kubeadm 构建多 master 多 node 的 K8S 集群

（Kubernetes 版本：1.28.1　containerd 运行时）

张岩峰老师微信，加我微信，邀请你加入 VIP 交流答疑群：

微信号：ZhangYanFeng0429

二维码：

一、环境规划

实验环境规划：

podSubnet（pod 网段）：10.244.0.0/16

serviceSubnet（service 网段）：10.10.0.0/16

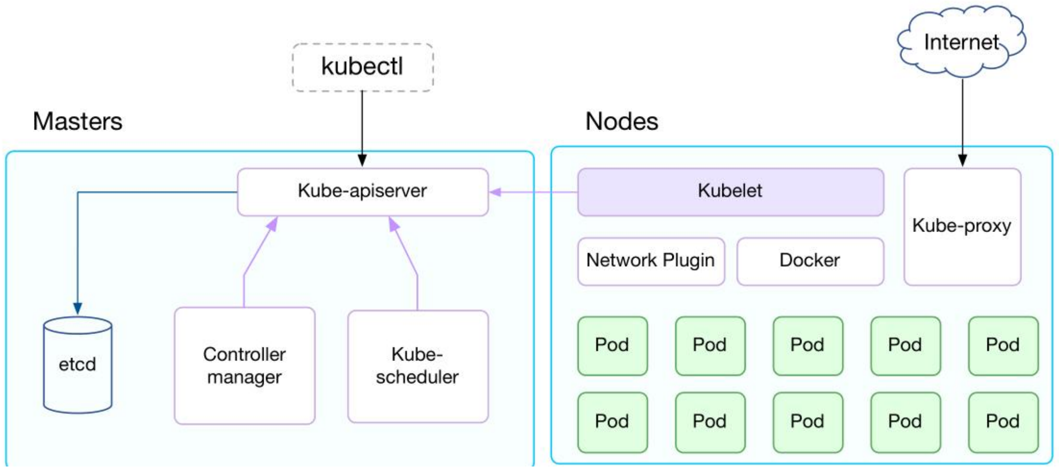
VIP：192.168.128.100

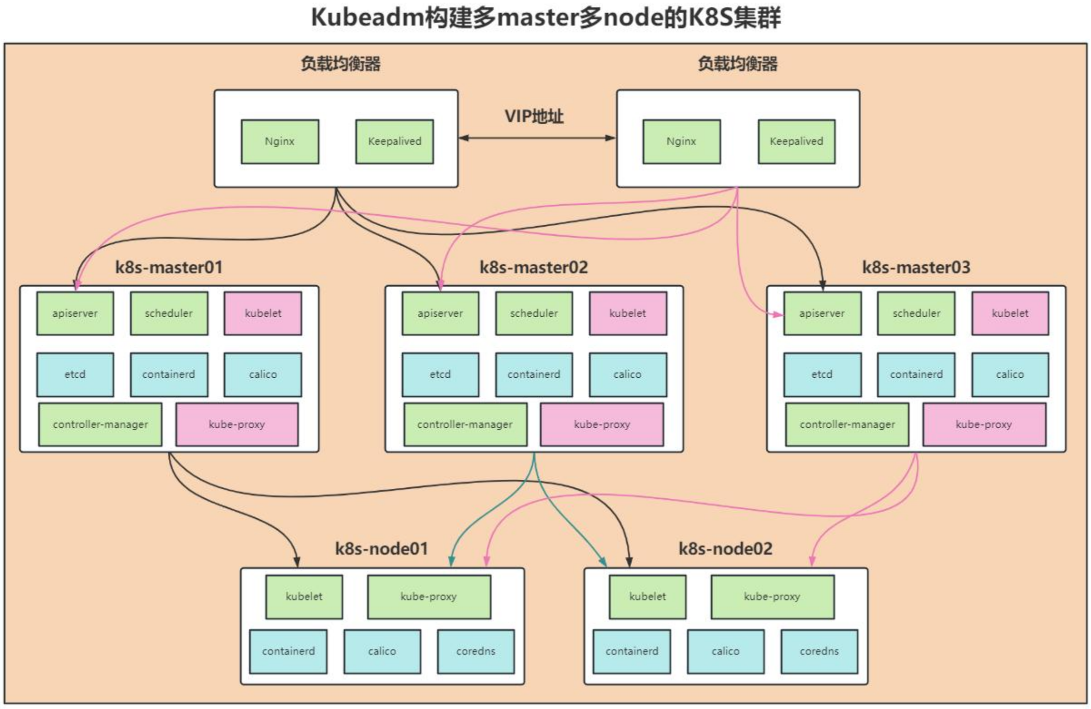
系统版本：Rocky 8.8 操作系统版本

角色	IP	主机名	组件	硬件
控制节点	192.168.128.11	k8s-master01	apiserver controller-manager scheduler etcd containerd keepalived+nginx calico kubelet	CPU：4vCPU 硬盘：100G 内存：4GB 开启虚拟化

			kube-proxy	
控制节点	192.168.128.12	k8s-master02	apiserver controller-manager scheduler etcd containerd keepalived+nginx calico kubelet kube-proxy	CPU：4vCPU 硬盘：100G 内存：4GB 开启虚拟化
控制节点	192.168.128.13	k8s-master03	apiserver controller-manager scheduler etcd containerd calico kubelet kube-proxy	CPU：4vCPU 硬盘：100G 内存：4GB 开启虚拟化
工作节点	192.168.128.21	k8s-node01	kubelet kube-proxy containerd calico coredns	CPU：6vCPU 硬盘：100G 内存：6GB 开启虚拟化
工作节点	192.168.128.22	k8s-node02	kubelet kube-proxy containerd calico coredns	CPU：6vCPU 硬盘：100G 内存：6GB 开启虚拟化

拓扑图：





二、初始化系统环境

1、配置机器主机名

128.11 节点执行：

```
[root@192 ~]# hostnamectl set-hostname k8s-master01 && bash
```

128.12 节点执行：

```
[root@192 ~]# hostnamectl set-hostname k8s-master02 && bash
```

128.13 节点执行：

```
[root@192 ~]# hostnamectl set-hostname k8s-master03 && bash
```

128.21 节点执行：

```
[root@192 ~]# hostnamectl set-hostname k8s-node01 && bash
```

128.22 节点执行：

```
[root@192 ~]# hostnamectl set-hostname k8s-node02 && bash
```

2、配置 hosts 解析

128.11、128.12、128.13、128.21、128.22 节点执行如下：

```
[root@k8s-master01 ~]# vi /etc/hosts
192.168.128.11 k8s-master01
```

```
192.168.128.12 k8s-master02
192.168.128.13 k8s-master03
192.168.128.21 k8s-node01
192.168.128.22 k8s-node02
```

3、配置主机之间无密码登录

128.11、128.12、128.13、128.21、128.22 节点执行如下：

```
[root@k8s-master01 ~]# ssh-keygen
[root@k8s-master01 ~]# ssh-copy-id k8s-master01
[root@k8s-master01 ~]# ssh-copy-id k8s-master02
[root@k8s-master01 ~]# ssh-copy-id k8s-master03
[root@k8s-master01 ~]# ssh-copy-id k8s-node01
```

4、关闭交换分区 swap，提升性能

128.11、128.12、128.13、128.21、128.22 节点执行如下：

```
[root@k8s-master01 ~]# swapoff -a

永久关闭：注释 swap 挂载，给 swap 这行开头加一下注释
[root@k8s-master01 ~]# vi /etc/fstab
#/dev/mapper/centos-swap swap        swap    defaults        0 0
```

5、修改机器内核参数

128.11、128.12、128.13、128.21、128.22 节点执行如下：

```
[root@k8s-master01 ~]# modprobe br_netfilter
[root@k8s-master01 ~]# echo "modprobe br_netfilter" >> /etc/profile
[root@k8s-master01 ~]# cat > /etc/sysctl.d/k8s.conf <<EOF
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
net.ipv4.ip_forward = 1
EOF
[root@k8s-master01 ~]# sysctl -p /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
net.ipv4.ip_forward = 1

一键执行：
modprobe br_netfilter
echo "modprobe br_netfilter" >> /etc/profile
cat > /etc/sysctl.d/k8s.conf <<EOF
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
net.ipv4.ip_forward = 1
```

```
    EOF
    sysctl -p /etc/sysctl.d/k8s.conf
```

6、关闭 firewalld 防火墙

128.11、128.12、128.13、128.21、128.22 节点执行如下：

```
    [root@k8s-master01 ~]# systemctl stop firewalld;systemctl disable
firewalld
```

7、关闭 selinux

128.11、128.12、128.13、128.21、128.22 节点执行如下：

```
    [root@k8s-master01             ~]#             sed             -i
’s/SELINUX=enforcing/SELINUX=disabled/g’ /etc/selinux/config
    [root@k8s-master01 ~]# setenforce 0

    一键执行：
    sed          -i          ’s/SELINUX=enforcing/SELINUX=disabled/g’
/etc/selinux/config
    setenforce 0
```

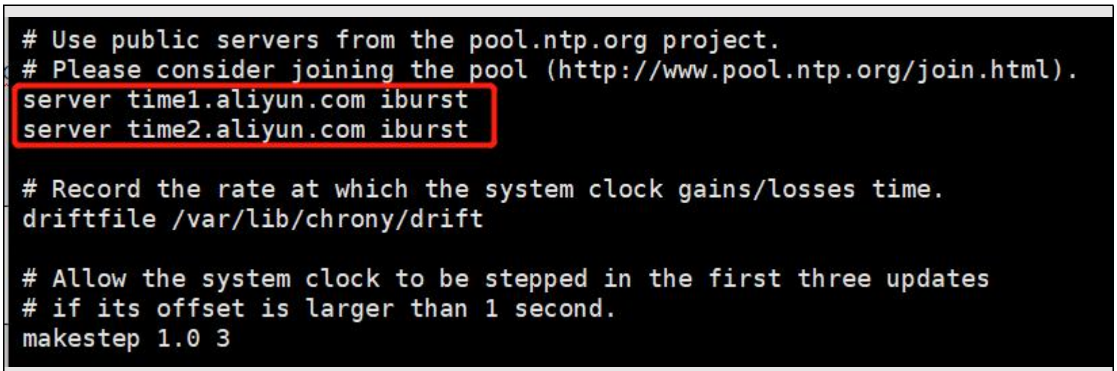
8、配置阿里云 repo 源

128.11、128.12、128.13、128.21、128.22 节点执行如下：

```
    yum -y install wget
    cd /etc/yum.repos.d/
    wget http://mirrors.aliyun.com/repo/Centos-8.repo
    yum clean all
    yum makecache
    yum -y install lrzsz net-tools
    cd
```

9、配置时间同步

```
所有节点执行：
    # 设置时区
    [root@k8s-master01 ~]# timedatectl set-timezone Asia/Shanghai

128.11 节点执行：
    [root@k8s-master01 ~]# yum -y install chrony
    [root@k8s-master01 ~]# vi /etc/chrony.conf
    server time1.aliyun.com iburst
    server time2.aliyun.com iburst
```



```
# Use public servers from the pool.ntp.org project.
# Please consider joining the pool (http://www.pool.ntp.org/join.html).
server time1.aliyun.com iburst
server time2.aliyun.com iburst

# Record the rate at which the system clock gains/losses time.
driftfile /var/lib/chrony/drift

# Allow the system clock to be stepped in the first three updates
# if its offset is larger than 1 second.
makestep 1.0 3
```

```
    [root@k8s-master01 ~]# systemctl restart chronyd && systemctl enable chronyd && systemctl status chronyd

128.12、128.13、128.21、128.22 节点执行：
    [root@k8s-node01 ~]# yum -y install chrony
    [root@k8s-node01 ~]# vi /etc/chrony.conf
    server 192.168.128.11 iburst
    [root@k8s-node01 ~]# systemctl restart chronyd && systemctl enable chronyd && systemctl status chronyd
```

10、开启 ipvs

128.11、128.12、128.13、128.21、128.22 节点执行如下：

```
    [root@k8s-master01 ~]# vi /etc/sysconfig/modules/ipvs.modules
    #!/bin/bash
    ipvs_modules="ip_vs ip_vs_lc ip_vs_wlc ip_vs_rr ip_vs_wrr ip_vs_lblc ip_vs_lblcr ip_vs_dh ip_vs_sh ip_vs_nq ip_vs_sed ip_vs_ftp nf_conntrack"
    for kernel_module in ${ipvs_modules}; do
     /sbin/modinfo -F filename ${kernel_module} > /dev/null 2>&1
     if [ 0 -eq 0 ]; then
     /sbin/modprobe ${kernel_module}
     fi
    done
    [root@k8s-master01 ~]# chmod 755 /etc/sysconfig/modules/ipvs.modules && bash /etc/sysconfig/modules/ipvs.modules && lsmod | grep ip_vs
```

11、安装基础软件包

128.11、128.12、128.13、128.21、128.22 节点执行如下：

```
    [root@k8s-master01 ~]# yum install -y yum-utils device-mapper-persistent-data lvm2 wget net-tools nfs-utils lrzsz gcc gcc-c++ make cmake libxml2-devel openssl-devel curl curl-devel unzip sudo libaio-devel wget vim ncurses-devel autoconf automake zlib-devel
```

```
epel-release openssh-server socat ipvsadm conntrack telnet ipvsadm
```

三、构建 K8S 集群

1、安装 k8s repo 源

128.11、128.12、128.13、128.21 节点执行如下：

```
    # 下面是阿里云的 yum 源
    [root@k8s-master01 ~]# vi /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
    [root@k8s-master01 ~]# yum makecache
    [root@k8s-master01 ~]# yum clean all
```

