二进制安装k8s 1.20

一、集群环境说明

主机名	IP地址	说明
k8s-master01	10.4.7.107	master节点
k8s-master02	10.4.7.108	master节点
k8s-master03	10.4.7.109	master节点
k8s-master-lb（在master节点）	10.4.7.236	keepalived虚拟IP
k8s-node01	10.4.7.110	worker节点
k8s-node02	10.4.7.111	worker节点

配置信息	备注
系统版本	CentOS 7.9
Docker版本	19.03.x
Pod网段	172.168.0.0/12
Service网段	10.96.0.0/12

注意：

```
VIP（虚拟IP）不要和公司内网IP重复，首先去ping一下，不通才可用。VIP需要和主机在同一个局域网内！
```

二、基础环境配置（以下操作所有节点都得执行）

2.1、配置hosts解析

```
hostnamectl set-hostname k8s-master01

# 查看
hostname
```

```
cat >> /etc/hosts << EFO
10.4.7.107 k8s-master01
10.4.7.108 k8s-master02
10.4.7.109 k8s-master03
10.4.7.236 k8s-master-lb
10.4.7.110 k8s-node01
10.4.7.111 k8s-node02
EFO
```

说明：10.4.7.236 k8s-master-lb # 如果不是高可用集群，该IP为Master01的IP

2.2、更换yum源码

```
curl -o /etc/yum.repos.d/CentOS-Base.repo https://mirrors.aliyun.com/repo/Centos-7.repo

yum install -y yum-utils device-mapper-persistent-data lvm2

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

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

sed -i -e '/mirrors.cloud.aliyuncs.com/d' -e '/mirrors.aliyuncs.com/d' /etc/yum.repos.d/CentOS-Base.repo
```

2.3、安装常用工具

```
yum install wget jq psmisc vim net-tools telnet yum-utils device-mapper-persistent-data lvm2 git lrzsz -y
```

2.4、关闭防火墙、selinux、dnsmasq、swap

```
systemctl disable --now firewalld
systemctl disable --now dnsmasq
systemctl disable --now NetworkManager

setenforce 0
sed -i 's#SELINUX=enforcing#SELINUX=disabled#g' /etc/sysconfig/selinux
sed -i 's#SELINUX=enforcing#SELINUX=disabled#g' /etc/selinux/config

# 关闭swap分区
swapoff -a && sysctl -w vm.swappiness=0
sed -ri '/^[^#]*swap/s@^@#@' /etc/fstab
```

2.5、时间同步配置

```
# 安装ntpdate
rpm -ivh http://mirrors.wlnmp.com/centos/wlnmp-release-centos.noarch.rpm
yum install ntpdate -y

# 更改时区
ln -sf /usr/share/zoneinfo/Asia/Shanghai /etc/localtime

# 设置定时任务同步时间
echo 'Asia/Shanghai' >/etc/timezone
ntpdate time2.aliyun.com

# 加入到crontab
crontab -e
*/5 * * * * ntpdate time2.aliyun.com
```

2.6、优化Linux

```
ulimit -SHn 65535

vim /etc/security/limits.conf
# 末尾添加如下内容
* soft nofile 655360
* hard nofile 131072
* soft nproc 655350
* hard nproc 655350
* soft memlock unlimited
* hard memlock unlimited

# 然后重启Linux
reboot
```

不做，2.7、所有节点升级系统并重启，此处升级没有升级内核，下节会单独升级内核：

```
# CentOS7需要升级，CentOS8可以按需升级系统
yum update -y --exclude=kernel* && reboot
```

三、内核升级

3.1、配置免密登录（Master01上）

Master01节点免密钥登录其他节点，安装过程中生成配置文件和证书均在Master01上操作，集群管理也在Master01上操作，阿里云或者AWS上需要单独一台kubectl服务器。密钥配置如下：

```
# 一直回车就行
ssh-keygen -t rsa

for i in k8s-master01 k8s-master02 k8s-master03 k8s-node01 k8s-node02;do ssh-copy-id -i .ssh/id_rsa.pub $i;done
```

3.2、下载安装所有的源码文件：（Master01上）

```
cd /root/ ; git clone https://github.com/dotbalo/k8s-ha-install.git

cd /root/; git clone git@git.zhlh6.cn:dotbalo/k8s-ha-install.git
```

3.3、下载升级所需安装包（Master01上）

CentOS7 需要升级内核至4.18+，本地升级的版本为4.19

```
# 在master01节点下载内核
cd /root
wget http://193.49.22.109/elrepo/kernel/el7/x86_64/RPMS/kernel-ml-devel-4.19.12-1.el7.elrepo.x86_64.rpm
wget http://193.49.22.109/elrepo/kernel/el7/x86_64/RPMS/kernel-ml-4.19.12-1.el7.elrepo.x86_64.rpm
# 从master01节点传到其他节点：
for i in k8s-master02 k8s-master03 k8s-node01 k8s-node02;do scp kernel-ml-4.19.12-1.el7.elrepo.x86_64.rpm kernel-ml-devel-4.19.12-1.el7.elrepo.x86_64.rpm $i:/root/ ; done
```

3.4、内核升级（所有节点）

```
# 安装内核
cd /root && yum localinstall -y kernel-ml*
grub2-set-default  0 && grub2-mkconfig -o /etc/grub2.cfg
grubby --args="user_namespace.enable=1" --update-kernel="$(grubby --default-kernel)"

# 检查默认内核是不是4.19
grubby --default-kernel /boot/vmlinuz-4.19.12-1.el7.elrepo.x86_64

# 所有节点重启，然后检查内核是不是4.19
reboot
[root@k8s-node02 ~]# uname -a
Linux k8s-node02 4.19.12-1.el7.elrepo.x86_64 #1 SMP Fri Dec 21 11:06:36 EST 2018 x86_64
x86_64 x86_64 GNU/Linux
```

