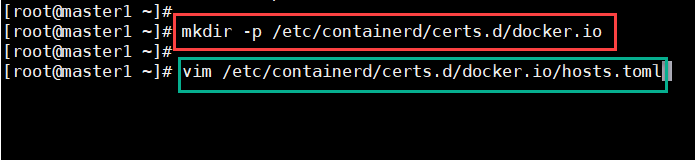
## 11.所有节点修改/etc/crictl.yaml 文件，修改/etc/crictl.yaml

**cat <<EOF > /etc/crictl.yaml   
runtime-endpoint: unix:///run/containerd/containerd.sock  
image-endpoint: unix:///run/containerd/containerd.sock  
timeout: 10  
debug: false  
EOF**

**配置 containerd 镜像加速器**

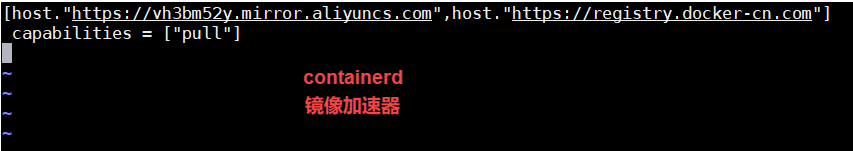
****

## 12.配置containerd镜像加速器

****

**[host."https://vh3bm52y.mirror.aliyuncs.com",host."https://registry.docker-cn.com"]**

**capabilities = ["pull"]**

****

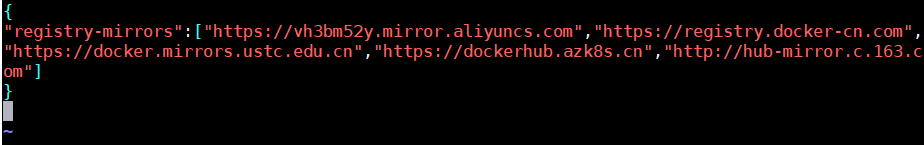
## 13.配置docker镜像加速器

****

**{**

**"registry-mirrors":["https://vh3bm52y.mirror.aliyuncs.com","https://registry.docker-cn.com","https://docker.mirrors.ustc.edu.cn","https://dockerhub.azk8s.cn","http://hub-mirror.c.163.com"]**

**}**

****

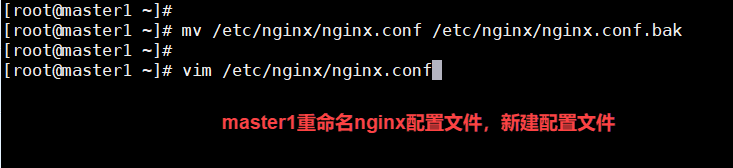
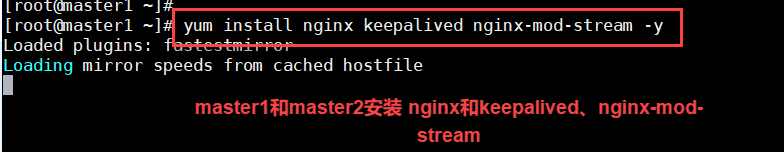
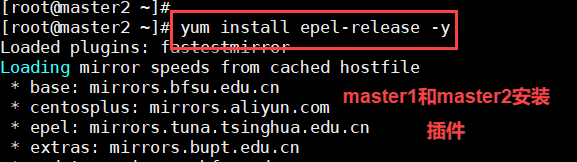
## 14.安装初始化k8s需要的软件包

**注：每个软件包的作用：**

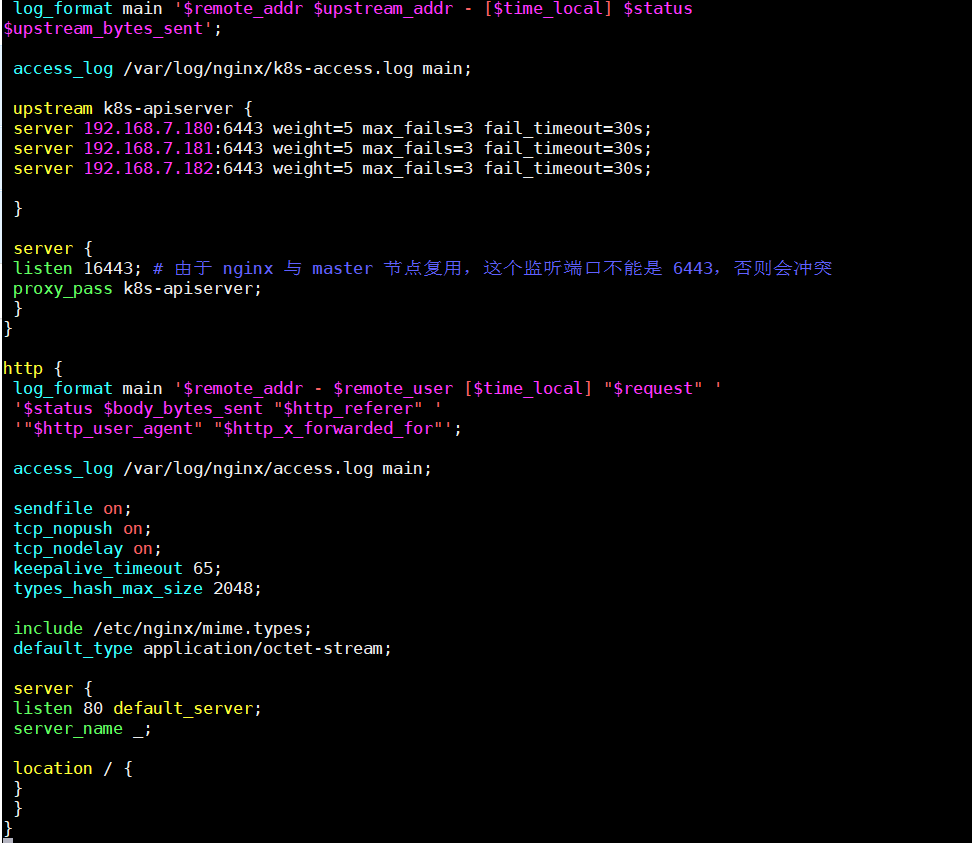
Kubeadm: kubeadm 是一个工具，用来初始化 k8s 集群的

kubelet: 安装在集群所有节点上，用于启动 Pod 的，kubeadm 安装 k8s，k8s 控制节点和工作节点的组件，都是基于 pod 运行的，只要 pod 启动，就需要 kubelet