2、部署 containerd 容器

在 Kubernetes 集群中，containerd 是容器运行时，它的主要作用是负责管理节点上的容器，实现容器的创建、销毁、运行、暂停、恢复等操作。而 Pod 是 Kubernetes 中最基本的调度单元，一个 Pod 包含一个或多个紧密关联的容器，在 Kubernetes 集群中，当一个 Pod 被调度到一个节点上时，Kubernetes 就会基于 containerd 在 pod 里运行容器。

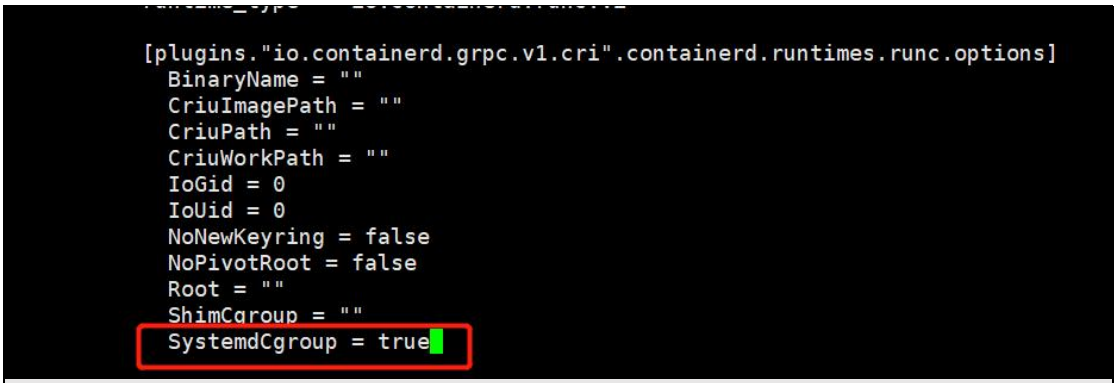
128.11、128.12、128.13、128.21、128.22 节点执行如下：

（1）安装 docker-ce 源：

```
    [root@k8s-master01 ~]# yum-config-manager --add-repo http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo
```

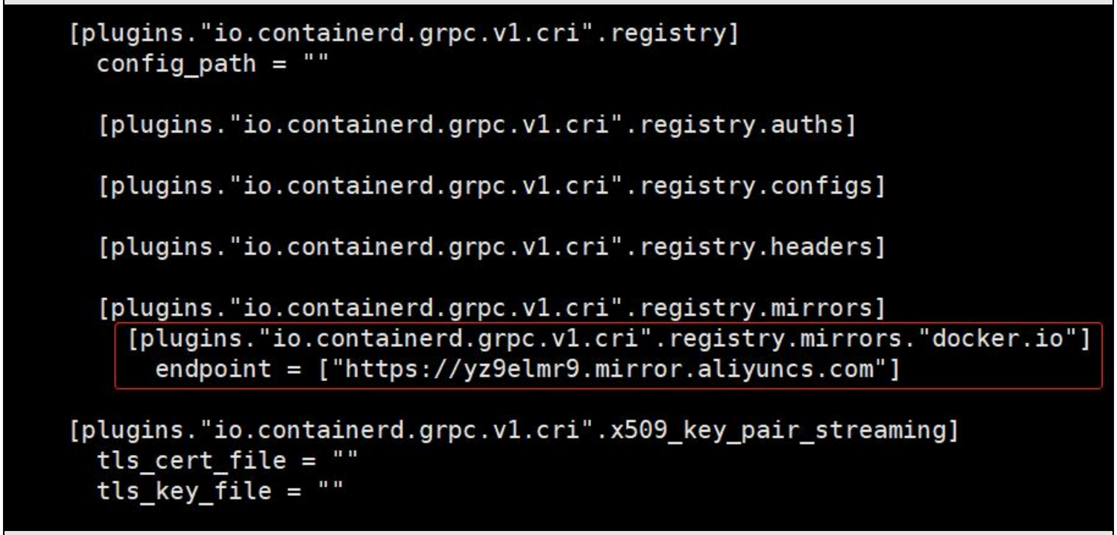
（2）安装、配置、启动 containerd 容器：

```
    # 安装
    [root@k8s-master01 ~]# yum -y install containerd

    # 导出默认的 containerd 配置
    [root@k8s-master01 ~]# containerd config default > /etc/containerd/config.toml

    # 修改 containerd 配置
    [root@k8s-master01 ~]# vi /etc/containerd/config.toml
    # 修改 cgroup Driver 为 systemd
    SystemdCgroup = true
```



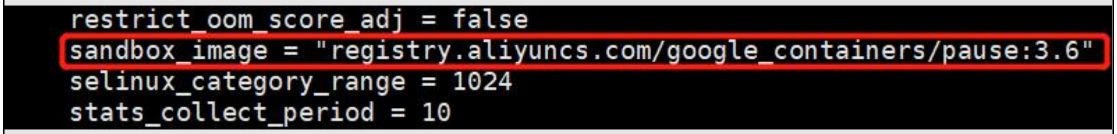
```
        [plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc.options]
          BinaryName = ""
          CriuImagePath = ""
          CriuPath = ""
          CriuWorkPath = ""
          IoGid = 0
          IoUid = 0
          NoNewKeyring = false
          NoPivotRoot = false
          Root = ""
          ShimCgroup = ""
          SystemdCgroup = true
```

```
# 镜像加速，endpoint 位置添加阿里云的镜像源
        [plugins."io.containerd.grpc.v1.cri".registry.mirrors."docker.io"]
          endpoint = ["https://yz9elmr9.mirror.aliyuncs.com"]
```



```
    [plugins."io.containerd.grpc.v1.cri".registry]
      config_path = ""

      [plugins."io.containerd.grpc.v1.cri".registry.auths]

      [plugins."io.containerd.grpc.v1.cri".registry.configs]

      [plugins."io.containerd.grpc.v1.cri".registry.headers]

      [plugins."io.containerd.grpc.v1.cri".registry.mirrors]
        [plugins."io.containerd.grpc.v1.cri".registry.mirrors."docker.io"]
          endpoint = ["https://yz9elmr9.mirror.aliyuncs.com"]

    [plugins."io.containerd.grpc.v1.cri".x509_key_pair_streaming]
      tls_cert_file = ""
      tls_key_file = ""
```

```
# 更改 sandbox_image
"registry.aliyuncs.com/google_containers/pause:3.6"
```



```
    restrict_oom_score_adj = false
    sandbox_image = "registry.aliyuncs.com/google_containers/pause:3.6"
    selinux_category_range = 1024
    stats_collect_period = 10
```

```
# 启动
[root@k8s-node01 ~]# systemctl restart containerd && systemctl enable containerd && systemctl status containerd
```

上述修改的内容解释说明：

SystemdCgroup = true 表示把 containerd 驱动变成 systemd，跟 kubelet 驱动保持一致。

pause 容器：当 Kubernetes 启动一个 Pod 时，会为其创建一个 Pause 容器。Pause 容器是一个极小的 Linux 容器，它不做任何事情，只是为 Pod 中的其他容

器创建一个Linux命名空间和一个网络命名空间，并且共享了一个IPC命名空间，以便其他容器可以与之通信。

3、安装初始化k8s需要的软件包

128.11、128.12、128.13、128.21、128.22节点执行如下：

```
    [root@k8s-master01 ~]# yum install -y kubelet-1.28.1 kubeadm-1.28.1
kubectl-1.28.1
```

提示：每个软件包的作用

- kubelet

kubelet：kubelet是Kubernetes集群中的一个核心组件，是每个节点上的代理服务，负责与主控制节点通信，管理节点上的Pod和容器。

kubelet的主要职责包括：

监控pod的状态并按需启动或停止容器、检查容器是否正常运行、与主控制节点通信，将节点状态和Pod状态上报给主控制节点、管理容器的生命周期，包括启动、停止、重启等、拉取镜像。

- kubeadm

kubeadm：用于初始化、升级k8s集群的命令行工具。

- kubectl

kubectl：用于和集群通信的命令行，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。

4、keepalive+nginx实现k8s apiserver节点高可用

128.11、128.12节点执行如下：

# 安装

```
    [root@k8s-master01 ~]# yum install -y keepalived nginx
nginx-mod-stream
```

# 配置nginx代理，128.11、128.12节点配置都一样

```
    [root@k8s-master01 ~]# vi /etc/nginx/nginx.conf
    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;
    }

    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     192.168.128.11:6443     weight=5     max_fails=3
fail_timeout=30s;
           server     192.168.128.12:6443     weight=5     max_fails=3
fail_timeout=30s;
           server     192.168.128.13:6443     weight=5     max_fails=3
fail_timeout=30s;
        }
        server {
           listen 16443;
           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 / {
            }
        }
    }
```

# 128.11 keepalived 配置：

```
[root@k8s-master01 ~]# vi /etc/keepalived/keepalived.conf
global_defs {
   router_id NGINX_MASTER
}

vrrp_script check_nginx {
    script "/etc/keepalived/check_nginx.sh"
}

vrrp_instance VI_1 {
    state MASTER
    interface ens33
    virtual_router_id 51
    priority 100
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        192.168.128.100/24
    }
    track_script {
        check_nginx
    }
}
```

解释如下：

```
[root@k8s-master01 ~]# vi /etc/keepalived/keepalived.conf
! Configuration File for keepalived

global_defs {
   router_id NGINX_MASTER
}
```

```
vrrp_script check_nginx {
    script "/etc/keepalived/check_nginx.sh"
}

vrrp_instance VI_1 {
    state MASTER
    interface ens33  # 修改为实际网卡名
    virtual_router_id 51  # VRRP 路由 ID 实例，每个实例是唯一的
    priority 100  # 优先级，备服务器设置 90
    advert_int 1  # 指定 VRRP 心跳包通告间隔时间，默认 1 秒
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        192.168.128.100/24
    }
    track_script {
        check_nginx
    }
}

# 128.11 keepalived 故障检测脚本
[root@k8s-master01 ~]# vi /etc/keepalived/check_nginx.sh
#!/bin/bash
count=`ps -C nginx --no-header | wc -l`
if [ ${count} -eq 0 ];then
    systemctl restart nginx
    sleep 2
    counter=`ps -C nginx --no-header | wc -l`
    if [ ${counter} -eq 0 ];then
        systemctl stop keepalived
    fi
fi
[root@k8s-master01 ~]# chmod +x /etc/keepalived/check_nginx.sh
```

# 128.12 keepalived 配置文件：

```
[root@k8s-master02 ~]# vi /etc/keepalived/keepalived.conf
global_defs {
   router_id NGINX_BACKUP
}
```

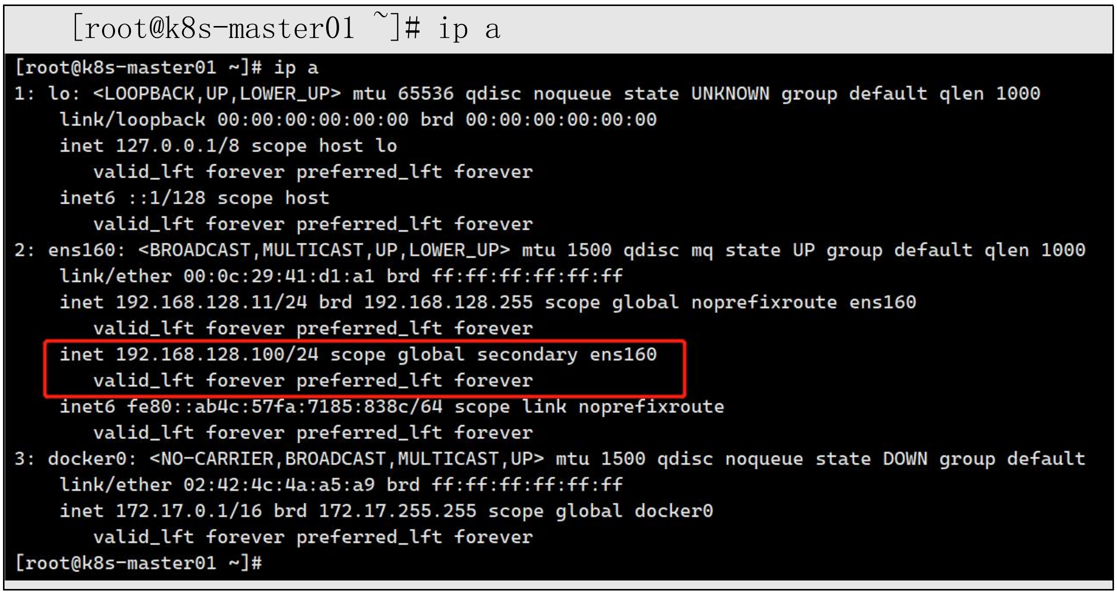
```
vrrp_script check_nginx {
    script "/etc/keepalived/check_nginx.sh"
}

vrrp_instance VI_1 {
    state BACKUP
    interface ens33
    virtual_router_id 51
    priority 90
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        192.168.128.100/24
    }
    track_script {
        check_nginx
    }
}

# 128.12 keepalived 故障检测脚本
[root@k8s-master02 ~]# vi /etc/keepalived/check_nginx.sh
#!/bin/bash
count=`ps -C nginx --no-header | wc -l`
if [ ${count} -eq 0 ];then
    systemctl restart nginx
    sleep 2
    counter=`ps -C nginx --no-header | wc -l`
    if [ ${counter} -eq 0 ];then
        systemctl stop keepalived
    fi
fi
[root@k8s-master02 ~]# chmod +x /etc/keepalived/check_nginx.sh
```