3.5、安装ipvsadm（所有节点）

```
yum install ipvsadm ipset sysstat conntrack libseccomp -y
```

所有节点配置ipvs模块，在内核4.19+版本nf_conntrack_ipv4已经改为nf_conntrack， 4.18以下使用nf_conntrack_ipv4即可：

```
# 加入以下内容
cat > /etc/modules-load.d/ipvs.conf << EFO
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_fo
ip_vs_nq
ip_vs_sed
ip_vs_ftp
ip_vs_sh
nf_conntrack
ip_tables
ip_set
xt_set
ipt_set
ipt_rpfilter
ipt_REJECT
ipip
EFO

# 然后执行
```

```
systemctl enable --now systemd-modules-load.service
```

3.6、开启一些k8s集群中必须的内核参数，配置k8s内核（所有节点）

```
cat <<EOF > /etc/sysctl.d/k8s.conf
net.ipv4.ip_forward = 1
net.bridge.bridge-nf-call-iptables = 1
net.bridge.bridge-nf-call-ip6tables = 1
fs.may_detach_mounts = 1
vm.overcommit_memory=1
vm.panic_on_oom=0
fs.inotify.max_user_watches=89100
fs.file-max=52706963
fs.nr_open=52706963
net.netfilter.nf_conntrack_max=2310720
net.ipv4.tcp_keepalive_time = 600
net.ipv4.tcp_keepalive_probes = 3
net.ipv4.tcp_keepalive_intvl =15
net.ipv4.tcp_max_tw_buckets = 36000
net.ipv4.tcp_tw_reuse = 1
net.ipv4.tcp_max_orphans = 327680
net.ipv4.tcp_orphan_retries = 3
net.ipv4.tcp_syncookies = 1
net.ipv4.tcp_max_syn_backlog = 16384
net.ipv4.ip_conntrack_max = 65536
net.ipv4.tcp_max_syn_backlog = 16384
net.ipv4.tcp_timestamps = 0
net.core.somaxconn = 16384
EOF

# 所有节点配置完内核后，重启服务器，保证重启后内核依旧加载
reboot
[root@k8s-master01 ~]# lsmod | grep --color=auto -e ip_vs -e nf_conntrack
ip_vs_ftp              16384  0
nf_nat                 32768  1 ip_vs_ftp
ip_vs_sed              16384  0
ip_vs_nq               16384  0
ip_vs_fo               16384  0
ip_vs_sh               16384  0
ip_vs_dh               16384  0
ip_vs_lblcr            16384  0
ip_vs_lblc             16384  0
ip_vs_wrr              16384  0
ip_vs_rr               16384  0
ip_vs_wlc              16384  0
ip_vs_lc               16384  0
ip_vs                 151552  24
ip_vs_wlc,ip_vs_rr,ip_vs_dh,ip_vs_lblcr,ip_vs_sh,ip_vs_fo,ip_vs_nq,ip_vs_lblc,ip_vs_wrr
,ip_vs_lc,ip_vs_sed,ip_vs_ftp
```

```
nf_conntrack          143360  2 nf_nat,ip_vs
nf_defrag_ipv6         20480  1 nf_conntrack
nf_defrag_ipv4         16384  1 nf_conntrack
libcrc32c              16384  4 nf_conntrack,nf_nat,xfs,ip_vs
```

四、Docker安装

4.1、安装Docker-ce 19.03（所有节点）

```
yum install docker-ce-19.03.* -y
```

4.1.1温馨提示：

由于新版kubelet建议使用systemd，所以可以把docker的CgroupDriver改成systemd

```
mkdir /etc/docker

cat > /etc/docker/daemon.json <<EOF
{
  "exec-opts": ["native.cgroupdriver=systemd"]
}
EOF
```

4.1.2、所有节点设置开机自启动Docker

```
systemctl daemon-reload && systemctl enable --now docker
```

五、K8s及etcd安装

5.1、下载kubernetes安装包（Master01上）

```
# 注意目前版本是1.20.0，你们安装时需要下载最新的1.20.x版本：
# https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/
[root@k8s-master01 ~]# wget https://dl.k8s.io/v1.20.0/kubernetes-server-linux-amd64.tar.gz
```

5.2、下载etcd安装包（Master01上）

```
[root@k8s-master01 ~]# wget https://github.com/etcd-io/etcd/releases/download/v3.4.13/etcd-v3.4.13-linux-amd64.tar.gz
```

5.3、解压kubernetes（Master01上）

```
tar -xf kubernetes-server-linux-amd64.tar.gz  --strip-components=3 -C /usr/local/bin
kubernetes/server/bin/kube{let,ctl,-apiserver,-controller-manager,-scheduler,-proxy}
```

5.4、解压etcd（Master01上）

```
tar -zxvf etcd-v3.4.13-linux-amd64.tar.gz --strip-components=1 -C /usr/local/bin etcd-
v3.4.13-linux-amd64/etcd{,ctl}
```

5.5、版本查看（Master01上）

```
[root@k8s-master01 ~]# kubelet --version
Kubernetes v1.20.0
[root@k8s-master01 ~]# etcdctl version
etcdctl version: 3.4.13
API version: 3.4
```