kubectl: 通过 kubectl 可以部署和管理应用，查看各种资源，创建、删除和更新各种组件

****

## 15. 通过 keepalived+nginx 实现 k8s api-server 节点高可用

****

**user nginx;**

**worker\_processes auto;**

**error\_log /var/log/nginx/error.log;**

**pid /run/nginx.pid;**

**include /usr/share/nginx/modules/\*.conf;**

**events {**

**worker\_connections 1024;**

**}**

**# 四层负载均衡，为两台 Master apiserver 组件提供负载均衡**

**stream {**

**log\_format main '$remote\_addr $upstream\_addr - [$time\_local] $status**

**$upstream\_bytes\_sent';**

**access\_log /var/log/nginx/k8s-access.log main;**

**upstream k8s-apiserver {**

**server 192.168.7.180:6443 weight=5 max\_fails=3 fail\_timeout=30s;**

**server 192.168.7.181:6443 weight=5 max\_fails=3 fail\_timeout=30s;**

**server 192.168.7.182:6443 weight=5 max\_fails=3 fail\_timeout=30s;**

**}**

**server {**

**listen 16443; # 由于 nginx 与 master 节点复用，这个监听端口不能是 6443，否则会冲突**

**proxy\_pass k8s-apiserver;**

**}**

**}**

**http {**

**log\_format main '$remote\_addr - $remote\_user [$time\_local] "$request" '**

**'$status $body\_bytes\_sent "$http\_referer" '**

**'"$http\_user\_agent" "$http\_x\_forwarded\_for"';**

**access\_log /var/log/nginx/access.log main;**

**sendfile on;**

**tcp\_nopush on;**

**tcp\_nodelay on;**

**keepalive\_timeout 65;**

**types\_hash\_max\_size 2048;**

**include /etc/nginx/mime.types;**

**default\_type application/octet-stream;**

**server {**

**listen 80 default\_server;**

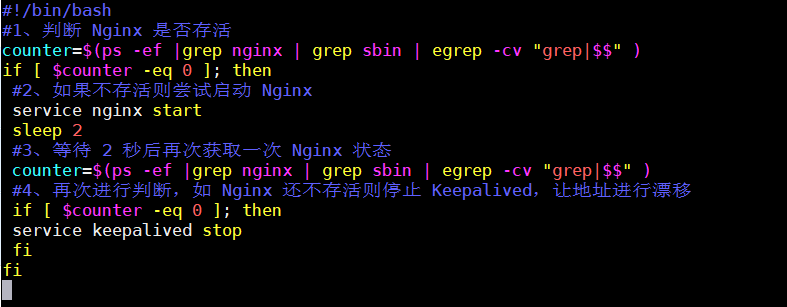
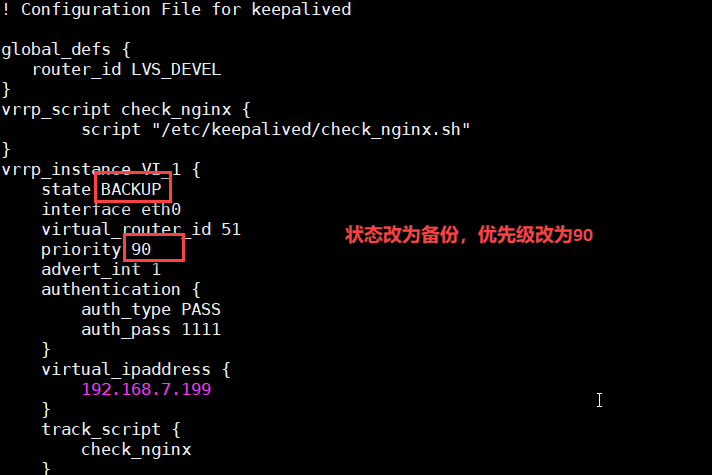
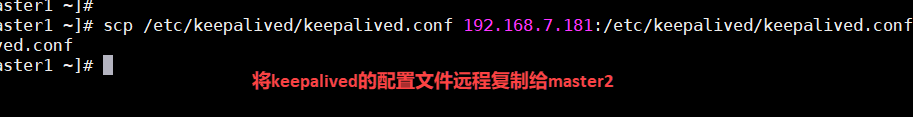
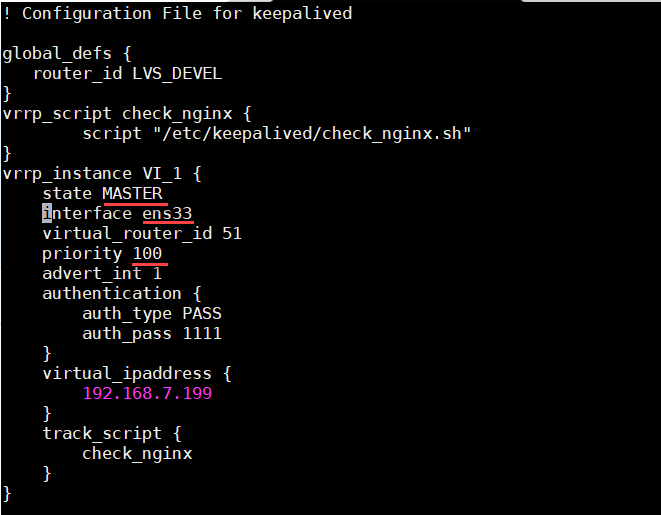
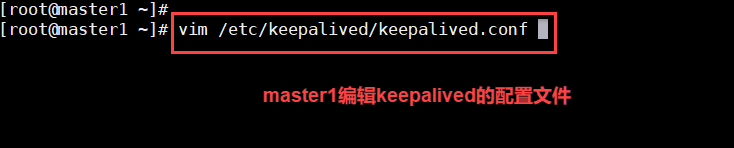
**server\_name \_;**