# 128.11、128.12 启动 nginx、keepalived 服务

```
[root@k8s-master01 ~]# systemctl restart nginx && systemctl enable nginx && systemctl status nginx
[root@k8s-master01 ~]# systemctl restart keepalived && systemctl enable keepalived && systemctl status keepalived
```

# 查看 vip 是否生成

```
[root@k8s-master01 ~]# ip a
```



```
[root@k8s-master01 ~]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens160: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 00:0c:29:41:d1:a1 brd ff:ff:ff:ff:ff:ff
    inet 192.168.128.11/24 brd 192.168.128.255 scope global noprefixroute ens160
       valid_lft forever preferred_lft forever
    inet 192.168.128.100/24 scope global secondary ens160
       valid_lft forever preferred_lft forever
    inet6 fe80::ab4c:57fa:7185:838c/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:4c:4a:a5:a9 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
[root@k8s-master01 ~]#
```

5、kubeadm 初始化 k8s 集群

在 k8s-master01 节点使用 kubeadm 初始化 k8s 集群：

1、生成初始化文件（在 128.11 节点执行）

（1）获取默认的初始化参数文件

```
[root@k8s-master01 ~]# kubeadm config print init-defaults > init.default.yaml
```

（2）修改初始化文件

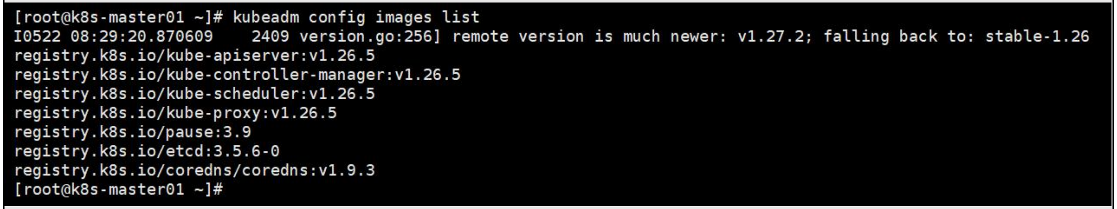
```
[root@k8s-master01 ~]# vi init.default.yaml
apiVersion: kubeadm.k8s.io/v1beta3
bootstrapTokens:
- groups:
  - system:bootstrappers:kubeadm:default-node-token
  token: abcdef.0123456789abcdef
  ttl: 24h0m0s
  usages:
  - signing
  - authentication
kind: InitConfiguration
localAPIEndpoint:
  advertiseAddress: 192.168.128.11
  bindPort: 6443
nodeRegistration:
  criSocket: unix:///run/containerd/containerd.sock
  imagePullPolicy: IfNotPresent
  name: k8s-master01
  taints: null
---
apiServer:
  certSANs:
  - 192.168.128.100
  - 192.168.128.11
  - 192.168.128.12
  - 192.168.128.13
  - 192.168.128.21
  - 192.168.128.22
  - 192.168.128.23
  - 192.168.128.24
  timeoutForControlPlane: 4m0s
apiVersion: kubeadm.k8s.io/v1beta3
certificatesDir: /etc/kubernetes/pki
clusterName: kubernetes
controllerManager: {}
dns: {}
etcd:
  local:
    dataDir: /var/lib/etcd
imageRepository: registry.aliyuncs.com/google_containers
kind: ClusterConfiguration
kubernetesVersion: 1.28.1
controlPlaneEndpoint: 192.168.128.100:16443
networking:
  dnsDomain: cluster.local
  serviceSubnet: 10.10.0.0/16
  podSubnet: 10.244.0.0/16
scheduler: {}
---
apiVersion: kubeproxy.config.k8s.io/v1alpha1
kind: KubeProxyConfiguration
mode: ipvs
---
apiVersion: kubelet.config.k8s.io/v1beta1
kind: KubeletConfiguration
cgroupDriver: systemd
```

初始化文件说明：

```
[root@k8s-master01 ~]# vi init.default.yaml
apiVersion: kubeadm.k8s.io/v1beta3
bootstrapTokens:
- groups:
  - system:bootstrappers:kubeadm:default-node-token
  token: abcdef.0123456789abcdef
  ttl: 24h0m0s
  usages:
  - signing
  - authentication
kind: InitConfiguration
localAPIEndpoint:
  advertiseAddress: 192.168.128.11  #master节点IP地址
  bindPort: 6443  #kube-apiserver组件监听的地址
nodeRegistration:
  criSocket: unix:///run/containerd/containerd.sock  #containerd容器运行时的路径
  imagePullPolicy: IfNotPresent  #镜像拉取策略
  name: k8s-master01  #加入到集群中，显示的名称
  taints: null  #在使用kubeadm初始化Kubernetes集群时，若不指定污点，将默认为Taints值为null，这意味着新生成的所有Node节点都不带任何污点，可以接受所有Pod的调度请求，不会对Pod的调度造成任何限制。
---
apiServer:
  certSANs:  #证书受信任IP地址，尽快多写几个，方便后期扩展Node节点。
  - 192.168.128.100
  - 192.168.128.11
  - 192.168.128.12
  - 192.168.128.13
  - 192.168.128.21
  - 192.168.128.22
  - 192.168.128.23
  - 192.168.128.24
  timeoutForControlPlane: 4m0s
apiVersion: kubeadm.k8s.io/v1beta3
certificatesDir: /etc/kubernetes/pki  #证书生成的位置
clusterName: kubernetes  #集群名称
controllerManager: {}
dns: {}
etcd:
  local:
    dataDir: /var/lib/etcd  #etcd 的数据目录
imageRepository: registry.aliyuncs.com/google_containers  #镜像加速地址
kind: ClusterConfiguration
kubernetesVersion: 1.28.1  #集群版本号
controlPlaneEndpoint: 192.168.128.100:16443   #nginx 负载均衡到 kube-apiserver 的入口地址
networking:
  dnsDomain: cluster.local
  serviceSubnet: 10.10.0.0/16  #service 网段范围
  podSubnet: 10.244.0.0/16  #pod 网段范围
scheduler: {}
# 下面则表示启用 ipvs
---
apiVersion: kubeproxy.config.k8s.io/v1alpha1
kind: KubeProxyConfiguration
mode: ipvs
---
apiVersion: kubelet.config.k8s.io/v1beta1
kind: KubeletConfiguration
cgroupDriver: systemd
```

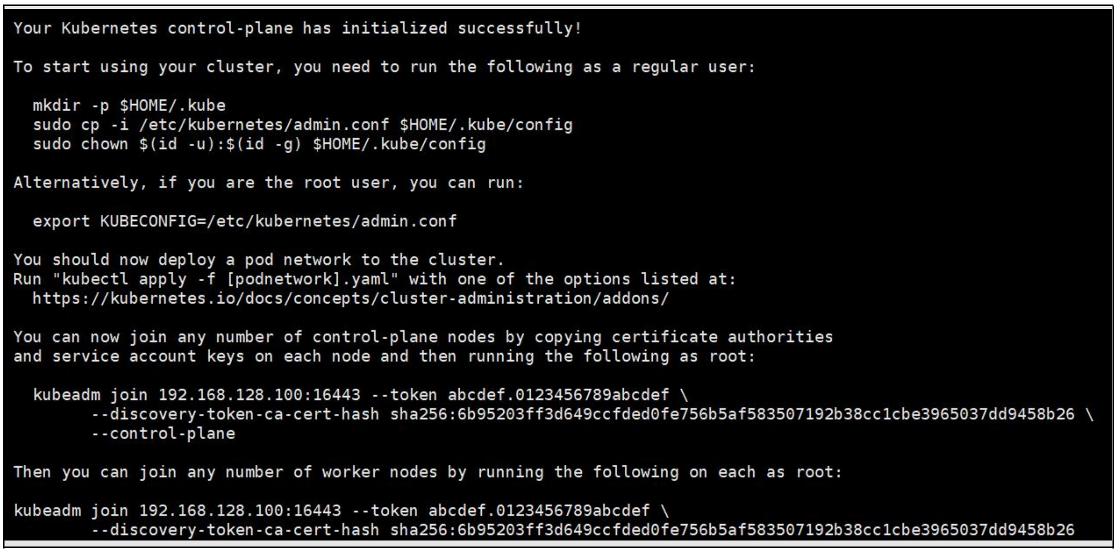
2、初始化集群

```
    # 检查初始化要拉取的镜像有哪些
    [root@k8s-master01 ~]# kubeadm config images list
```



```
[root@k8s-master01 ~]# kubeadm config images list
I0522 08:29:20.870609    2409 version.go:256] remote version is much newer: v1.27.2; falling back to: stable-1.26
registry.k8s.io/kube-apiserver:v1.26.5
registry.k8s.io/kube-controller-manager:v1.26.5
registry.k8s.io/kube-scheduler:v1.26.5
registry.k8s.io/kube-proxy:v1.26.5
registry.k8s.io/pause:3.9
registry.k8s.io/etcd:3.5.6-0
registry.k8s.io/coredns/coredns:v1.9.3
[root@k8s-master01 ~]#
```

```
    # 拉取初始化时需要的镜像（可不执行）
    [root@k8s-master01 ~]# kubeadm config images pull

    # 直接进行初始化，镜像不存在会去拉取
    [root@k8s-master01 ~]# kubeadm init --config=init.default.yaml

    显示如下，表示安装完成：
```



```
Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

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

Alternatively, if you are the root user, you can run:

  export KUBECONFIG=/etc/kubernetes/admin.conf

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
  https://kubernetes.io/docs/concepts/cluster-administration/addons/

You can now join any number of control-plane nodes by copying certificate authorities
and service account keys on each node and then running the following as root:

  kubeadm join 192.168.128.100:16443 --token abcdef.0123456789abcdef \
        --discovery-token-ca-cert-hash sha256:6b95203ff3d649ccfded0fe756b5af583507192b38cc1cbe3965037dd9458b26 \
        --control-plane

Then you can join any number of worker nodes by running the following on each as root:

kubeadm join 192.168.128.100:16443 --token abcdef.0123456789abcdef \
        --discovery-token-ca-cert-hash sha256:6b95203ff3d649ccfded0fe756b5af583507192b38cc1cbe3965037dd9458b26
```

扩展：kubeadm init 初始化流程分析

kubeadm 在执行安装之前进行了相当细致的环境检测，下面看一看：

(1)检查执行 init 命令的用户是否为 root,如果不是 root,直接快速失败(fail fast)。

（2）检查待安装的 k8s 版本是否被当前版本的 kubeadm 支持（kubeadm 版本>=待安装 k8s 版本）。

（3）检查防火墙，如果防火墙未关闭，提示开放端口 10250。

（4）检查端口是否已被占用，6443（或你指定的监听端口）、10257、10259。

（5）检查文件是否已经存在，/etc/kubernetes/manifests/*.yaml。

（6）检查是否存在代理，连接本机网络、服务网络、Pod 网络，都会检查，目前不允许代理。

（7）检查容器运行时，使用 CRI 还是 Docker，如果是 Docker，进一步检查 Docker 服务是否已启动，是否设置了开机自启动。

（8）对于 Linux 系统，会额外检查以下内容：

（8.1）检查以下命令是否存在：crictl、ip、iptables、mount、nsenter、ebtables、ethtool、socat、tc、touch。

（8.2）检查 /proc/sys/net/bridge/bridge-nf-call-iptables、/proc/sys/net/ipv4/ip-forward 内容是否为 1。

（8.3）检查 swap 是否是关闭状态。

（9）检查内核是否被支持，Docker 版本及后端存储 GraphDriver 是否被支持。 对于 Linux 系统，还需检查 OS 版本和 cgroup 支持程度（支持哪些资源的隔离）。

（10）检查主机名访问可达性。

（11）检查 kubelet 版本，要高于 kubeadm 需要的最低版本，同时不高于待安装的 k8s 版本。

（12）检查 kubelet 服务是否开机自启动。

（13）检查 10250 端口是否被占用。

（14）如果开启 IPVS 功能，检查系统内核是否加载了 ipvs 模块。

（15）对于 etcd，如果使用 Local etcd，则检查 2379 端口是否被占用，/var/lib/etcd/

是否为空目录。如果使用 External etcd，则检查证书文件是否存在（CA、key、cert），验证 etcd 服务版本是否符合要求。

（16）如果使用 IPv6，检查/proc/sys/net/bridge/bridge-nf-call-iptables、/proc/sys/net/ipv6/conf/default/forwarding 内容是否为 1。

以上就是 kubeadm init 需要检查的所有项目了！

3、创建 kubectl 授权文件

配置 kubectl 的配置文件 config，相当于对 kubectl 进行授权，这样 kubectl 命令可以使用这个证书对 k8s 集群进行管理。

```
[root@k8s-master01 ~]# mkdir -p $HOME/.kube
[root@k8s-master01 ~]# sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
[root@k8s-master01 ~]# sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

4、查看集群节点

```
[root@k8s-master01 ~]# kubectl get nodes
NAME           STATUS     ROLES           AGE   VERSION
k8s-master01   NotReady   control-plane   55s   v1.28.1
# 此时集群状态还是 NotReady 状态，因为没有安装网络插件。
```

6、扩展 K8S 集群-添加 master 节点

- 扩展 k8s-master02 节点

（1）在 k8s-master02 创建证书存放目录：