5.6、将组件发送到其他节点（Master01上）

```
[root@k8s-master01 ~]# MasterNodes='k8s-master02 k8s-master03'
[root@k8s-master01 ~]# WorkNodes='k8s-node01 k8s-node02'

[root@k8s-master01 ~]# for NODE in $MasterNodes; do echo $NODE; scp
/usr/local/bin/kube{let,ctl,-apiserver,-controller-manager,-scheduler,-proxy}
$NODE:/usr/local/bin/; scp /usr/local/bin/etcd* $NODE:/usr/local/bin/; done

[root@k8s-master01 ~]# for NODE in $WorkNodes; do     scp /usr/local/bin/kube{let,-
proxy} $NODE:/usr/local/bin/ ; done
```

5.7、创建/opt/cni/bin目录（所有节点）

```
mkdir -p /opt/cni/bin
```

5.8、切换分支（Master01上）

```
# 切换到1.20.x分支（其他版本可以切换到其他分支）
# 查看所有分支

[root@k8s-master01 ~]# cd k8s-ha-install/
[root@k8s-master01 k8s-ha-install]# git branch -a
* master
  remotes/origin/HEAD -> origin/master
  remotes/origin/manual-installation
  remotes/origin/manual-installation-v1.16.x
  remotes/origin/manual-installation-v1.17.x
  remotes/origin/manual-installation-v1.18.x
  remotes/origin/manual-installation-v1.19.x
  remotes/origin/manual-installation-v1.20.x
  remotes/origin/master

# 切换分之
git checkout manual-installation-v1.20.x

# 生成证书所需文件，如下所示
[root@k8s-master01 k8s-ha-install]# ls
bootstrap  calico  CoreDNS  dashboard  kube-proxy  metrics-server-0.4.x  metrics-
server-0.4.x-kubeadm  pki  snapshotter
```

六、生成证书

二进制安装最关键步骤，一步错误全盘皆输，一定要注意每个步骤都要是正确的

6.1、下载生成证书工具（Master01）

```
wget "https://pkg.cfssl.org/R1.2/cfssl_linux-amd64" -O /usr/local/bin/cfssl
wget "https://pkg.cfssl.org/R1.2/cfssljson_linux-amd64" -O /usr/local/bin/cfssljson

cd /root
[root@k8s-master01 ~]# mv cfssl_linux-amd64 /usr/local/bin/cfssl
[root@k8s-master01 ~]# mv cfssljson_linux-amd64  /usr/local/bin/cfssljson
[root@k8s-master01 ~]# chmod +x /usr/local/bin/cfssl /usr/local/bin/cfssljson
```

6.2、生成etcd证书

6.2.1、创建etcd证书目录（所有Master节点）

```
mkdir /etc/etcd/ssl -p
```

6.2.1、创建kubernetes证书目录（所有节点）

```
mkdir -p /etc/kubernetes/pki
```

6.2.3、生成etcd证书（Master01节点）将证书复制到其他节点

```
# 生成证书的CSR文件：证书签名请求文件，配置了一些域名、公司、单位
[root@k8s-master01 ~]# cd /root/k8s-ha-install/pki

# 生成etcd CA证书和CA证书的key
[root@k8s-master01 pki]# cfssl gencert -initca etcd-ca-csr.json | cfssljson -bare
/etc/etcd/ssl/etcd-ca
2020/12/21 01:58:02 [INFO] generating a new CA key and certificate from CSR
```

```
2020/12/21 01:58:02 [INFO] generate received request
2020/12/21 01:58:02 [INFO] received CSR
2020/12/21 01:58:02 [INFO] generating key: rsa-2048
2020/12/21 01:58:03 [INFO] encoded CSR
2020/12/21 01:58:03 [INFO] signed certificate with serial number
140198241947074029848239512164671290627608591138

# 可以在-hostname 参数后面预留几个ip，方便日后扩容
[root@k8s-master01 pki]# cfssl gencert \
   -ca=/etc/etcd/ssl/etcd-ca.pem \
   -ca-key=/etc/etcd/ssl/etcd-ca-key.pem \
   -config=ca-config.json \
   -hostname=127.0.0.1,k8s-master01,k8s-master02,k8s-
master03,10.4.7.107,10.4.7.108,10.4.7.109 \
   -profile=kubernetes \
   etcd-csr.json | cfssljson -bare /etc/etcd/ssl/etcd

# 执行结果
2020/12/21 02:00:04 [INFO] generate received request
2020/12/21 02:00:04 [INFO] received CSR
2020/12/21 02:00:04 [INFO] generating key: rsa-2048
2020/12/21 02:00:05 [INFO] encoded CSR
2020/12/21 02:00:05 [INFO] signed certificate with serial number
470467884878418179395781489624244078991295464856
```

6.2.3、将证书复制到其他节点（Master01节点）

```
MasterNodes='k8s-master02 k8s-master03'
WorkNodes='k8s-node01 k8s-node02'

for NODE in $MasterNodes; do
     ssh $NODE "mkdir -p /etc/etcd/ssl"
     for FILE in etcd-ca-key.pem  etcd-ca.pem  etcd-key.pem  etcd.pem; do
       scp /etc/etcd/ssl/${FILE} $NODE:/etc/etcd/ssl/${FILE}
     done
 done
```