**location / {**

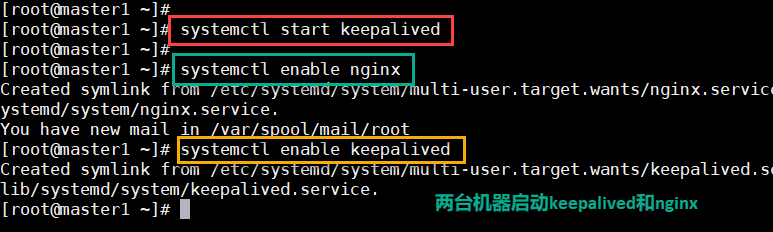
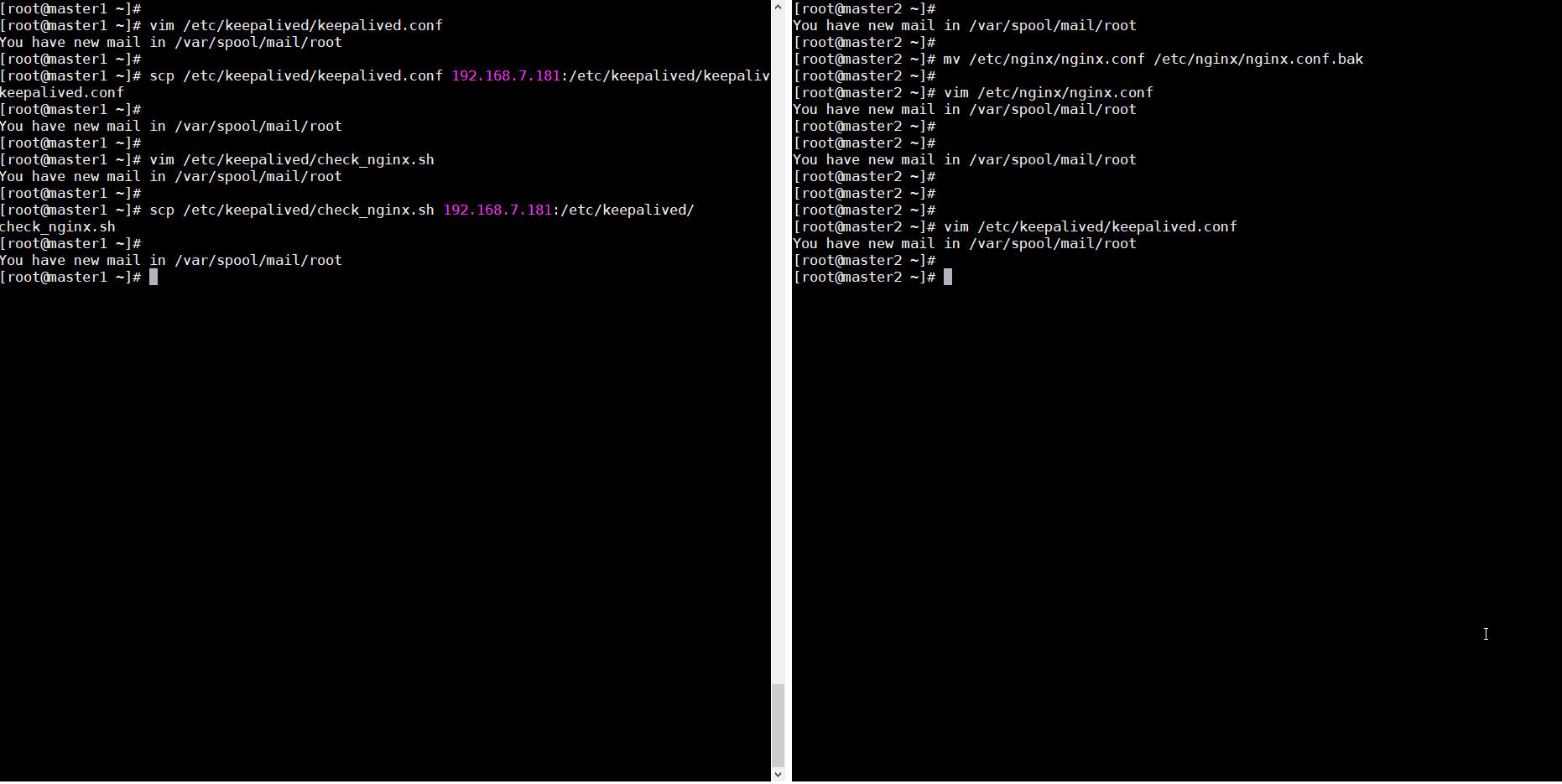
**}**

**}**

**}**

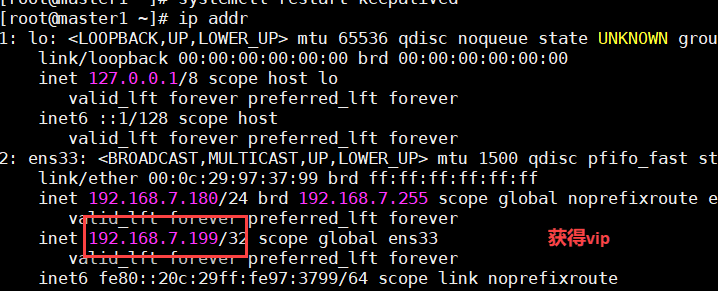
****

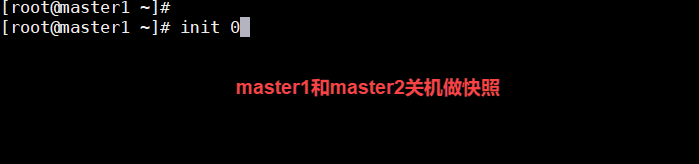
**#!/bin/bash   
#1、判断 Nginx 是否存活   
counter=$(ps -ef |grep nginx | grep sbin | egrep -cv "grep|$$" )   
if [ $counter -eq 0 ]; then   
 #2、如果不存活则尝试启动 Nginx   
 service nginx start   
 sleep 2   
 #3、等待 2 秒后再次获取一次 Nginx 状态   
 counter=$(ps -ef |grep nginx | grep sbin | egrep -cv "grep|$$" )   
 #4、再次进行判断，如 Nginx 还不存活则停止 Keepalived，让地址进行漂移   
 if [ $counter -eq 0 ]; then   
 service keepalived stop   
 fi   
fi**