```
[root@k8s-master02 ~]# cd /root && mkdir -p /etc/kubernetes/pki/etcd && mkdir -p ~/.kube/
```

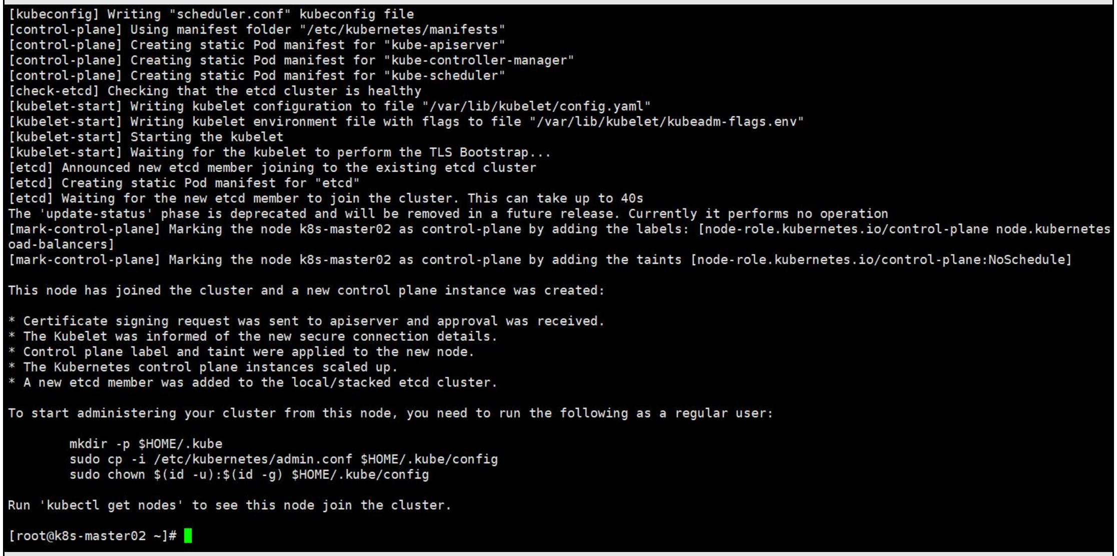
（2）把 k8s-master01 节点的证书拷贝到 k8s-master02 上：

```
在 k8s-master01 节点执行：
scp /etc/kubernetes/pki/ca.crt k8s-master02:/etc/kubernetes/pki/
scp /etc/kubernetes/pki/ca.key k8s-master02:/etc/kubernetes/pki/
scp /etc/kubernetes/pki/sa.key k8s-master02:/etc/kubernetes/pki/
scp /etc/kubernetes/pki/sa.pub k8s-master02:/etc/kubernetes/pki/
scp /etc/kubernetes/pki/front-proxy-ca.crt k8s-master02:/etc/kubernetes/pki/
scp /etc/kubernetes/pki/front-proxy-ca.key k8s-master02:/etc/kubernetes/pki/
scp /etc/kubernetes/pki/etcd/ca.crt k8s-master02:/etc/kubernetes/pki/etcd/
scp /etc/kubernetes/pki/etcd/ca.key k8s-master02:/etc/kubernetes/pki/etcd/
```

（3）加入集群

```
# 在 k8s-master01 上查看加入节点的命令：
[root@k8s-master01 ~]# kubeadm token create --print-join-command
```

```
    kubeadm join 192.168.128.100:16443 --token mykspq.mgcoax02to5cg6xb
--discovery-token-ca-cert-hash
sha256:6b95203ff3d649ccfded0fe756b5af583507192b38cc1cbe3965037dd9458b
26

    # 在 k8s-master02 上加入 k8s-master01 节点：
    [root@k8s-master02 ~]# kubeadm join 192.168.128.100:16443 --token
mykspq.mgcoax02to5cg6xb                --discovery-token-ca-cert-hash
sha256:6b95203ff3d649ccfded0fe756b5af583507192b38cc1cbe3965037dd9458b
26 --control-plane
```



```
[kubeconfig] Writing "scheduler.conf" kubeconfig file
[control-plane] Using manifest folder "/etc/kubernetes/manifests"
[control-plane] Creating static Pod manifest for "kube-apiserver"
[control-plane] Creating static Pod manifest for "kube-controller-manager"
[control-plane] Creating static Pod manifest for "kube-scheduler"
[check-etcd] Checking that the etcd cluster is healthy
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...
[etcd] Announced new etcd member joining to the existing etcd cluster
[etcd] Creating static Pod manifest for "etcd"
[etcd] Waiting for the new etcd member to join the cluster. This can take up to 40s
The 'update-status' phase is deprecated and will be removed in a future release. Currently it performs no operation
[mark-control-plane] Marking the node k8s-master02 as control-plane by adding the labels: [node-role.kubernetes.io/control-plane node.kubernetes
oad-balancers]
[mark-control-plane] Marking the node k8s-master02 as control-plane by adding the taints [node-role.kubernetes.io/control-plane:NoSchedule]

This node has joined the cluster and a new control plane instance was created:

* Certificate signing request was sent to apiserver and approval was received.
* The Kubelet was informed of the new secure connection details.
* Control plane label and taint were applied to the new node.
* The Kubernetes control plane instances scaled up.
* A new etcd member was added to the local/stacked etcd cluster.

To start administering your cluster from this node, you need to run the following as a regular user:

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

Run 'kubectl get nodes' to see this node join the cluster.

[root@k8s-master02 ~]#
```

```
    [root@k8s-master02 ~]# mkdir -p $HOME/.kube
    [root@k8s-master02  ~]#  sudo  cp  -i  /etc/kubernetes/admin.conf
$HOME/.kube/config
    [root@k8s-master02   ~]#   sudo   chown   $(id   -u):$(id   -g)
$HOME/.kube/config

    #  查看集群节点状态
    [root@k8s-master02 ~]# kubectl get nodes
    NAME            STATUS     ROLES           AGE     VERSION
    k8s-master01    NotReady   control-plane   9m48s   v1.28.1
    k8s-master02    NotReady   control-plane   28s     v1.28.1
```

- 扩展 k8s-master03 节点

（1）在 k8s-master03 创建证书存放目录：

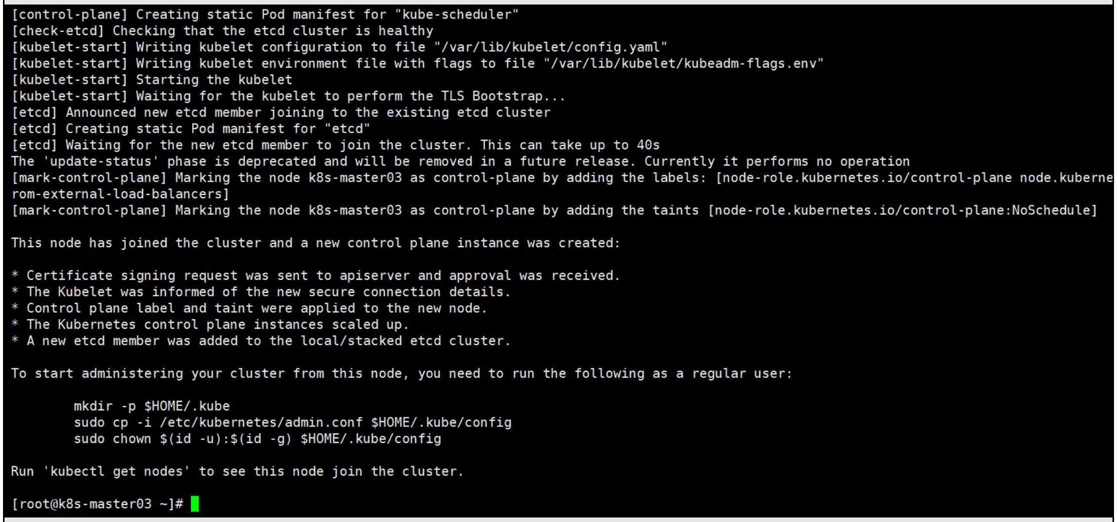
```
    [root@k8s-master03 ~]# cd /root && mkdir -p /etc/kubernetes/pki/etcd
&& mkdir -p ~/.kube/
```

（2）把 k8s-master01 节点的证书拷贝到 k8s-master03 上：

```
    在 k8s-master01 节点执行：
    scp /etc/kubernetes/pki/ca.crt k8s-master03:/etc/kubernetes/pki/
    scp /etc/kubernetes/pki/ca.key k8s-master03:/etc/kubernetes/pki/
    scp /etc/kubernetes/pki/sa.key k8s-master03:/etc/kubernetes/pki/
    scp /etc/kubernetes/pki/sa.pub k8s-master03:/etc/kubernetes/pki/
    scp /etc/kubernetes/pki/front-proxy-ca.crt k8s-master03:/etc/kubernetes/pki/
    scp /etc/kubernetes/pki/front-proxy-ca.key k8s-master03:/etc/kubernetes/pki/
    scp /etc/kubernetes/pki/etcd/ca.crt k8s-master03:/etc/kubernetes/pki/etcd/
    scp /etc/kubernetes/pki/etcd/ca.key k8s-master03:/etc/kubernetes/pki/etcd/
```

（3）加入集群

```
    # 在 k8s-master01 上查看加入节点的命令：
    [root@k8s-master01 ~]# kubeadm token create --print-join-command
    kubeadm join 192.168.128.100:16443 --token mykspq.mgcoax02to5cg6xb
--discovery-token-ca-cert-hash
sha256:6b95203ff3d649ccfded0fe756b5af583507192b38cc1cbe3965037dd9458b
26

    # 在 k8s-master02 上加入 k8s-master01 节点：
    [root@k8s-master02 ~]# kubeadm join 192.168.128.100:16443 --token
mykspq.mgcoax02to5cg6xb                  --discovery-token-ca-cert-hash
sha256:6b95203ff3d649ccfded0fe756b5af583507192b38cc1cbe3965037dd9458b
26 --control-plane
```



```
[control-plane] Creating static Pod manifest for "kube-scheduler"
[check-etcd] Checking that the etcd cluster is healthy
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...
[etcd] Announced new etcd member joining to the existing etcd cluster
[etcd] Creating static Pod manifest for "etcd"
[etcd] Waiting for the new etcd member to join the cluster. This can take up to 40s
The 'update-status' phase is deprecated and will be removed in a future release. Currently it performs no operation
[mark-control-plane] Marking the node k8s-master03 as control-plane by adding the labels: [node-role.kubernetes.io/control-plane node.kuberne
rom-external-load-balancers]
[mark-control-plane] Marking the node k8s-master03 as control-plane by adding the taints [node-role.kubernetes.io/control-plane:NoSchedule]

This node has joined the cluster and a new control plane instance was created:

* Certificate signing request was sent to apiserver and approval was received.
* The Kubelet was informed of the new secure connection details.
* Control plane label and taint were applied to the new node.
* The Kubernetes control plane instances scaled up.
* A new etcd member was added to the local/stacked etcd cluster.

To start administering your cluster from this node, you need to run the following as a regular user:

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

Run 'kubectl get nodes' to see this node join the cluster.

[root@k8s-master03 ~]#
```

```
    [root@k8s-master03 ~]# mkdir -p $HOME/.kube
    [root@k8s-master03  ~]#  sudo  cp  -i  /etc/kubernetes/admin.conf
$HOME/.kube/config
    [root@k8s-master03   ~]#   sudo   chown   $(id   -u):$(id   -g)
```

```
$HOME/.kube/config

    #  查看集群节点状态
    [root@k8s-master03 ~]# kubectl get nodes
    NAME            STATUS     ROLES           AGE     VERSION
    k8s-master01    NotReady   control-plane   12m     v1.28.1
    k8s-master02    NotReady   control-plane   3m21s   v1.28.1
    k8s-master03    NotReady   control-plane   12s     v1.28.1
```

7、扩展 K8S 集群-添加 node 节点

● 扩展 k8s-node01 节点

（1）在任意 master 节点上查看加入节点的命令：

```
    [root@k8s-master03 ~]# kubeadm token create --print-join-command
    kubeadm join 192.168.128.100:16443 --token gg7cyu.qhenxik9emyn70we
--discovery-token-ca-cert-hash
sha256:6b95203ff3d649ccfded0fe756b5af583507192b38cc1cbe3965037dd9458b
26
```

（2）把 k8s-node01 加入 k8s 集群：

```
    [root@k8s-node01 ~]# kubeadm join 192.168.128.100:16443 --token
gg7cyu.qhenxik9emyn70we                --discovery-token-ca-cert-hash
sha256:6b95203ff3d649ccfded0fe756b5af583507192b38cc1cbe3965037dd9458b
26
```

（3）在任意 master 节点上查看集群节点状况：

```
    [root@k8s-master03 ~]# kubectl get nodes
    NAME            STATUS     ROLES           AGE     VERSION
    k8s-master01    NotReady   control-plane   16m     v1.28.1
    k8s-master02    NotReady   control-plane   6m42s   v1.28.1
    k8s-master03    NotReady   control-plane   3m33s   v1.28.1
    k8s-node01      NotReady   <none>          7s      v1.28.1
```

● 扩展 k8s-node02 节点

（1）在任意 master 节点上查看加入节点的命令：

```
    [root@k8s-master03 ~]# kubeadm token create --print-join-command
    kubeadm join 192.168.128.100:16443 --token gg7cyu.qhenxik9emyn70we
--discovery-token-ca-cert-hash
sha256:6b95203ff3d649ccfded0fe756b5af583507192b38cc1cbe3965037dd9458b
26
```

（2）把 k8s-node02 加入 k8s 集群：

```
    [root@k8s-node02 ~]# kubeadm join 192.168.128.100:16443 --token
gg7cyu.qhenxik9emyn70we                     --discovery-token-ca-cert-hash
sha256:6b95203ff3d649ccfded0fe756b5af583507192b38cc1cbe3965037dd9458b
26
```

（3）在任意 master 节点上查看集群节点状况：