6.3、k8s组件证书

6.3.1、生成kubernetes证书（Master 01节点）

```
[root@k8s-master01 ~]# cd /root/k8s-ha-install/pki

[root@k8s-master01 pki]# cfssl gencert -initca ca-csr.json | cfssljson -bare
/etc/kubernetes/pki/ca
# 执行结果
2020/12/21 02:05:33 [INFO] generating a new CA key and certificate from CSR
2020/12/21 02:05:33 [INFO] generate received request
2020/12/21 02:05:33 [INFO] received CSR
```

```
2020/12/21 02:05:33 [INFO] generating key: rsa-2048
2020/12/21 02:05:34 [INFO] encoded CSR
2020/12/21 02:05:34 [INFO] signed certificate with serial number 41601140313910114593243737048758611445671732018

# 10.96.0.1是k8s service的网段，如果说需要更改k8s service网段，那就需要更改10.96.0.1，
# 如果不是高可用集群，10.4.7.236为Master01的IP
[root@k8s-master01 pki]# cfssl gencert   -ca=/etc/kubernetes/pki/ca.pem   -ca-key=/etc/kubernetes/pki/ca-key.pem   -config=ca-config.json -hostname=10.96.0.1,10.4.7.236,127.0.0.1,kubernetes,kubernetes.default,kubernetes.default.svc,kubernetes.default.svc.cluster,kubernetes.default.svc.cluster.local,10.4.7.107,10.4.7.108,10.4.7.109   -profile=kubernetes   apiserver-csr.json | cfssljson -bare /etc/kubernetes/pki/apiserver

# 执行结果
2020/12/21 02:07:26 [INFO] generate received request
2020/12/21 02:07:26 [INFO] received CSR
2020/12/21 02:07:26 [INFO] generating key: rsa-2048
2020/12/21 02:07:26 [INFO] encoded CSR
2020/12/21 02:07:26 [INFO] signed certificate with serial number 538625498609814572541825087295197801303230523180
```

6.3.2、生成apiserver的聚合证书（Master 01节点）

```
[root@k8s-master01 pki]# cfssl gencert   -initca front-proxy-ca-csr.json | cfssljson -bare /etc/kubernetes/pki/front-proxy-ca
# 执行结果
2020/12/21 02:08:45 [INFO] generating a new CA key and certificate from CSR
2020/12/21 02:08:45 [INFO] generate received request
2020/12/21 02:08:45 [INFO] received CSR
2020/12/21 02:08:45 [INFO] generating key: rsa-2048
2020/12/21 02:08:46 [INFO] encoded CSR
2020/12/21 02:08:46 [INFO] signed certificate with serial number 614553480240998616305316696839282255811191572397

[root@k8s-master01 pki]# cfssl gencert   -ca=/etc/kubernetes/pki/front-proxy-ca.pem   -ca-key=/etc/kubernetes/pki/front-proxy-ca-key.pem   -config=ca-config.json   -profile=kubernetes   front-proxy-client-csr.json | cfssljson -bare /etc/kubernetes/pki/front-proxy-client

# 返回结果（忽略警告）
2020/12/21 02:09:23 [INFO] generate received request
2020/12/21 02:09:23 [INFO] received CSR
2020/12/21 02:09:23 [INFO] generating key: rsa-2048
2020/12/21 02:09:23 [INFO] encoded CSR
2020/12/21 02:09:23 [INFO] signed certificate with serial number 525521597243375822253206665544676632452020336672
2020/12/21 02:09:23 [WARNING] This certificate lacks a "hosts" field. This makes it unsuitable for
```

```
websites. For more information see the Baseline Requirements for the Issuance and
Management
of Publicly-Trusted Certificates, v.1.1.6, from the CA/Browser Forum
(https://cabforum.org);
specifically, section 10.2.3 ("Information Requirements").
```