****

**加一条systemctl daemon-reload**

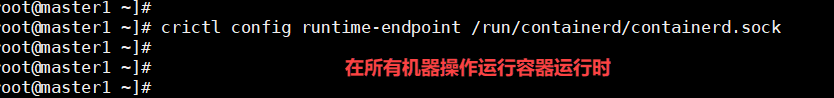
**测试 vip 是否绑定成功：**

****

****

## 16.kubeadm初始化k8s集群

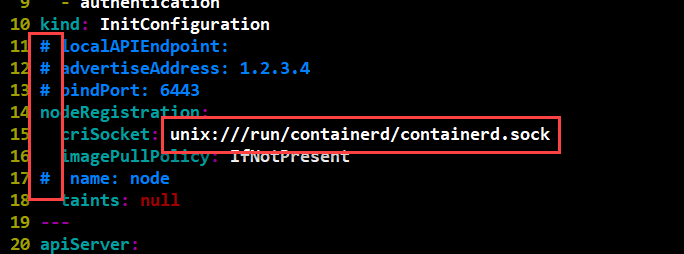
**#设置容器运行时**

****

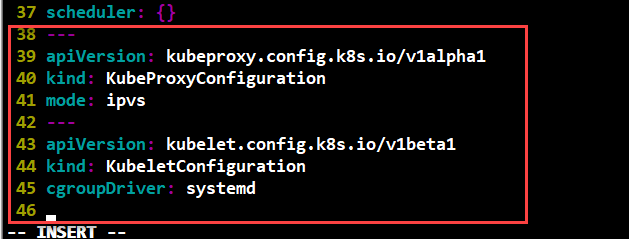
**kubeadm初始化k8s集群**

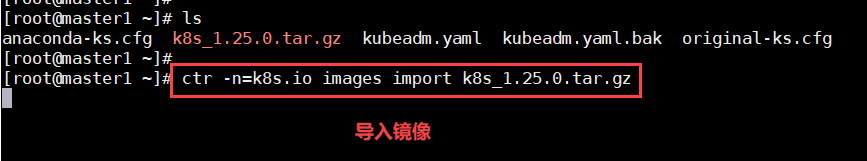
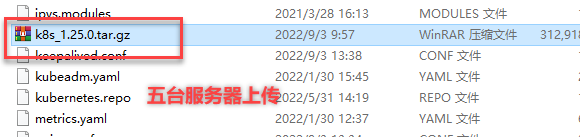
**回到master1节点操作：kubeadm config print init-defaults > kubeadm.yaml**

****

****

**registry.cn-hangzhou.aliyuncs.com/google\_containers**

****

****

**备注：k8s\_1.25.0.tar.gz 这个文件如何来的？**

**这个文件把安装 k8s 需要的镜像都继承好了，这个是我第一次安装 1.25.0 这个版本，获取到对应的**

**镜像，通过 ctr images export 这个命令把镜像输出到 k8s\_1.25.0.tar.gz 文件，如果大家安装其他版**

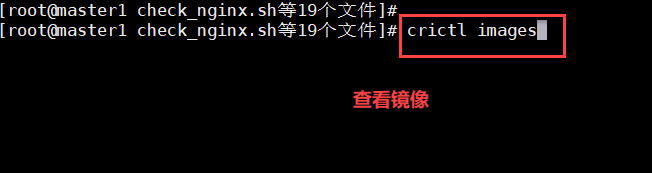
**本，那就不需要实现解压镜像，可以默认从网络拉取镜像即可。**

**ctr 是 containerd 自带的工具，有命名空间的概念，若是 k8s 相关的镜像，都默认在 k8s.io 这个**

**命名空间，所以导入镜像时需要指定命令空间为 k8s.io**

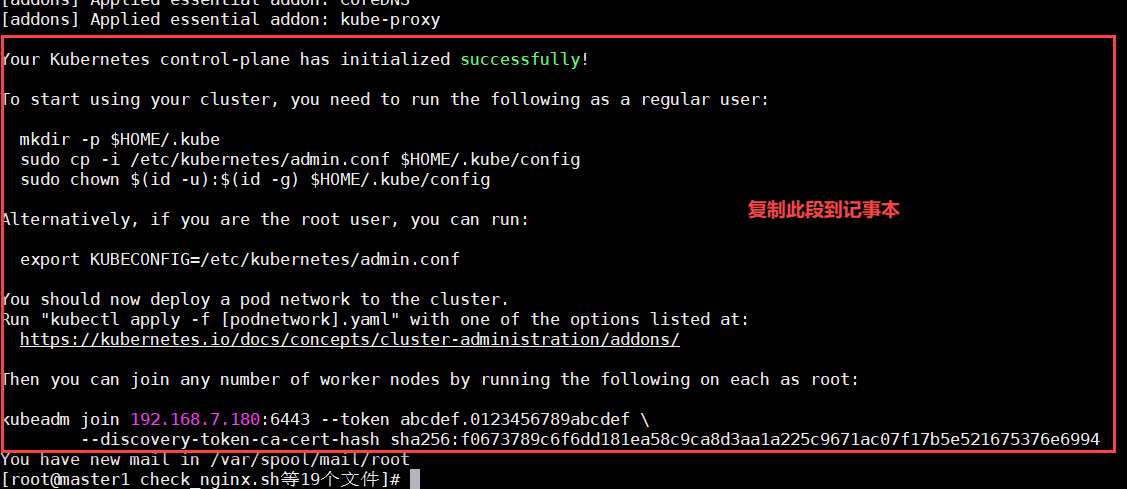
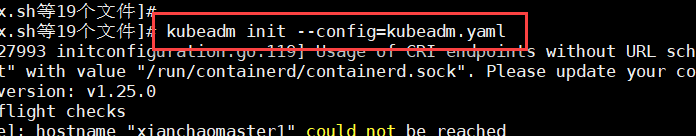
**#使用 ctr 命令指定命名空间导入镜像**