```
    [root@k8s-master03 ~]# kubectl get nodes
    NAME            STATUS     ROLES           AGE     VERSION
    k8s-master01    NotReady   control-plane   16m     v1.28.1
    k8s-master02    NotReady   control-plane   7m6s    v1.28.1
    k8s-master03    NotReady   control-plane   3m57s   v1.28.1
    k8s-node01      NotReady   <none>          31s     v1.28.1
    k8s-node02      NotReady   <none>          4s      v1.28.1
```

- 对节点设置角色标签

可以看到 k8s-node01、k8s-node02 的 ROLES 角色为空，可以把 k8s-node01 和 k8s-node02 的 ROLES 变成 work，按照如下方法：

语法格式：

```
语法：
    设置 role：
    kubectl label node [NAME] node-role.kubernetes.io/[ROLES]=

    取消 role：
    kubectl label node [NAME] node-role.kubernetes.io/[ROLES]-

    查询 role：
    kubectl get nodes
```

设置标签：

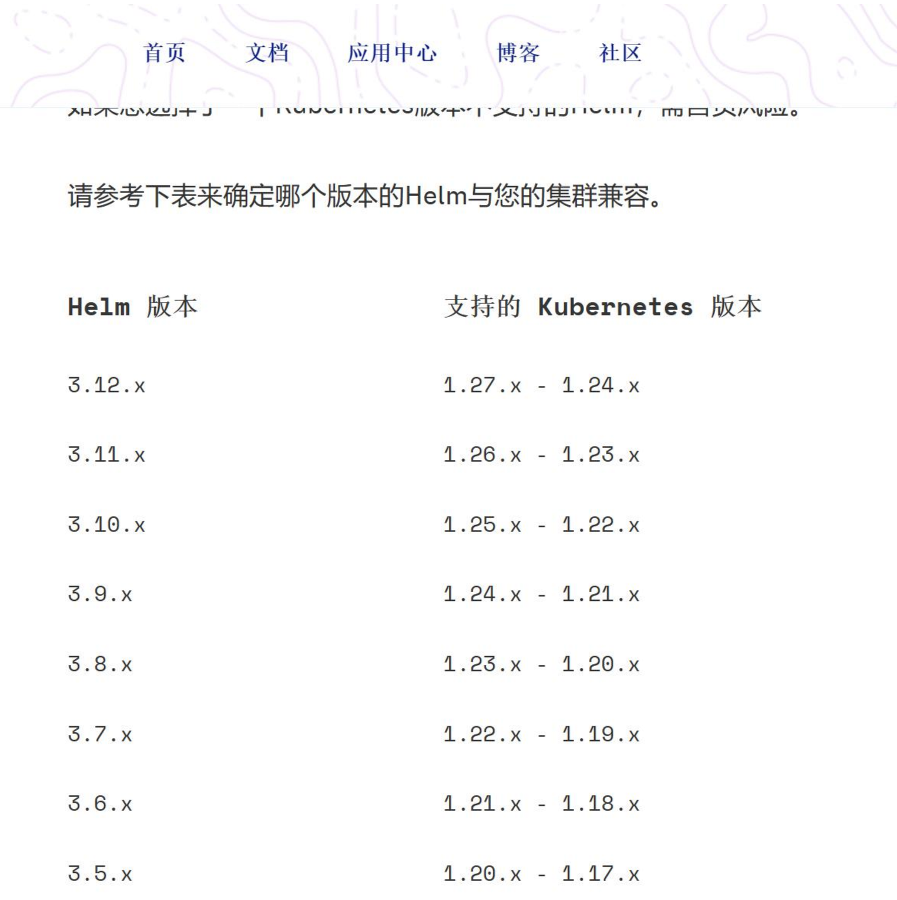
```
    [root@k8s-master01 ~]# kubectl label node k8s-node01
node-role.kubernetes.io/worker=worker
    [root@k8s-master01 ~]# kubectl label node k8s-node02
node-role.kubernetes.io/worker=worker

    [root@k8s-master03 ~]# kubectl get nodes
    NAME            STATUS     ROLES           AGE     VERSION
    k8s-master01    NotReady   control-plane   17m     v1.28.1
    k8s-master02    NotReady   control-plane   7m43s   v1.28.1
    k8s-master03    NotReady   control-plane   4m34s   v1.28.1
    k8s-node01      NotReady   worker          68s     v1.28.1
    k8s-node02      NotReady   worker          41s     v1.28.1
```

8、安装 kubernetes 网络组件-Calico

（1）安装 helm

K8s 版本支持的各个 helm 版本对照表：

官方网址：https://helm.sh/zh/docs/topics/version_skew/



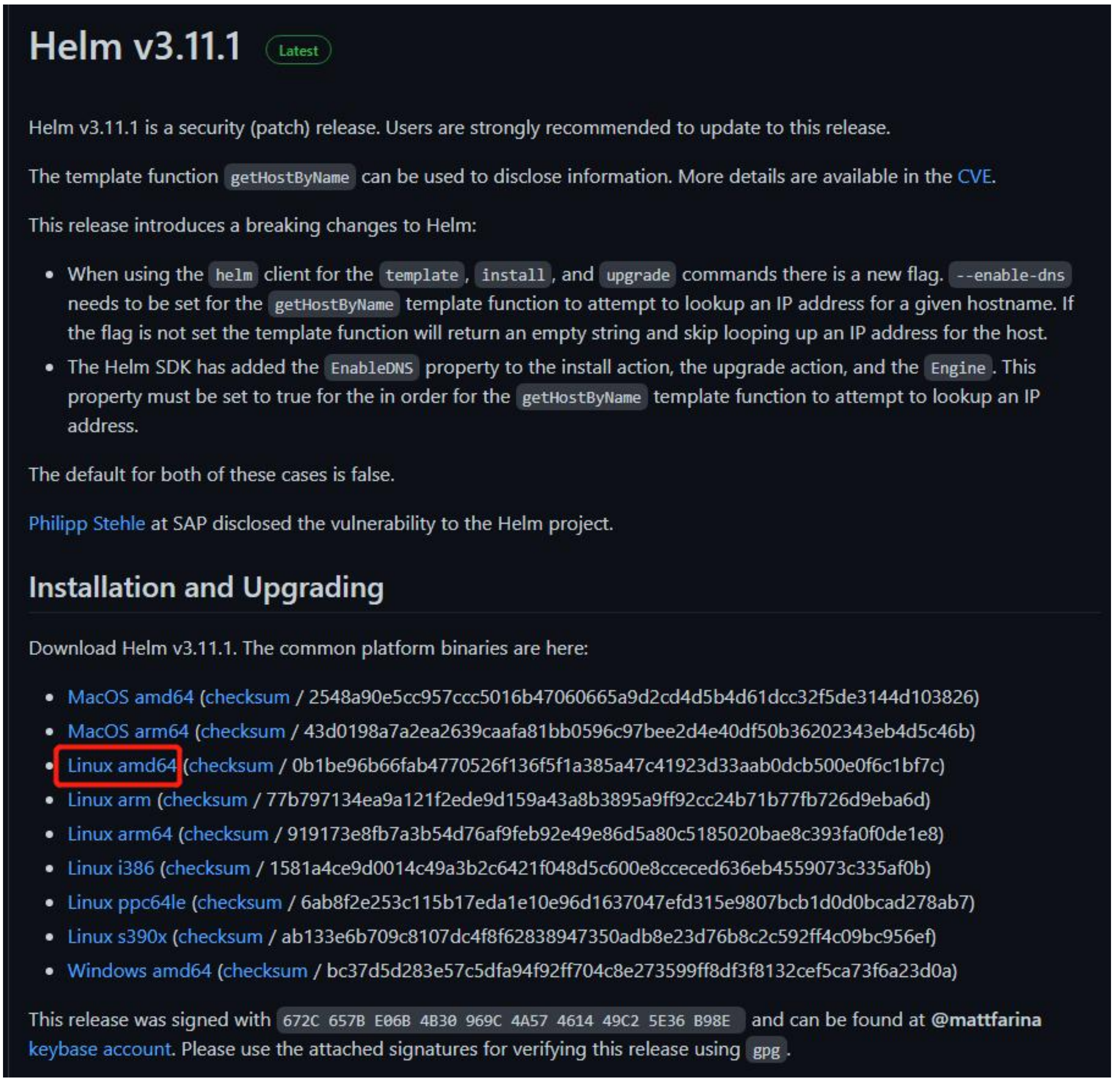
如果您选择了一个Kubernetes版本不支持的Helm，需自负风险。

请参考下表来确定哪个版本的Helm与您的集群兼容。

Helm 版本	支持的 **Kubernetes** 版本
3.12.x	1.27.x - 1.24.x
3.11.x	1.26.x - 1.23.x
3.10.x	1.25.x - 1.22.x
3.9.x	1.24.x - 1.21.x
3.8.x	1.23.x - 1.20.x
3.7.x	1.22.x - 1.19.x
3.6.x	1.21.x - 1.18.x
3.5.x	1.20.x - 1.17.x

下载 helm 软件包：

下载地址：



Helm v3.11.1

Helm v3.11.1 is a security (patch) release. Users are strongly recommended to update to this release.

The template function `getHostByName` can be used to disclose information. More details are available in the CVE.

This release introduces a breaking changes to Helm:

- When using the `helm` client for the `template`, `install`, and `upgrade` commands there is a new flag. `--enable-dns` needs to be set for the `getHostByName` template function to attempt to lookup an IP address for a given hostname. If the flag is not set the template function will return an empty string and skip looping up an IP address for the host.
- The Helm SDK has added the `EnableDNS` property to the install action, the upgrade action, and the `Engine`. This property must be set to true for the in order for the `getHostByName` template function to attempt to lookup an IP address.

The default for both of these cases is false.

Philipp Stehle at SAP disclosed the vulnerability to the Helm project.

Installation and Upgrading

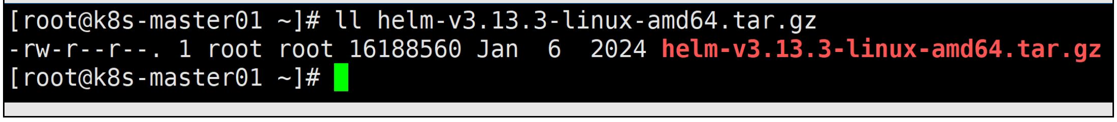
Download Helm v3.11.1. The common platform binaries are here:

- MacOS amd64 (checksum / 2548a90e5cc957ccc5016b47060665a9d2cd4d5b4d61dcc32f5de3144d103826)
- MacOS arm64 (checksum / 43d0198a7a2ea2639caafa81bb0596c97bee2d4e40df50b36202343eb4d5c46b)
- Linux amd64 (checksum / 0b1be96b66fab4770526f136f5f1a385a47c41923d33aab0dcb500e0f6c1bf7c)
- Linux arm (checksum / 77b797134ea9a121f2ede9d159a43a8b3895a9ff92cc24b71b77fb726d9eba6d)
- Linux arm64 (checksum / 919173e8fb7a3b54d76af9feb92e49e86d5a80c5185020bae8c393fa0f0de1e8)
- Linux i386 (checksum / 1581a4ce9d0014c49a3b2c6421f048d5c600e8cceced636eb4559073c335af0b)
- Linux ppc64le (checksum / 6ab8f2e253c115b17eda1e10e96d1637047efd315e9807bcb1d0d0bcad278ab7)
- Linux s390x (checksum / ab133e6b709c8107dc4f8f62838947350adb8e23d76b8c2c592ff4c09bc956ef)
- Windows amd64 (checksum / bc37d5d283e57c5dfa94f92ff704c8e273599ff8df3f8132cef5ca73f6a23d0a)

This release was signed with `672C 657B E06B 4B30 969C 4A57 4614 49C2 5E36 B98E` and can be found at **@mattfarina** keybase account. Please use the attached signatures for verifying this release using `gpg`.

上传软件包到 k8s-master01 节点：

```
[root@k8s-master01 ~]# ll helm-v3.13.3-linux-amd64.tar.gz
```



```
[root@k8s-master01 ~]# ll helm-v3.13.3-linux-amd64.tar.gz
-rw-r--r--. 1 root root 16188560 Jan  6  2024 helm-v3.13.3-linux-amd64.tar.gz
[root@k8s-master01 ~]#
```

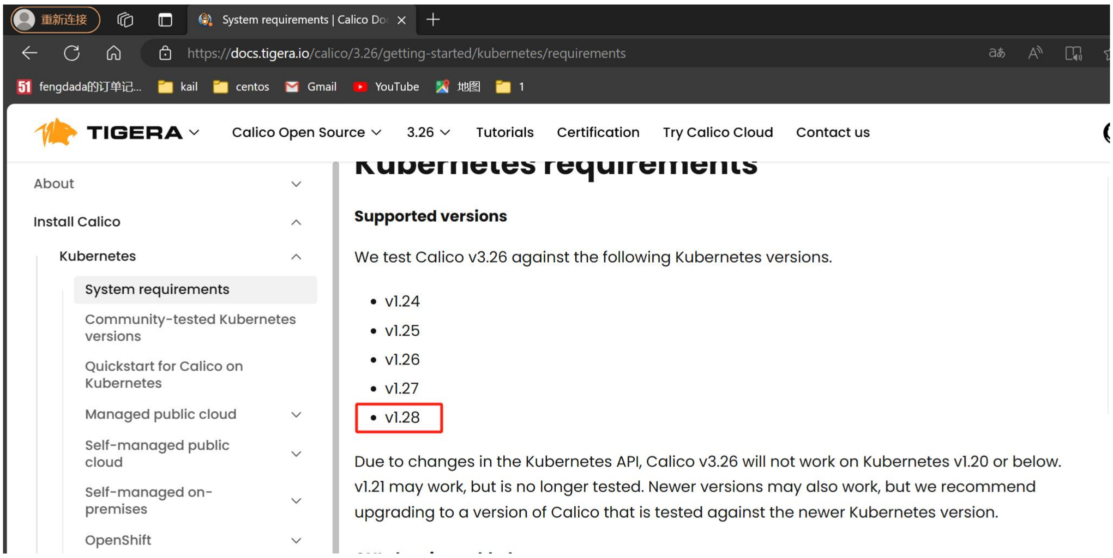
解压、安装：