6.3.3、生成controller-manage的证书（Master01节点）

```
[root@k8s-master01 pki]# cfssl gencert \
   -ca=/etc/kubernetes/pki/ca.pem \
   -ca-key=/etc/kubernetes/pki/ca-key.pem \
   -config=ca-config.json \
   -profile=kubernetes \
   manager-csr.json | cfssljson -bare /etc/kubernetes/pki/controller-manager
# 执行结果
2020/12/21 02:10:59 [INFO] generate received request
2020/12/21 02:10:59 [INFO] received CSR
2020/12/21 02:10:59 [INFO] generating key: rsa-2048
2020/12/21 02:10:59 [INFO] encoded CSR
2020/12/21 02:10:59 [INFO] signed certificate with serial number
90004917734039884153079426464391358123145661914
2020/12/21 02:10:59 [WARNING] This certificate lacks a "hosts" field. This makes it
unsuitable for
websites. For more information see the Baseline Requirements for the Issuance and
Management
of Publicly-Trusted Certificates, v.1.1.6, from the CA/Browser Forum
(https://cabforum.org);
specifically, section 10.2.3 ("Information Requirements").


# 注意，如果不是高可用集群，10.4.7.236:8443改为master01的地址，8443改为apiserver的端口，默认是
6443
# set-cluster：设置一个集群项,10.4.7.236是VIP
kubectl config set-cluster kubernetes \
     --certificate-authority=/etc/kubernetes/pki/ca.pem \
     --embed-certs=true \
     --server=https://10.4.7.236:8443 \
     --kubeconfig=/etc/kubernetes/controller-manager.kubeconfig
# 执行结果
Cluster "kubernetes" set.


# 设置一个环境项，一个上下文
[root@k8s-master01 pki]# kubectl config set-context system:kube-controller-
manager@kubernetes \
    --cluster=kubernetes \
    --user=system:kube-controller-manager \
    --kubeconfig=/etc/kubernetes/controller-manager.kubeconfig
# 执行结果
```

```
Context "system:kube-controller-manager@kubernetes" created.


# set-credentials 设置一个用户项
[root@k8s-master01 pki]# kubectl config set-credentials system:kube-controller-manager
\
     --client-certificate=/etc/kubernetes/pki/controller-manager.pem \
     --client-key=/etc/kubernetes/pki/controller-manager-key.pem \
     --embed-certs=true \
     --kubeconfig=/etc/kubernetes/controller-manager.kubeconfig
# 执行结果
User "system:kube-controller-manager" set.


# 使用某个环境当做默认环境
[root@k8s-master01 pki]# kubectl config use-context system:kube-controller-
manager@kubernetes \
     --kubeconfig=/etc/kubernetes/controller-manager.kubeconfig
# 执行结果
Switched to context "system:kube-controller-manager@kubernetes".

# 生成scheduler证书
cfssl gencert \
   -ca=/etc/kubernetes/pki/ca.pem \
   -ca-key=/etc/kubernetes/pki/ca-key.pem \
   -config=ca-config.json \
   -profile=kubernetes \
   scheduler-csr.json | cfssljson -bare /etc/kubernetes/pki/scheduler
# 执行结果
2020/12/21 02:16:12 [INFO] generate received request
2020/12/21 02:16:12 [INFO] received CSR
2020/12/21 02:16:12 [INFO] generating key: rsa-2048
2020/12/21 02:16:12 [INFO] encoded CSR
2020/12/21 02:16:12 [INFO] signed certificate with serial number
741886658001030420505820371082564099763326530 77
2020/12/21 02:16:12 [WARNING] This certificate lacks a "hosts" field. This makes it
unsuitable for
websites. For more information see the Baseline Requirements for the Issuance and
Management
of Publicly-Trusted Certificates, v.1.1.6, from the CA/Browser Forum
(https://cabforum.org);
specifically, section 10.2.3 ("Information Requirements").


# 注意，如果不是高可用集群，10.4.7.236:8443改为master01的地址，8443改为apiserver的端口，默认是
6443
kubectl config set-cluster kubernetes \
     --certificate-authority=/etc/kubernetes/pki/ca.pem \
     --embed-certs=true \
     --server=https://10.4.7.236:8443 \
     --kubeconfig=/etc/kubernetes/scheduler.kubeconfig

kubectl config set-credentials system:kube-scheduler \
     --client-certificate=/etc/kubernetes/pki/scheduler.pem \
     --client-key=/etc/kubernetes/pki/scheduler-key.pem \
     --embed-certs=true \
     --kubeconfig=/etc/kubernetes/scheduler.kubeconfig

kubectl config set-context system:kube-scheduler@kubernetes \
     --cluster=kubernetes \
     --user=system:kube-scheduler \
     --kubeconfig=/etc/kubernetes/scheduler.kubeconfig

kubectl config use-context system:kube-scheduler@kubernetes \
     --kubeconfig=/etc/kubernetes/scheduler.kubeconfig

cfssl gencert \
   -ca=/etc/kubernetes/pki/ca.pem \
   -ca-key=/etc/kubernetes/pki/ca-key.pem \
   -config=ca-config.json \
   -profile=kubernetes \
   admin-csr.json | cfssljson -bare /etc/kubernetes/pki/admin

# 注意，如果不是高可用集群，10.4.7.236:8443改为master01的地址，8443改为apiserver的端口，默认是6443
kubectl config set-cluster kubernetes     --certificate-authority=/etc/kubernetes/pki/ca.pem     --embed-certs=true     --server=https://10.4.7.236:8443     --kubeconfig=/etc/kubernetes/admin.kubeconfig

kubectl config set-credentials kubernetes-admin     --client-certificate=/etc/kubernetes/pki/admin.pem     --client-key=/etc/kubernetes/pki/admin-key.pem     --embed-certs=true     --kubeconfig=/etc/kubernetes/admin.kubeconfig

kubectl config set-context kubernetes-admin@kubernetes     --cluster=kubernetes     --user=kubernetes-admin     --kubeconfig=/etc/kubernetes/admin.kubeconfig

kubectl config use-context kubernetes-admin@kubernetes     --kubeconfig=/etc/kubernetes/admin.kubeconfig
```

6.3.4、创建ServiceAccount Key asecret

```
[root@k8s-master01 pki]# openssl genrsa -out /etc/kubernetes/pki/sa.key 2048
# 执行结果
Generating RSA private key, 2048 bit long modulus
..............................+++
.............................................................+++
e is 65537 (0x10001)

[root@k8s-master01 pki]# openssl rsa -in /etc/kubernetes/pki/sa.key -pubout -out
/etc/kubernetes/pki/sa.pub
# 执行结果
writing RSA key
```

6.3.5、发送证书至其他节点

```
for NODE in k8s-master02 k8s-master03; do
for FILE in $(ls /etc/kubernetes/pki | grep -v etcd); do
scp /etc/kubernetes/pki/${FILE} $NODE:/etc/kubernetes/pki/${FILE};
done;
for FILE in admin.kubeconfig controller-manager.kubeconfig scheduler.kubeconfig; do
scp /etc/kubernetes/${FILE} $NODE:/etc/kubernetes/${FILE};
done;
done
```

6.3.6、查看证书文件

```
[root@k8s-master01 pki]# ls /etc/kubernetes/pki/
admin.csr      apiserver.csr      ca.csr      controller-manager.csr      front-proxy-
ca.csr      front-proxy-client.csr      sa.key         scheduler-key.pem
admin-key.pem  apiserver-key.pem  ca-key.pem  controller-manager-key.pem  front-proxy-
ca-key.pem  front-proxy-client-key.pem  sa.pub         scheduler.pem
admin.pem      apiserver.pem      ca.pem      controller-manager.pem      front-proxy-
ca.pem      front-proxy-client.pem      scheduler.csr

[root@k8s-master01 pki]# ls /etc/kubernetes/pki/ |wc -l
23
```