**ctr -n=k8s.io images import k8s\_1.25.0.tar.gz**

****

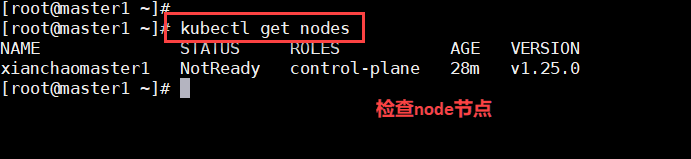
## 17.基于kubeadm.yaml初始化k8s集群

**切换到master1服务器下 cd退回到root目录下：**

****

## 18.配置kubectl的配置文件config

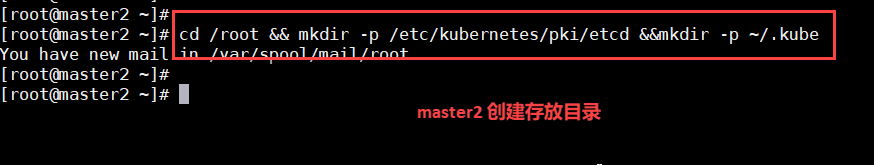
命令可在前一步复制的内容中粘贴

****

**若不成功可以查看一下kubelet状态，start启动一下**

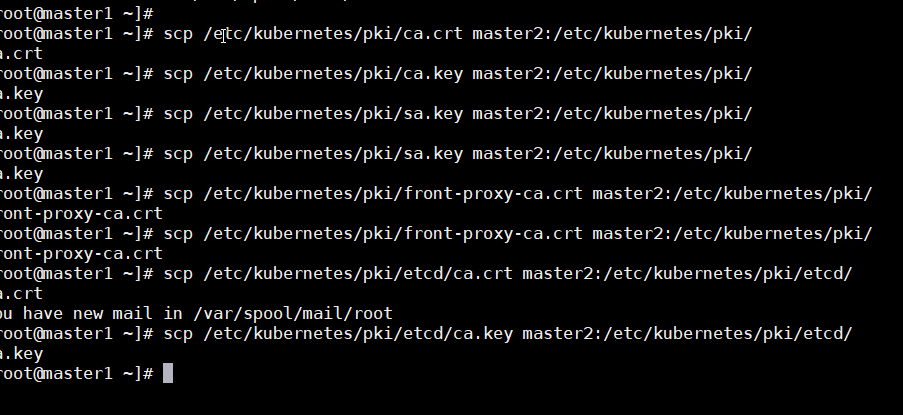
## 19. 扩容 k8s master 节点-把 master2 添加到 K8s 集群把master1节点的证书拷贝到master2上

**创建master2证书存放的目录**

****

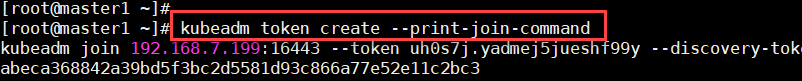
**cd /root && mkdir -p /etc/kubernetes/pki/etcd &&mkdir -p ~/.kube/**

**拷贝：**

****

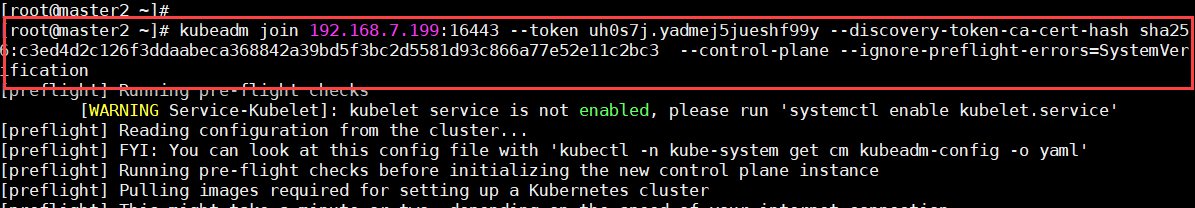
**#把 master1 节点的证书拷贝到 master2 上：**  
**scp /etc/kubernetes/pki/ca.crt master2:/etc/kubernetes/pki/   
scp /etc/kubernetes/pki/ca.key master2:/etc/kubernetes/pki/   
scp /etc/kubernetes/pki/sa.key master2:/etc/kubernetes/pki/   
scp /etc/kubernetes/pki/sa.pub master2:/etc/kubernetes/pki/   
scp /etc/kubernetes/pki/front-proxy-ca.crt master2:/etc/kubernetes/pki/   
scp /etc/kubernetes/pki/front-proxy-ca.key master2:/etc/kubernetes/pki/   
scp /etc/kubernetes/pki/etcd/ca.crt master2:/etc/kubernetes/pki/etcd/   
scp /etc/kubernetes/pki/etcd/ca.key master2:/etc/kubernetes/pki/etcd/**