```
[root@k8s-master01 ~]# tar xf helm-v3.13.3-linux-amd64.tar.gz
[root@k8s-master01 ~]# mv linux-amd64/helm /usr/local/bin/
[root@k8s-master01 ~]# helm version
version.BuildInfo{Version:"v3.13.3", GitCommit:"c8b948945e52abba22ff885446a1486cb5fd3474", GitTreeState:"clean", GoVersion:"go1.20.11"}
```

（2）查看 calico 组件版本对 kubernetes 集群版本的要求

Calico 官网“版本对应关系”：

https://docs.tigera.io/calico/3.25/getting-started/kubernetes/requirements


重新连接
System requirements | Calico Do
https://docs.tigera.io/calico/3.26/getting-started/kubernetes/requirements
fengdada的订单记... kail centos Gmail YouTube 地图 1
TIGERA
Calico Open Source
3.26
Tutorials
Certification
Try Calico Cloud
Contact us
About
Install Calico
Kubernetes
System requirements
Community-tested Kubernetes versions
Quickstart for Calico on Kubernetes
Managed public cloud
Self-managed public cloud
Self-managed on-premises
OpenShift

Kubernetes requirements

Supported versions

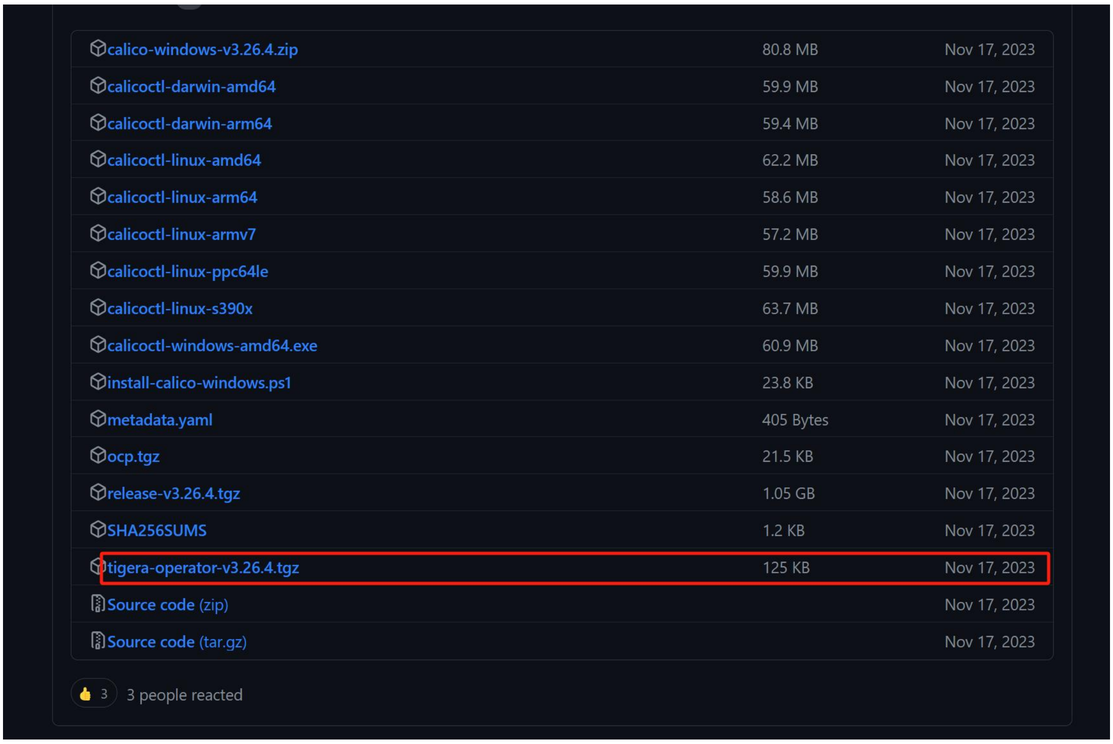
We test Calico v3.26 against the following Kubernetes versions.

- v1.24
- v1.25
- v1.26
- v1.27
- v1.28

Due to changes in the Kubernetes API, Calico v3.26 will not work on Kubernetes v1.20 or below. v1.21 may work, but is no longer tested. Newer versions may also work, but we recommend upgrading to a version of Calico that is tested against the newer Kubernetes version.

（3）安装 calico

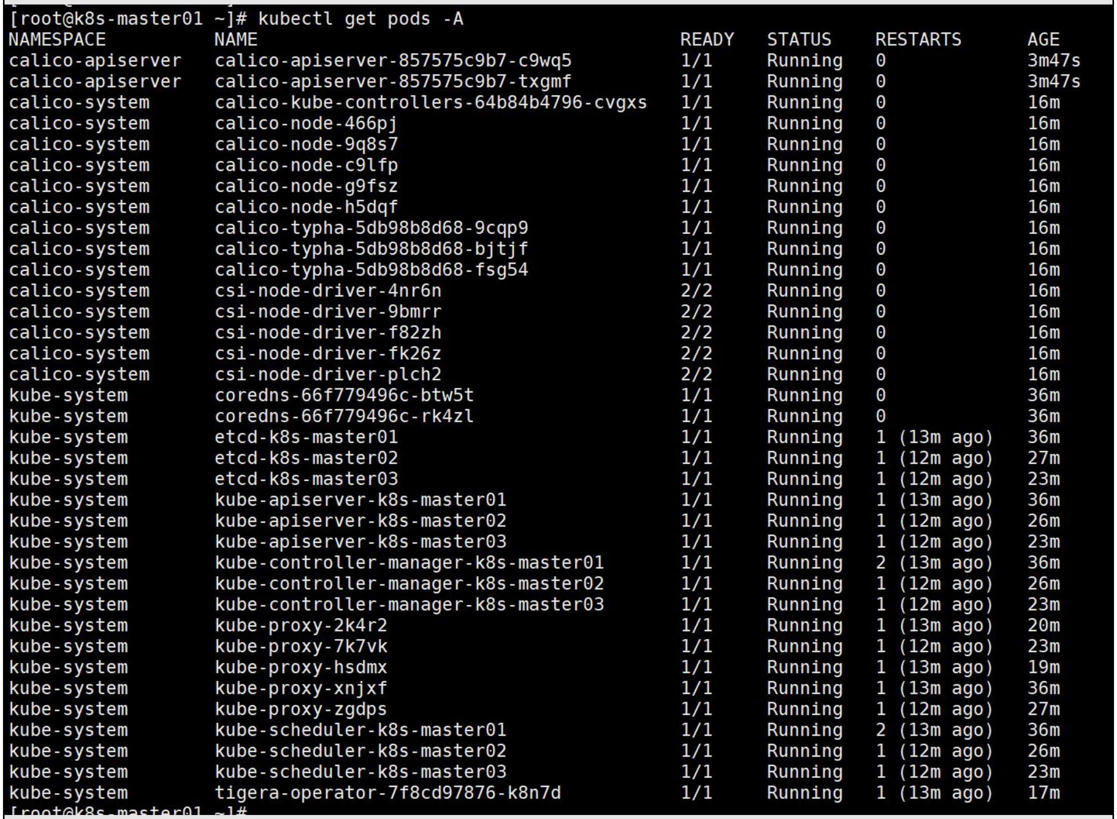
Calico github 下载地址：



calico-windows-v3.26.4.zip	80.8 MB	Nov 17, 2023
calicoctl-darwin-amd64	59.9 MB	Nov 17, 2023
calicoctl-darwin-arm64	59.4 MB	Nov 17, 2023
calicoctl-linux-amd64	62.2 MB	Nov 17, 2023
calicoctl-linux-arm64	58.6 MB	Nov 17, 2023
calicoctl-linux-armv7	57.2 MB	Nov 17, 2023
calicoctl-linux-ppc64le	59.9 MB	Nov 17, 2023
calicoctl-linux-s390x	63.7 MB	Nov 17, 2023
calicoctl-windows-amd64.exe	60.9 MB	Nov 17, 2023
install-calico-windows.ps1	23.8 KB	Nov 17, 2023
metadata.yaml	405 Bytes	Nov 17, 2023
ocp.tgz	21.5 KB	Nov 17, 2023
release-v3.26.4.tgz	1.05 GB	Nov 17, 2023
SHA256SUMS	1.2 KB	Nov 17, 2023
tigera-operator-v3.26.4.tgz	125 KB	Nov 17, 2023
Source code (zip)		Nov 17, 2023
Source code (tar.gz)		Nov 17, 2023

```
[root@k8s-master01 ~]# wget https://github.com/projectcalico/calico/releases/download/v3.26.4/tigera-operator-v3.26.4.tgz
```

```
    # 安装 calico
    [root@k8s-master01    ~]#    helm    install    calico
tigera-operator-v3.26.4.tgz -n kube-system --create-namespace
    NAME: calico
    LAST DEPLOYED: Sat Jan  6 18:11:34 2024
    NAMESPACE: kube-system
    STATUS: deployed
    REVISION: 1
    TEST SUITE: None

    # 检查
    [root@k8s-master01 ~]# kubectl get pods -A
```



```
[root@k8s-master01 ~]# kubectl get pods -A
NAMESPACE          NAME                                       READY   STATUS    RESTARTS       AGE
calico-apiserver   calico-apiserver-857575c9b7-c9wq5          1/1     Running   0              3m47s
calico-apiserver   calico-apiserver-857575c9b7-txgmf          1/1     Running   0              3m47s
calico-system      calico-kube-controllers-64b84b4796-cvgxs   1/1     Running   0              16m
calico-system      calico-node-466pj                          1/1     Running   0              16m
calico-system      calico-node-9q8s7                          1/1     Running   0              16m
calico-system      calico-node-c9lfp                          1/1     Running   0              16m
calico-system      calico-node-g9fsz                          1/1     Running   0              16m
calico-system      calico-node-h5dqf                          1/1     Running   0              16m
calico-system      calico-typha-5db98b8d68-9cqp9              1/1     Running   0              16m
calico-system      calico-typha-5db98b8d68-bjtjf              1/1     Running   0              16m
calico-system      calico-typha-5db98b8d68-fsg54              1/1     Running   0              16m
calico-system      csi-node-driver-4nr6n                      2/2     Running   0              16m
calico-system      csi-node-driver-9bmrr                      2/2     Running   0              16m
calico-system      csi-node-driver-f82zh                      2/2     Running   0              16m
calico-system      csi-node-driver-fk26z                      2/2     Running   0              16m
calico-system      csi-node-driver-plch2                      2/2     Running   0              16m
kube-system        coredns-66f779496c-btw5t                   1/1     Running   0              36m
kube-system        coredns-66f779496c-rk4zl                   1/1     Running   0              36m
kube-system        etcd-k8s-master01                          1/1     Running   1 (13m ago)    36m
kube-system        etcd-k8s-master02                          1/1     Running   1 (12m ago)    27m
kube-system        etcd-k8s-master03                          1/1     Running   1 (12m ago)    23m
kube-system        kube-apiserver-k8s-master01                1/1     Running   1 (13m ago)    36m
kube-system        kube-apiserver-k8s-master02                1/1     Running   1 (12m ago)    26m
kube-system        kube-apiserver-k8s-master03                1/1     Running   1 (12m ago)    23m
kube-system        kube-controller-manager-k8s-master01       1/1     Running   2 (13m ago)    36m
kube-system        kube-controller-manager-k8s-master02       1/1     Running   1 (12m ago)    26m
kube-system        kube-controller-manager-k8s-master03       1/1     Running   1 (12m ago)    23m
kube-system        kube-proxy-2k4r2                           1/1     Running   1 (13m ago)    20m
kube-system        kube-proxy-7k7vk                           1/1     Running   1 (12m ago)    23m
kube-system        kube-proxy-hsdmx                           1/1     Running   1 (13m ago)    19m
kube-system        kube-proxy-xnjxf                           1/1     Running   1 (13m ago)    36m
kube-system        kube-proxy-zgdps                           1/1     Running   1 (12m ago)    27m
kube-system        kube-scheduler-k8s-master01                1/1     Running   2 (13m ago)    36m
kube-system        kube-scheduler-k8s-master02                1/1     Running   1 (12m ago)    26m
kube-system        kube-scheduler-k8s-master03                1/1     Running   1 (12m ago)    23m
kube-system        tigera-operator-7f8cd97876-k8n7d           1/1     Running   1 (13m ago)    17m
[root@k8s-master01 ~]#
```

```
    [root@k8s-master01 ~]# kubectl get nodes
    NAME           STATUS   ROLES           AGE   VERSION
    k8s-master01   Ready    control-plane   36m   v1.28.1
    k8s-master02   Ready    control-plane   27m   v1.28.1
    k8s-master03   Ready    control-plane   24m   v1.28.1
    k8s-node01     Ready    worker          20m   v1.28.1
    k8s-node02     Ready    worker          20m   v1.28.1
```

9、测试在k8s创建pod是否可以正常访问网络

在master执行启动pod：

```
[root@k8s-master01 ~]# kubectl run busybox --image busybox:latest
--restart=Never --rm -it busybox -- sh
If you don't see a command prompt, try pressing enter.