七、Kubernetes系统组件配置

7.1、Etcd配置（所有Master节点）

etcd配置大致相同，注意修改每个Master节点的etcd配置的主机名和IP地址

7.1.1、Master 01上

```
[root@k8s-master01 ~]# vim /etc/etcd/etcd.config.yml
```

```
name: 'k8s-master01'
data-dir: /var/lib/etcd
wal-dir: /var/lib/etcd/wal
snapshot-count: 5000
heartbeat-interval: 100
election-timeout: 1000
quota-backend-bytes: 0
listen-peer-urls: 'https://10.4.7.107:2380'
listen-client-urls: 'https://10.4.7.107:2379,http://127.0.0.1:2379'
max-snapshots: 3
max-wals: 5
cors:
initial-advertise-peer-urls: 'https://10.4.7.107:2380'
advertise-client-urls: 'https://10.4.7.107:2379'
discovery:
discovery-fallback: 'proxy'
discovery-proxy:
discovery-srv:
initial-cluster: 'k8s-master01=https://10.4.7.107:2380,k8s-master02=https://10.4.7.108:2380,k8s-master03=https://10.4.7.109:2380'
initial-cluster-token: 'etcd-k8s-cluster'
initial-cluster-state: 'new'
strict-reconfig-check: false
enable-v2: true
enable-pprof: true
proxy: 'off'
proxy-failure-wait: 5000
proxy-refresh-interval: 30000
proxy-dial-timeout: 1000
proxy-write-timeout: 5000
proxy-read-timeout: 0
client-transport-security:
  cert-file: '/etc/kubernetes/pki/etcd/etcd.pem'
  key-file: '/etc/kubernetes/pki/etcd/etcd-key.pem'
  client-cert-auth: true
  trusted-ca-file: '/etc/kubernetes/pki/etcd/etcd-ca.pem'
  auto-tls: true
peer-transport-security:
  cert-file: '/etc/kubernetes/pki/etcd/etcd.pem'
  key-file: '/etc/kubernetes/pki/etcd/etcd-key.pem'
  peer-client-cert-auth: true
  trusted-ca-file: '/etc/kubernetes/pki/etcd/etcd-ca.pem'
  auto-tls: true
debug: false
log-package-levels:
log-outputs: [default]
force-new-cluster: false
```

7.1.2、Master 02上

```
[root@k8s-master02 ~]# vim /etc/etcd/etcd.config.yml
```

```
name: 'k8s-master02'
data-dir: /var/lib/etcd
wal-dir: /var/lib/etcd/wal
snapshot-count: 5000
heartbeat-interval: 100
election-timeout: 1000
quota-backend-bytes: 0
listen-peer-urls: 'https://10.4.7.108:2380'
listen-client-urls: 'https://10.4.7.108:2379,http://127.0.0.1:2379'
max-snapshots: 3
max-wals: 5
cors:
initial-advertise-peer-urls: 'https://10.4.7.108:2380'
advertise-client-urls: 'https://10.4.7.108:2379'
discovery:
discovery-fallback: 'proxy'
discovery-proxy:
discovery-srv:
initial-cluster: 'k8s-master01=https://10.4.7.107:2380,k8s-master02=https://10.4.7.108:2380,k8s-master03=https://10.4.7.109:2380'
initial-cluster-token: 'etcd-k8s-cluster'
initial-cluster-state: 'new'
strict-reconfig-check: false
enable-v2: true
enable-pprof: true
proxy: 'off'
proxy-failure-wait: 5000
proxy-refresh-interval: 30000
proxy-dial-timeout: 1000
proxy-write-timeout: 5000
proxy-read-timeout: 0
client-transport-security:
  cert-file: '/etc/kubernetes/pki/etcd/etcd.pem'
  key-file: '/etc/kubernetes/pki/etcd/etcd-key.pem'
  client-cert-auth: true
  trusted-ca-file: '/etc/kubernetes/pki/etcd/etcd-ca.pem'
  auto-tls: true
peer-transport-security:
  cert-file: '/etc/kubernetes/pki/etcd/etcd.pem'
  key-file: '/etc/kubernetes/pki/etcd/etcd-key.pem'
  peer-client-cert-auth: true
  trusted-ca-file: '/etc/kubernetes/pki/etcd/etcd-ca.pem'
  auto-tls: true
debug: false
log-package-levels:
log-outputs: [default]
force-new-cluster: false
```

7.1.3、Master 03上

```
[root@k8s-master03 ~]# vim /etc/etcd/etcd.config.yml
```

```
name: 'k8s-master03'
data-dir: /var/lib/etcd
wal-dir: /var/lib/etcd/wal
snapshot-count: 5000
heartbeat-interval: 100
election-timeout: 1000
quota-backend-bytes: 0
listen-peer-urls: 'https://10.4.7.109:2380'
listen-client-urls: 'https://10.4.7.109:2379,http://127.0.0.1:2379'
max-snapshots: 3
max-wals: 5
cors:
initial-advertise-peer-urls: 'https://10.4.7.109:2380'
advertise-client-urls: 'https://10.4.7.109:2379'
discovery:
discovery-fallback: 'proxy'
discovery-proxy:
discovery-srv:
initial-cluster: 'k8s-master01=https://10.4.7.107:2380,k8s-master02=https://10.4.7.108:2380,k8s-master03=https://10.4.7.109:2380'
initial-cluster-token: 'etcd-k8s-cluster'
initial-cluster-state: 'new'
strict-reconfig-check: false
enable-v2: true
enable-pprof: true
proxy: 'off'
proxy-failure-wait: 5000
proxy-refresh-interval: 30000
proxy-dial-timeout: 1000
proxy-write-timeout: 5000
proxy-read-timeout: 0
client-transport-security:
  cert-file: '/etc/kubernetes/pki/etcd/etcd.pem'
  key-file: '/etc/kubernetes/pki/etcd/etcd-key.pem'
  client-cert-auth: true
  trusted-ca-file: '/etc/kubernetes/pki/etcd/etcd-ca.pem'
  auto-tls: true
peer-transport-security:
  cert-file: '/etc/kubernetes/pki/etcd/etcd.pem'
  key-file: '/etc/kubernetes/pki/etcd/etcd-key.pem'
  peer-client-cert-auth: true
  trusted-ca-file: '/etc/kubernetes/pki/etcd/etcd-ca.pem'
  auto-tls: true
debug: false
log-package-levels:
log-outputs: [default]
force-new-cluster: false
```