**在master1节点查看加入节点的命令：**

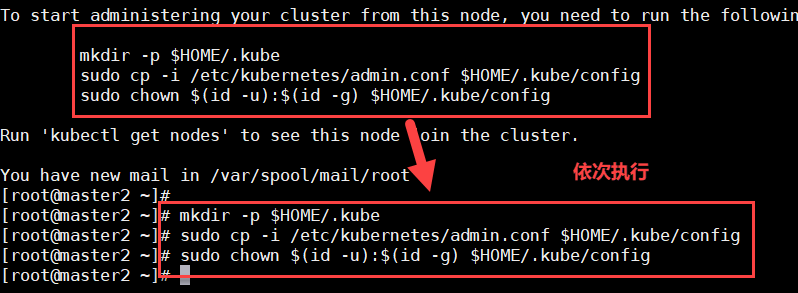
****

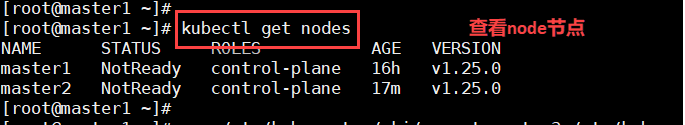
**将命令添加下列参数后缀后拷贝到master2：**

**--control-plane --ignore-preflight-errors=SystemVerification**

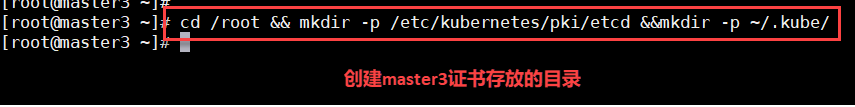
****

**按照结果依次执行：**

****

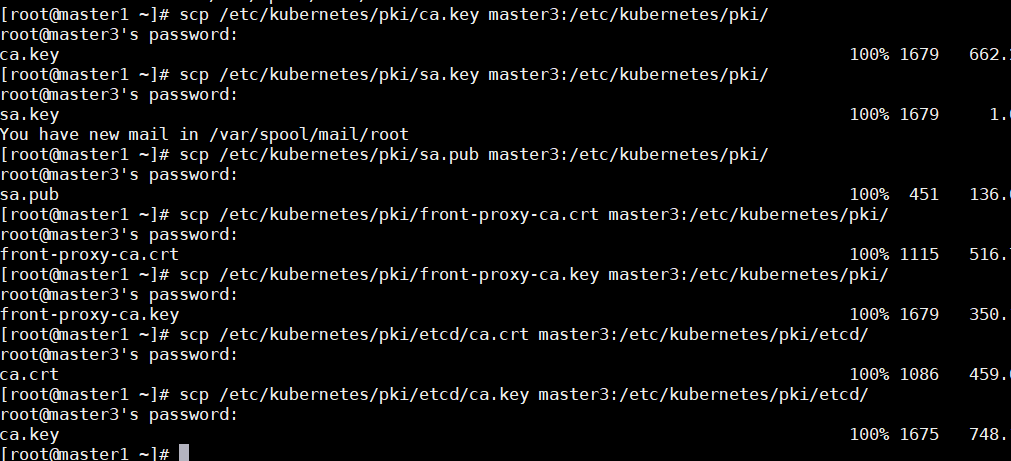
****

**Master2已成功加入节点，接下来将master3加入节点：**

****

**cd /root && mkdir -p /etc/kubernetes/pki/etcd &&mkdir -p ~/.kube/**

**#把 master1 节点的证书拷贝到 master3 上**：

****

**scp /etc/kubernetes/pki/ca.crt master3:/etc/kubernetes/pki/**

**scp /etc/kubernetes/pki/ca.key master3:/etc/kubernetes/pki/**

**scp /etc/kubernetes/pki/sa.key master3:/etc/kubernetes/pki/**

**scp /etc/kubernetes/pki/sa.pub master3:/etc/kubernetes/pki/**

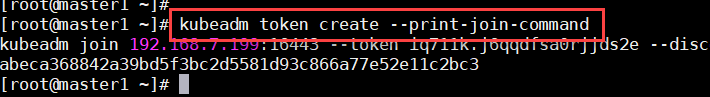
**scp /etc/kubernetes/pki/front-proxy-ca.crt master3:/etc/kubernetes/pki/**

**scp /etc/kubernetes/pki/front-proxy-ca.key master3:/etc/kubernetes/pki/**

**scp /etc/kubernetes/pki/etcd/ca.crt master3:/etc/kubernetes/pki/etcd/**

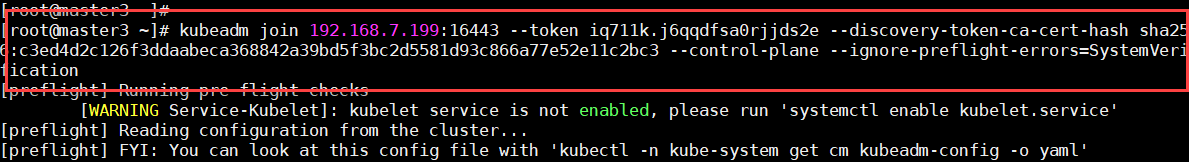
**scp /etc/kubernetes/pki/etcd/ca.key master3:/etc/kubernetes/pki/etcd/**

**在master1节点查看加入节点的命令：**

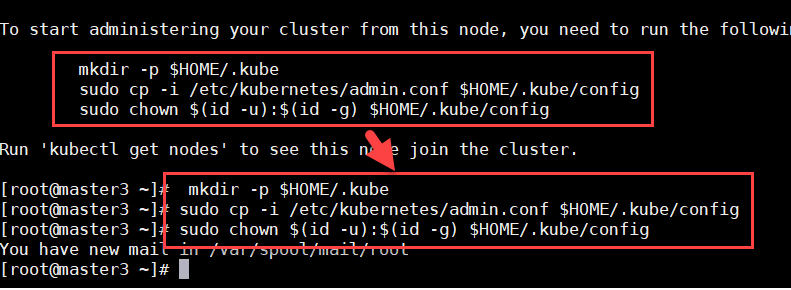
****

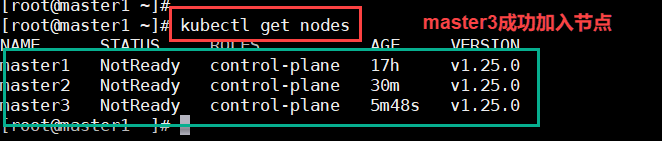
**将命令添加下列参数后缀后拷贝到master3：**

**--control-plane --ignore-preflight-errors=SystemVerification**

****

**按照结果依次执行：**

****



按照上述方法将node1和node2加入节点：

1. 在node1和node2创建存放证书的目录
2. 在master1将证书拷贝到node1和node2

**Node1**

**scp /etc/kubernetes/pki/ca.crt node1:/etc/kubernetes/pki/**

**scp /etc/kubernetes/pki/ca.key node1:/etc/kubernetes/pki/**

**scp /etc/kubernetes/pki/sa.key node1:/etc/kubernetes/pki/**

**scp /etc/kubernetes/pki/sa.pub node1:/etc/kubernetes/pki/**

**scp /etc/kubernetes/pki/front-proxy-ca.crt node1:/etc/kubernetes/pki/**

**scp /etc/kubernetes/pki/front-proxy-ca.key node1:/etc/kubernetes/pki/**

**scp /etc/kubernetes/pki/etcd/ca.crt node1:/etc/kubernetes/pki/etcd/**

**scp /etc/kubernetes/pki/etcd/ca.key node1:/etc/kubernetes/pki/etcd/**

**node2**

**scp /etc/kubernetes/pki/ca.crt node2:/etc/kubernetes/pki/**

**scp /etc/kubernetes/pki/ca.key node2:/etc/kubernetes/pki/**

**scp /etc/kubernetes/pki/sa.key node2:/etc/kubernetes/pki/**

**scp /etc/kubernetes/pki/sa.pub node2:/etc/kubernetes/pki/**

**scp /etc/kubernetes/pki/front-proxy-ca.crt node2:/etc/kubernetes/pki/**

**scp /etc/kubernetes/pki/front-proxy-ca.key node2:/etc/kubernetes/pki/**

**scp /etc/kubernetes/pki/etcd/ca.crt node2:/etc/kubernetes/pki/etcd/**

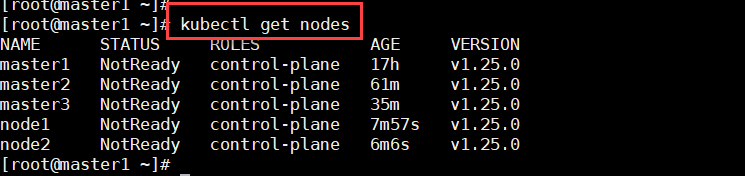
**scp /etc/kubernetes/pki/etcd/ca.key node2:/etc/kubernetes/pki/etcd/**