/ # 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=49 time=37.353 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
```

10、测试k8s集群中部署tomcat服务

在任意k8s-master节点执行：

（1）创建pod：

```
[root@k8s-master01 ~]# vi tomcat.yaml
apiVersion: v1
kind: Pod
metadata:
  name: demo-pod
  namespace: default
  labels:
    app: myapp
    env: dev
spec:
  containers:
  - name:  tomcat-pod-java
    ports:
    - containerPort: 8080
    image: tomcat:8.5-jre8-alpine
    imagePullPolicy: IfNotPresent

解释如下：
apiVersion: v1  #pod属于k8s核心组v1
kind: Pod  #创建的是一个Pod资源
metadata:  #元数据
  name: demo-pod  #pod名字
  namespace: default  #pod所属的名称空间
  labels:
    app: myapp  #pod具有的标签
    env: dev      #pod具有的标签
```

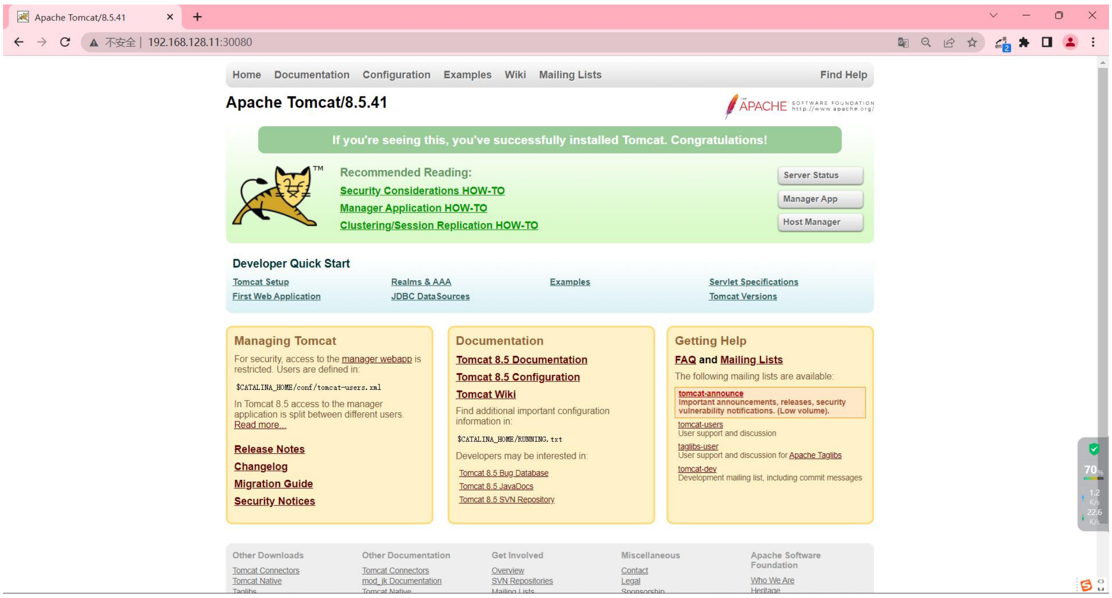
```
spec:
  containers:   #定义一个容器，容器是对象列表，下面可以有多个 name
  - name:  tomcat-pod-java  #容器的名字
    ports:
    - containerPort: 8080
    image: tomcat:8.5-jre8-alpine   #容器使用的镜像
    imagePullPolicy: IfNotPresent

[root@k8s-master01 ~]# kubectl apply -f tomcat.yaml

[root@k8s-master01 ~]# kubectl get pods
NAME        READY    STATUS     RESTARTS    AGE
demo-pod    1/1      Running    0           2m5s
```

（2）创建 svc：

```
[root@k8s-master01 ~]# vi tomcat-svc.yaml
apiVersion: v1
kind: Service
metadata:
  name: tomcat
spec:
  type: NodePort
  ports:
    - port: 8080
      nodePort: 30080
  selector:
    app: myapp
    env: dev
[root@k8s-master01 ~]# kubectl apply -f tomcat-svc.yaml
[root@k8s-master01 ~]# kubectl get svc
NAME      TYPE      CLUSTER-IP     EXTERNAL-IP    PORT(S)           AGE
kubernetes ClusterIP 10.10.0.1     <none>         443/TCP           82m
tomcat     NodePort  10.10.3.208   <none>         8080:30080/TCP    6s
```

（3）测试

访问集群任意地址+30080 端口访问：



Apache Tomcat/8.5.41

If you're seeing this, you've successfully installed Tomcat. Congratulations!

Recommended Reading:
Security Considerations HOW-TO
Manager Application HOW-TO
Clustering/Session Replication HOW-TO

Developer Quick Start
Tomcat Setup
First Web Application
Realms & AAA
JDBC DataSources
Examples
Servlet Specifications
Tomcat Versions

Managing Tomcat
For security, access to the manager webapp is restricted. Users are defined in:
$CATALINA_HOME/conf/tomcat-users.xml
In Tomcat 8.5 access to the manager application is split between different users. Read more...
Release Notes
Changelog
Migration Guide
Security Notices

Documentation
Tomcat 8.5 Documentation
Tomcat 8.5 Configuration
Tomcat Wiki
Find additional important configuration information in:
$CATALINA_HOME/RUNNING.txt
Developers may be interested in:
Tomcat 8.5 Bug Database
Tomcat 8.5 JavaDocs
Tomcat 8.5 SVN Repository

Getting Help
FAQ and Mailing Lists
The following mailing lists are available:
tomcat-announce
Important announcements, releases, security vulnerability notifications. (Low volume).
tomcat-users
User support and discussion
taglibs-user
User support and discussion for Apache Taglibs
tomcat-dev
Development mailing list, including commit messages

Other Downloads
Tomcat Connectors
Tomcat Native
Taglibs

Other Documentation
Tomcat Connectors
mod_jk Documentation
Tomcat Native

Get Involved
Overview
SVN Repositories
Mailing Lists

Miscellaneous
Contact
Legal
Sponsorship

Apache Software Foundation
Who We Are
Heritage

11、测试 coredns 是否正常

```
    [root@k8s-master01  ~]#  kubectl  run  dig  --rm  -it
--image=docker.io/azukiapp/dig /bin/sh
    / # nslookup kubernetes.default.svc.cluster.local
    Server:        10.10.0.10
    Address:   10.10.0.10#53

    Name:  kubernetes.default.svc.cluster.local
    Address: 10.10.0.1

    / # nslookup www.baidu.com
    Server:        10.10.0.10
    Address:   10.10.0.10#53

    Non-authoritative answer:
    www.baidu.com canonical name = www.a.shifen.com.
    Name:  www.a.shifen.com
    Address: 220.181.38.149
    Name:  www.a.shifen.com
    Address: 220.181.38.150
```

10.10.0.10 就是我们 coreDNS 的 clusterIP，说明 coreDNS 配置好了。解析内部 Service 的名称，是通过 coreDNS 去解析的。

注意：busybox 自带的 nslookup 实现的不是很完全，会导致测试 DNS 失败。所以这里使用的是带 nslookup 的 alphine。

12、部署 Docker 服务

Kubernetes 1.24 之后的版本已经不支持 docker 了，但是还要把 docker 安装在 k8s 节点上，主要是为了用 docker build 基于 dockerfile 做镜像，docker 和 containerd 不冲突，不会互相影响。

安装、配置、启动 docker 容器：

128.11、128.12、128.13、128.21、128.22 节点执行如下：

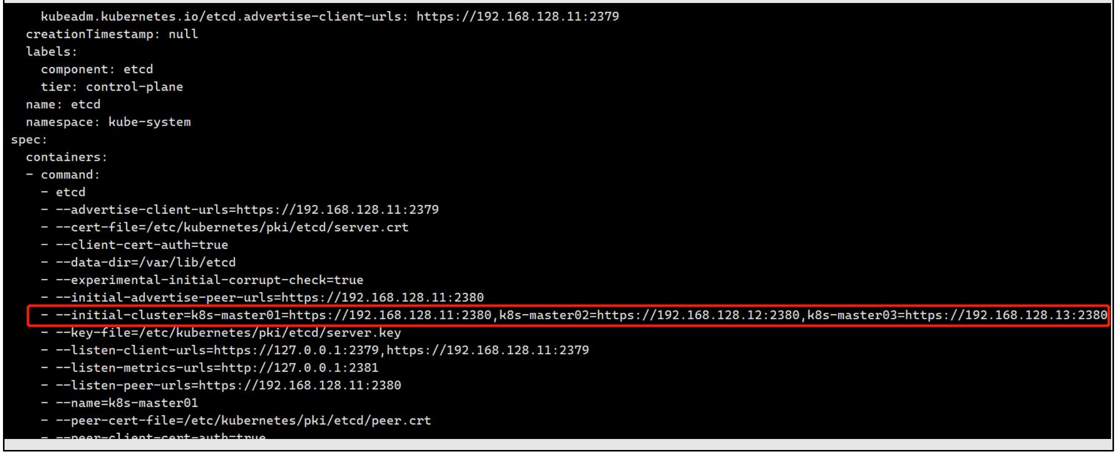
```
    [root@k8s-master01 ~]# yum -y install docker-ce
    [root@k8s-master01 ~]# cd /etc/docker/
    [root@k8s-master01 docker]# vi /etc/docker/daemon.json
    {
      "registry-mirrors": ["https://q2gr04ke.mirror.aliyuncs.com"],
      "exec-opts": ["native.cgroupdriver=systemd"]
    }
    [root@k8s-master01 ~]# systemctl restart docker && systemctl enable
docker && systemctl status docker
```

13、配置 etcd 高可用

操作如下：

（1）修改 k8s-master01、k8s-master02、k8s-master03 上的 etcd.yaml 文件

```
    [root@k8s-master01 ~]# vi /etc/kubernetes/manifests/etcd.yaml
    将：
        - --initial-cluster=k8s-master01=https://192.168.128.11:2380
    修改为：
        -
--initial-cluster=k8s-master01=https://192.168.128.11:2380,k8s-master
02=https://192.168.128.12:2380,k8s-master03=https://192.168.128.13:23
80
```



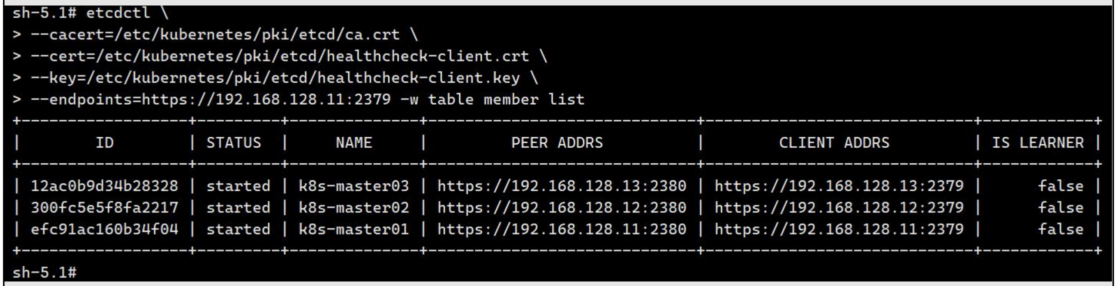
```
    kubeadm.kubernetes.io/etcd.advertise-client-urls: https://192.168.128.11:2379
  creationTimestamp: null
  labels:
    component: etcd
    tier: control-plane
  name: etcd
  namespace: kube-system
spec:
  containers:
  - command:
    - etcd
    - --advertise-client-urls=https://192.168.128.11:2379
    - --cert-file=/etc/kubernetes/pki/etcd/server.crt
    - --client-cert-auth=true
    - --data-dir=/var/lib/etcd
    - --experimental-initial-corrupt-check=true
    - --initial-advertise-peer-urls=https://192.168.128.11:2380
    - --initial-cluster=k8s-master01=https://192.168.128.11:2380,k8s-master02=https://192.168.128.12:2380,k8s-master03=https://192.168.128.13:2380
    - --key-file=/etc/kubernetes/pki/etcd/server.key
    - --listen-client-urls=https://127.0.0.1:2379,https://192.168.128.11:2379
    - --listen-metrics-urls=http://127.0.0.1:2381
    - --listen-peer-urls=https://192.168.128.11:2380
    - --name=k8s-master01
    - --peer-cert-file=/etc/kubernetes/pki/etcd/peer.crt
    - --peer-client-cert-auth=true
```

（2）重启 k8s-master01、k8s-master02、k8s-master03 的 kubelet 服务

```
    [root@k8s-master01 ~]# systemctl restart kubelet
```

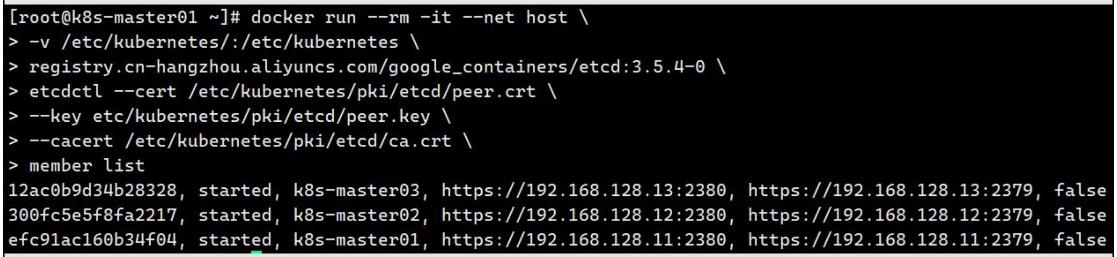
（3）测试 etcd 集群是否配置成功