7.2、 创建Service

7.2.1、创建etcd service并启动（所有Master节点）

```
vim /usr/lib/systemd/system/etcd.service
```

```
[Unit]
Description=Etcd Service
Documentation=https://coreos.com/etcd/docs/latest/
After=network.target

[Service]
Type=notify
ExecStart=/usr/local/bin/etcd --config-file=/etc/etcd/etcd.config.yml
Restart=on-failure
RestartSec=10
LimitNOFILE=65536

[Install]
WantedBy=multi-user.target
Alias=etcd3.service
```

7.2.2、创建etcd的证书目录（所有Master节点）

```
#所有
mkdir /etc/kubernetes/pki/etcd
ln -s /etc/etcd/ssl/* /etc/kubernetes/pki/etcd/
systemctl daemon-reload
systemctl enable --now etcd
```

7.2.3、查看集群状态（任意master）

```
[root@k8s-master03 ~]# export ETCDCTL_API=3
[root@k8s-master03 ~]# etcdctl --
endpoints="10.4.7.109:2379,10.4.7.108:2379,10.4.7.107:2379" --
cacert=/etc/kubernetes/pki/etcd/etcd-ca.pem --cert=/etc/kubernetes/pki/etcd/etcd.pem --
key=/etc/kubernetes/pki/etcd/etcd-key.pem  endpoint status --write-out=table
+--------------------+------------------+---------+---------+-----------+------------+-
----------+------------+--------------------+--------+
|      ENDPOINT      |        ID        | VERSION | DB SIZE | IS LEADER | IS LEARNER |
RAFT TERM | RAFT INDEX | RAFT APPLIED INDEX | ERRORS |
+--------------------+------------------+---------+---------+-----------+------------+-
----------+------------+--------------------+--------+
| 10.4.7.109:2379 | a77e5ca7bd4dc035 |  3.4.13 |   20 kB |     false |      false |
  465 |          9 |                 9 |        |
| 10.4.7.108:2379 | e3adc675ac3b3dbd |  3.4.13 |   20 kB |     false |      false |
  465 |          9 |                 9 |        |
| 10.4.7.107:2379 | 47a087175e3f17b3 |  3.4.13 |   20 kB |      true |      false |
  465 |          9 |                 9 |        |
+--------------------+------------------+---------+---------+-----------+------------+-
----------+------------+--------------------+--------+
```

八、高可用配置

```
高可用配置（注意：如果不是高可用集群，haproxy和keepalived无需安装）
如果在云上安装也无需执行此章节的步骤，可以直接使用云上的lb，比如阿里云slb，腾讯云elb等
Slb -> haproxy -> apiserver
```

8.1、安装keepalived和haproxy（所有Master节点）

```
yum install keepalived haproxy -y
```

8.2、Master配置HAProxy，Master节点都配置一样

```
vim /etc/haproxy/haproxy.cfg
```

```
 global
  maxconn  2000
  ulimit-n  16384
  log  127.0.0.1 local0 err
  stats timeout 30s

defaults
  log global
  mode  http
  option  httplog
  timeout connect 5000
  timeout client  50000
  timeout server  50000
  timeout http-request 15s
  timeout http-keep-alive 15s

frontend k8s-master
  bind 0.0.0.0:8443
  bind 127.0.0.1:8443
  mode tcp
  option tcplog
  tcp-request inspect-delay 5s
  default_backend k8s-master

backend k8s-master
  mode tcp
  option tcplog
  option tcp-check
  balance roundrobin
  default-server inter 10s downinter 5s rise 2 fall 2 slowstart 60s maxconn 250 maxqueue 256 weight 100
  server k8s-master01    10.4.7.107:6443  check
  server k8s-master02    10.4.7.108:6443  check
  server k8s-master03    10.4.7.109:6443  check
```

8.3、配置KeepAlived（Master节点）

注意每个节点的IP和网卡（interface参数）

8.3.1、Master01

```
vim /etc/keepalived/keepalived.conf
```

```
! Configuration File for keepalived
global_defs {
    router_id LVS_DEVEL
}
vrrp_script chk_apiserver {
    script "/etc/keepalived/check_apiserver.sh"
    interval 5
    weight -5
    fall 2
    rise 1
}
vrrp_instance VI_1 {
    state MASTER
    interface ens33
    mcast_src_ip 10.4.7.107
    virtual_router_id 51
    priority 101
    nopreempt
    advert_int 2
    authentication {
        auth_type PASS
        auth_pass K8SHA_KA_AUTH
    }
    virtual_ipaddress {
        10.4.7.236
    }
    track_script {
      chk_apiserver
} }
```