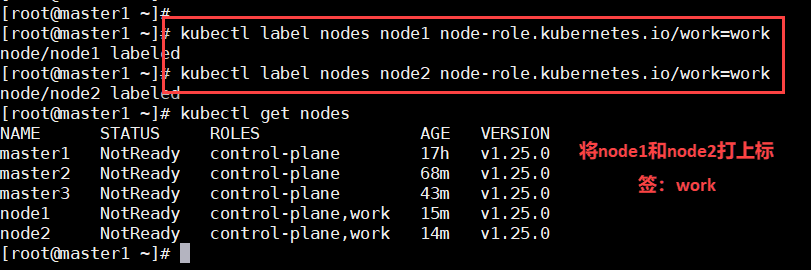
1. Master1查看加入节点的命令，加入参数拷贝到node1和node2
2. 依次执行命令

**mkdir -p $HOME/.kube**

**sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config**

**sudo chown $(id -u):$(id -g) $HOME/.kube/config**





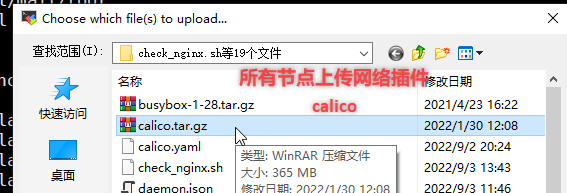
**kubectl label nodes node1 node-role.kubernetes.io/work=work**

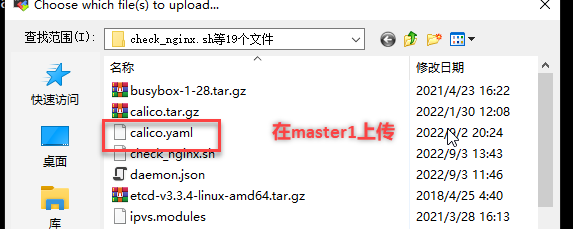
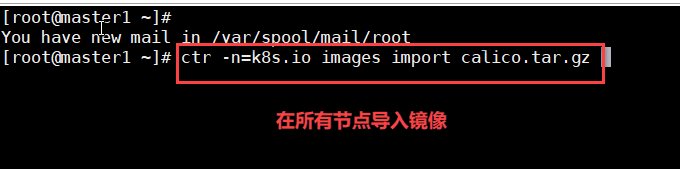
**kubectl label nodes node2 node-role.kubernetes.io/work=work**

**安装 kubernetes 网络组件-Calico**

**把安装 calico 需要的镜像 calico.tar.gz 传到 master1、master2、**

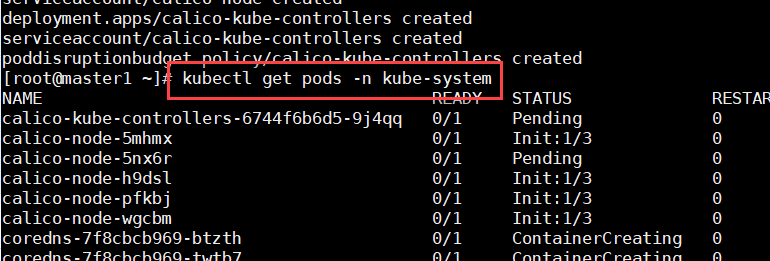
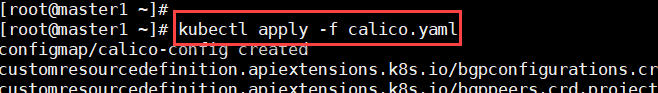
**master3 和 node1 节点，手动解压**：