```
方式一：
    # 进入任意 etcd 节点，进行查询
    [root@k8s-master01  ~]#  kubectl  exec  -it  etcd-k8s-master01  -n
kube-system -- /bin/sh
    sh-5.1# etcdctl \
    --cacert=/etc/kubernetes/pki/etcd/ca.crt \
    --cert=/etc/kubernetes/pki/etcd/healthcheck-client.crt \
    --key=/etc/kubernetes/pki/etcd/healthcheck-client.key \
    --endpoints=https://192.168.128.11:2379 -w table member list
```



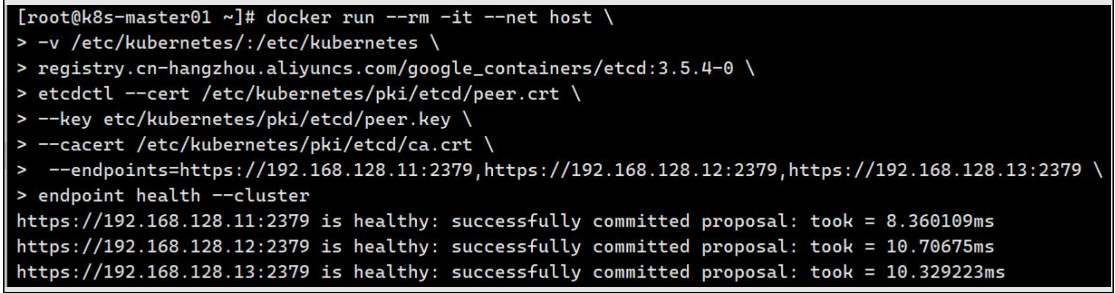
```
sh-5.1# etcdctl \
> --cacert=/etc/kubernetes/pki/etcd/ca.crt \
> --cert=/etc/kubernetes/pki/etcd/healthcheck-client.crt \
> --key=/etc/kubernetes/pki/etcd/healthcheck-client.key \
> --endpoints=https://192.168.128.11:2379 -w table member list
+------------------+---------+--------------+-----------------------------+-----------------------------+------------+
|        ID        | STATUS  |     NAME     |         PEER ADDRS          |        CLIENT ADDRS         | IS LEARNER |
+------------------+---------+--------------+-----------------------------+-----------------------------+------------+
| 12ac0b9d34b28328 | started | k8s-master03 | https://192.168.128.13:2380 | https://192.168.128.13:2379 |      false |
| 300fc5e5f8fa2217 | started | k8s-master02 | https://192.168.128.12:2380 | https://192.168.128.12:2379 |      false |
| efc91ac160b34f04 | started | k8s-master01 | https://192.168.128.11:2380 | https://192.168.128.11:2379 |      false |
+------------------+---------+--------------+-----------------------------+-----------------------------+------------+
sh-5.1#
```

```
方式二：
    # 启动一个临时容器，进行查询
    # 获取 etcd 集群成员列表
    [root@k8s-master01 ~]# 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
```



```
[root@k8s-master01 ~]# 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
12ac0b9d34b28328, started, k8s-master03, https://192.168.128.13:2380, https://192.168.128.13:2379, false
300fc5e5f8fa2217, started, k8s-master02, https://192.168.128.12:2380, https://192.168.128.12:2379, false
efc91ac160b34f04, started, k8s-master01, https://192.168.128.11:2380, https://192.168.128.11:2379, false
```

```
    # 获取 etcd 集群节点运行状态
    [root@k8s-master01 ~]# 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://192.168.128.11:2379,https://192.168.128.12:2379,https://192.168.128.13:2379 \
    endpoint health --cluster
```



```
[root@k8s-master01 ~]# 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://192.168.128.11:2379,https://192.168.128.12:2379,https://192.168.128.13:2379 \
> endpoint health --cluster
https://192.168.128.11:2379 is healthy: successfully committed proposal: took = 8.360109ms
https://192.168.128.12:2379 is healthy: successfully committed proposal: took = 10.70675ms
https://192.168.128.13:2379 is healthy: successfully committed proposal: took = 10.329223ms
```

14、模拟 k8s 故障并快速修复

面试题：

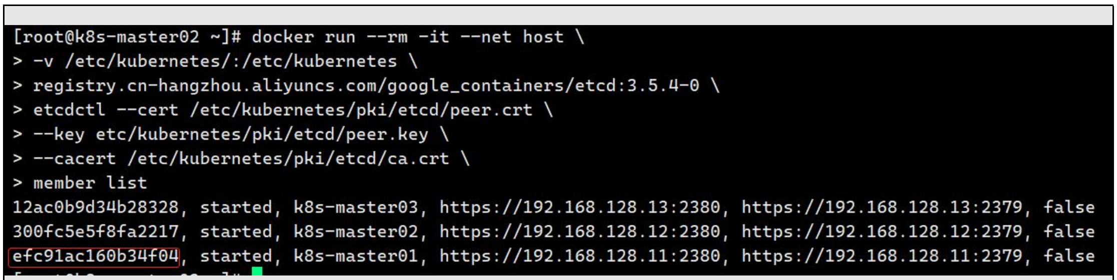
公司里有 3 个控制节点和 1 个工作节点的 Kubernetes 集群，有一个控制节点 k8s-master01 出问题关机了，修复不成功，然后我们 kubectl delete nodes k8s-master01 把 k8s-master01 移除，移除之后，我把机器恢复了，上架了，我打算还这个机器加到 k8s 集群，还是做控制节点，应该如何做？具体说一下实现方式。

移除 k8s-master01 节点

```
    [root@k8s-master02 ~]# kubectl delete nodes k8s-master01
    node "k8s-master01" deleted
```

（1）第一步：把 k8s-master01 这个机器的 etcd 从 etcd 集群删除

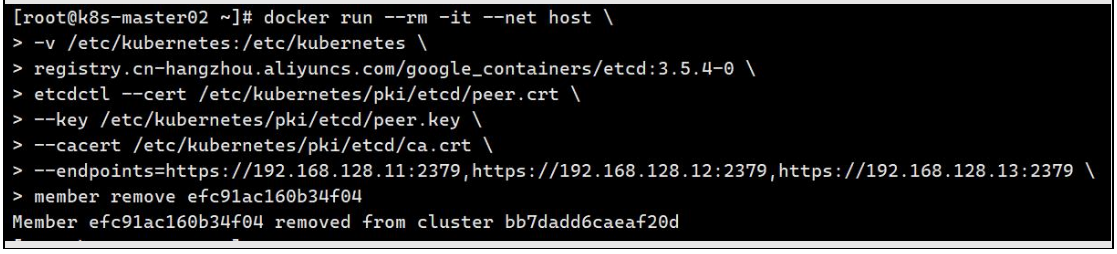
```
    # 查询 k8s-master01 机器 etcd 的 id
    [root@k8s-master02 ~]# docker run --rm -it --net host \
    -v /etc/kubernetes/:/etc/kubernetes \
    registry.cn-hangzhou.aliyuncs.com/google_containers/etcd:3.5.4-0
\
    etcdctl --cert /etc/kubernetes/pki/etcd/peer.crt \
    --key etc/kubernetes/pki/etcd/peer.key \
    --cacert /etc/kubernetes/pki/etcd/ca.crt \
    member list
```



```
[root@k8s-master02 ~]# 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
12ac0b9d34b28328, started, k8s-master03, https://192.168.128.13:2380, https://192.168.128.13:2379, false
300fc5e5f8fa2217, started, k8s-master02, https://192.168.128.12:2380, https://192.168.128.12:2379, false
efc91ac160b34f04, started, k8s-master01, https://192.168.128.11:2380, https://192.168.128.11:2379, false
```

通过上面结果可以看到 k8s-master01 这个机器的 etcd 的 id 是：efc91ac160b34f04

```
# 删除 k8s-master01 的 etcd
[root@k8s-master02 ~]# 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://192.168.128.11:2379,https://192.168.128.12:2379,https://192.168.128.13:2379 \
member remove efc91ac160b34f04
```



```
[root@k8s-master02 ~]# 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://192.168.128.11:2379,https://192.168.128.12:2379,https://192.168.128.13:2379 \
> member remove efc91ac160b34f04
Member efc91ac160b34f04 removed from cluster bb7dadd6caeaf20d
```

（2）第二步：在 k8s-master01 节点上，创建存放证书目录

# 这里我直接使用“kubeadm reset”初始化一下 k8s-master01 节点。如果是新安装的机器，正常部署基础环境即可。

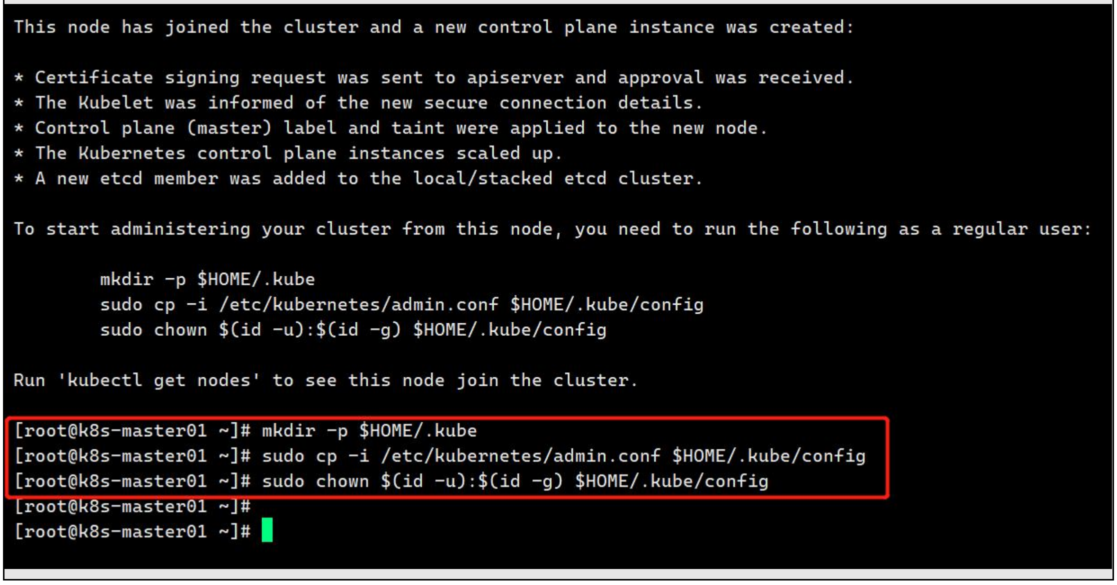
```
[root@k8s-master01 ~]# kubeadm reset

# 创建证书存放目录
[root@k8s-master01 ~]# cd /root && mkdir -p /etc/kubernetes/pki/etcd && mkdir -p ~/.kube/
```

（3）第三步：将任意 k8s-master 节点证书推送到 k8s-master01 节点

```
scp /etc/kubernetes/pki/ca.crt k8s-master01:/etc/kubernetes/pki/
scp /etc/kubernetes/pki/ca.key k8s-master01:/etc/kubernetes/pki/
scp /etc/kubernetes/pki/sa.key k8s-master01:/etc/kubernetes/pki/
scp /etc/kubernetes/pki/sa.pub k8s-master01:/etc/kubernetes/pki/
scp /etc/kubernetes/pki/front-proxy-ca.crt k8s-master01:/etc/kubernetes/pki/
scp /etc/kubernetes/pki/front-proxy-ca.key k8s-master01:/etc/kubernetes/pki/
scp /etc/kubernetes/pki/etcd/ca.crt k8s-master01:/etc/kubernetes/pki/etcd/
scp /etc/kubernetes/pki/etcd/ca.key k8s-master01:/etc/kubernetes/pki/etcd/
```

（4）第四步：将 k8s-master01 加入到集群

```
# 在 k8s-master02 上查看加入节点的命令
[root@k8s-master02 ~]# kubeadm token create --print-join-command
kubeadm join 192.168.128.100:16443 --token zivp6i.ixagl2qhgfhmdin2 --discovery-token-ca-cert-hash sha256:5250c56776060cc67f49326d0a1d5b07bd37f530a47daf97410d06ac78fe5eb2

# 在 k8s-master01 节点执行，加入集群
[root@k8s-master01 ~]# kubeadm join 192.168.128.100:16443 --token zivp6i.ixagl2qhgfhmdin2 --discovery-token-ca-cert-hash sha256:5250c56776060cc67f49326d0a1d5b07bd37f530a47daf97410d06ac78fe5eb2 --control-plane
```



```
This node has joined the cluster and a new control plane instance was created:

* Certificate signing request was sent to apiserver and approval was received.
* The Kubelet was informed of the new secure connection details.
* Control plane (master) label and taint were applied to the new node.
* The Kubernetes control plane instances scaled up.
* A new etcd member was added to the local/stacked etcd cluster.

To start administering your cluster from this node, you need to run the following as a regular user:

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

Run 'kubectl get nodes' to see this node join the cluster.

[root@k8s-master01 ~]# mkdir -p $HOME/.kube
[root@k8s-master01 ~]# sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
[root@k8s-master01 ~]# sudo chown $(id -u):$(id -g) $HOME/.kube/config
[root@k8s-master01 ~]#
[root@k8s-master01 ~]#
```

（5）第五步：验证是否成功加入集群

```
[root@k8s-master01 ~]# kubectl get nodes
NAME           STATUS   ROLES           AGE    VERSION
k8s-master01   Ready    control-plane   30s    v1.26.5
k8s-master02   Ready    control-plane   126m   v1.26.5
k8s-master03   Ready    control-plane   122m   v1.26.5
k8s-node01     Ready    worker          120m   v1.26.5
k8s-node02     Ready    worker          119m   v1.26.5
```

说明：etcd 会重新加入到 etcd 集群，无需重新配置 etcd 高可用。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。