**上传 calico.yaml 到 master1 上，使用 yaml 文件安装 calico 网络插件**

[root@master1 ~]# **kubectl apply -f calico.yaml**



**etcd 配置成高可用状态**

keepalived+nginx只是对api-server做高可用

修改 master1、master2、master3 上的 etcd.yaml 文件

**vim /etc/kubernetes/manifests/etcd.yaml**

把

**- --initial-cluster=master1=https://192.168.7.180:2380**

变成如下：

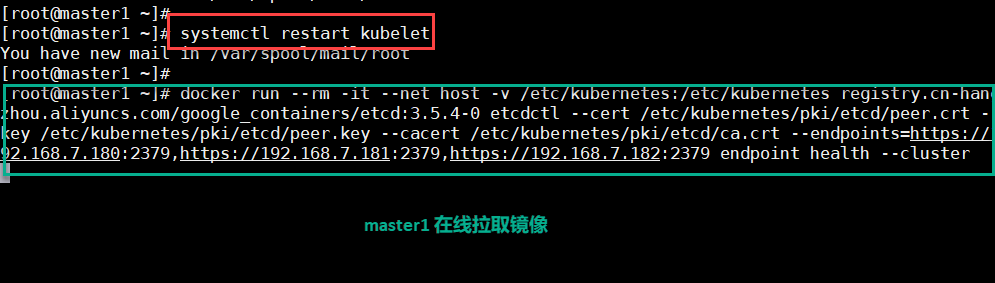
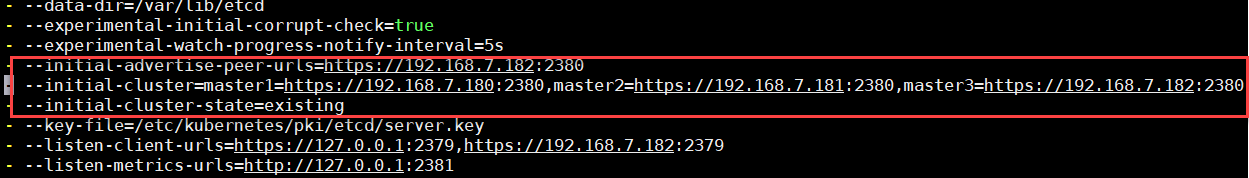
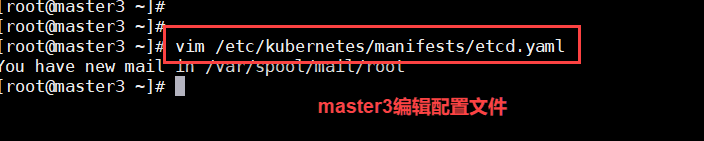
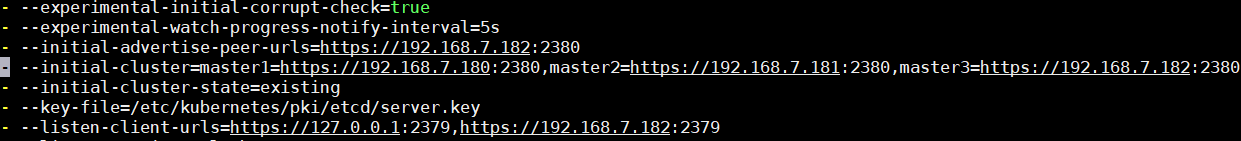
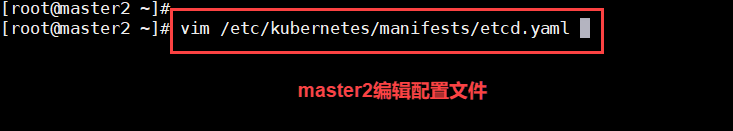
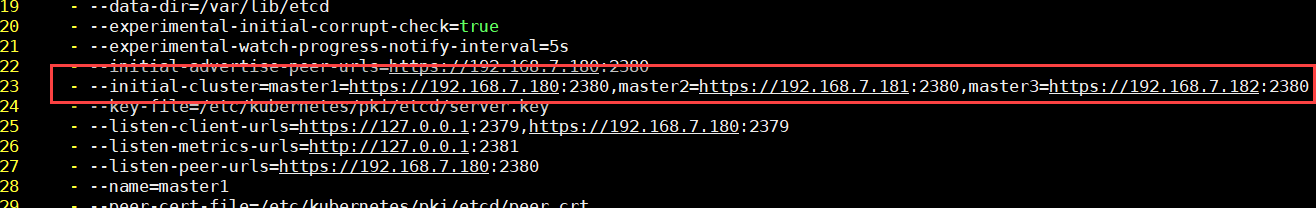
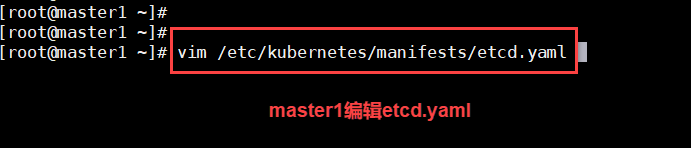
**---initial-cluster=master1=https://192.168.7.180:2380,master2=https://192.168.7.181:2380,master3=https://192.168.7.183:2380**

修改成功之后重启 kubelet：

[root@master1 ~]# **systemctl restart kubelet**

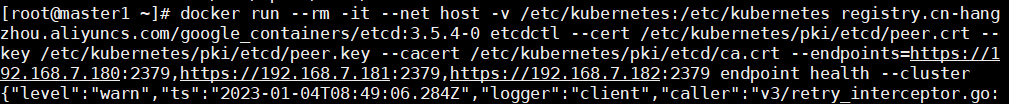
[root@master2 ~]# **systemctl restart kubelet**

[root@master3 ~]# **systemctl restart kubelet**

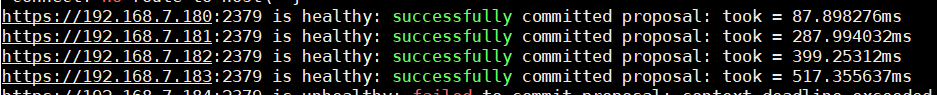


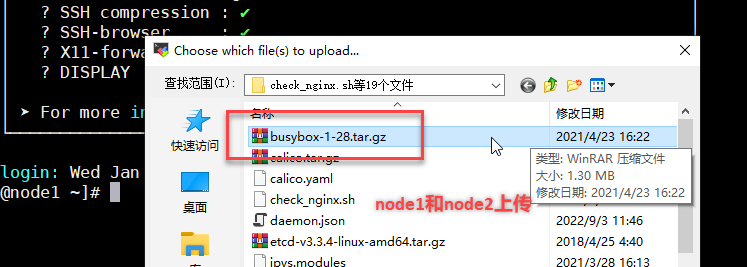
**测试 etcd 集群是否配置成功：**

**docker run --rm -it --net host -v /etc/kubernetes:/etc/kubernetes registry.cn-hangzhou.aliyuncs.com/google\_containers/etcd:3.5.4-0 etcdctl --cert /etc/kubernetes/pki/etcd/peer.crt --key /etc/kubernetes/pki/etcd/peer.key --cacert /etc/kubernetes/pki/etcd/ca.crt member list**

****

**docker run --rm -it --net host -v /etc/kubernetes:/etc/kubernetes registry.cn-hangzhou.aliyuncs.com/google\_containers/etcd:3.5.4-0 etcdctl --cert /etc/kubernetes/pki/etcd/peer.crt --key /etc/kubernetes/pki/etcd/peer.key --cacert /etc/kubernetes/pki/etcd/ca.crt --endpoints=https://192.168.7.180:2379,https://192.168.7.181:2379,https://192.168.7.182:2379 endpoint health –cluster**

****

****

**把 busybox-1-28.tar.gz 上传到 node1 节点，手动解压**

**[root@node1 ~]# ctr images import busybox-1-28.tar.gz**

**[root@master1 ~]# kubectl run busybox --image docker.io/library/busybox:1.28 --image-pull-policy=IfNotPresent --restart=Never --rm -it busybox – sh**

**/ # ping www.baidu.com**

**PING www.baidu.com (39.156.66.18): 56 data bytes**

**64 bytes from 39.156.66.18: seq=0 ttl=127 time=39.3 ms**

**#通过上面可以看到能访问网络，说明 calico 网络插件已经被正常安装了**