8.3.2、Master02

```
vim /etc/keepalived/keepalived.conf
```

```
! Configuration File for keepalived
global_defs {
    router_id LVS_DEVEL
}
vrrp_script chk_apiserver {
    script "/etc/keepalived/check_apiserver.sh"
    interval 5
    weight -5
    fall 2
    rise 1

}
vrrp_instance VI_1 {
    state BACKUP
    interface ens33
    mcast_src_ip 10.4.7.108
    virtual_router_id 51
    priority 100
    nopreempt
    advert_int 2
    authentication {
        auth_type PASS
        auth_pass K8SHA_KA_AUTH
    }
    virtual_ipaddress {
        10.4.7.236
    }
    track_script {
      chk_apiserver
} }
```

8.3.3、Master03

```
vim /etc/keepalived/keepalived.conf
```

```
! Configuration File for keepalived
global_defs {
    router_id LVS_DEVEL
}
vrrp_script chk_apiserver {
    script "/etc/keepalived/check_apiserver.sh"
    interval 5
    weight -5
    fall 2
    rise 1
}
vrrp_instance VI_1 {
    state BACKUP
    interface ens33
    mcast_src_ip 10.4.7.109
    virtual_router_id 51
    priority 100
    nopreempt
    advert_int 2
    authentication {
        auth_type PASS
        auth_pass K8SHA_KA_AUTH
    }
    virtual_ipaddress {
        10.4.7.236
    }
    track_script {
      chk_apiserver
} }
```

8.3.4、健康检查脚本（所有master节点）

```
cat > /etc/keepalived/check_apiserver.sh  << EFO
#!/bin/bash

err=0
for k in $(seq 1 3)
do
    check_code=$(pgrep haproxy)
    if [[ $check_code == "" ]]; then
        err=$(expr $err + 1)
        sleep 1
        continue
    else
```

```
        err=0
        break
    fi
done

if [[ $err != "0" ]]; then
    echo "systemctl stop keepalived"
    /usr/bin/systemctl stop keepalived
    exit 1
else
    exit 0
fi
EFO

# 授权
chmod +x /etc/keepalived/check_apiserver.sh
```

8.3.5、节点启动haproxy和keepalived（所有master节点）

```
systemctl daemon-reload
systemctl enable --now haproxy
systemctl enable --now keepalived
```

8.3.6、VIP测试 （master01）

重要：如果安装了keepalived和haproxy，需要测试keepalived是否是正常的

```
# 看到有VIP绑定到ens33网卡上了
[root@k8s-master01 ~]# ip a | grep ens33
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group
default qlen 1000
    inet 10.4.7.107/24 brd 192.168.1.255 scope global ens33
    inet 10.4.7.236/32 scope global ens33

# 任意节点，检查haproxy
telnet 10.4.7.236 8443

如果ping不通且telnet没有出现 "]"，则认为VIP不可以，不可在继续往下执行，需要排查keepalived的问题，比
如防火墙和selinux，haproxy和keepalived的状态，监听端口等
所有节点查看防火墙状态必须为disable和inactive：systemctl status firewalld
所有节点查看selinux状态，必须为disable：getenforce
master节点查看haproxy和keepalived状态：systemctl status keepalived haproxy
master节点查看监听端口：netstat -lntp
```

九、Kubernetes组件配置

9.1、Apiserver

所有Master节点创建kube-apiserver service，# 注意，如果不是高可用集群，10.4.7.236改为master01的地址

注意本文档使用的k8s service网段为10.96.0.0/12，该网段不能和宿主机的网段、Pod网段的重复，请按需修改

9.1.1、Master01配置

```
vim /usr/lib/systemd/system/kube-apiserver.service
```

```
[Unit]
Description=Kubernetes API Server
Documentation=https://github.com/kubernetes/kubernetes
After=network.target

[Service]
ExecStart=/usr/local/bin/kube-apiserver \
      --v=2  \
      --logtostderr=true  \
      --allow-privileged=true  \
      --bind-address=0.0.0.0  \
      --secure-port=6443  \
      --insecure-port=0  \
      --advertise-address=10.4.7.107 \
      --service-cluster-ip-range=10.96.0.0/12  \
      --service-node-port-range=30000-32767  \
      --etcd-servers=https://10.4.7.107:2379,https://10.4.7.108:2379,https://10.4.7.109:2379 \
      --etcd-cafile=/etc/etcd/ssl/etcd-ca.pem  \
      --etcd-certfile=/etc/etcd/ssl/etcd.pem  \
      --etcd-keyfile=/etc/etcd/ssl/etcd-key.pem  \
      --client-ca-file=/etc/kubernetes/pki/ca.pem  \
      --tls-cert-file=/etc/kubernetes/pki/apiserver.pem  \
      --tls-private-key-file=/etc/kubernetes/pki/apiserver-key.pem  \
      --kubelet-client-certificate=/etc/kubernetes/pki/apiserver.pem  \
      --kubelet-client-key=/etc/kubernetes/pki/apiserver-key.pem  \
      --service-account-key-file=/etc/kubernetes/pki/sa.pub  \
      --service-account-signing-key-file=/etc/kubernetes/pki/sa.key  \
      --service-account-issuer=https://kubernetes.default.svc.cluster.local \
      --kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname  \
      --enable-admission-plugins=NamespaceLifecycle,LimitRanger,ServiceAccount,DefaultStorageClass,DefaultTolerationSeconds,NodeRestriction,ResourceQuota  \
      --authorization-mode=Node,RBAC  \
      --enable-bootstrap-token-auth=true  \
      --requestheader-client-ca-file=/etc/kubernetes/pki/front-proxy-ca.pem  \
      --proxy-client-cert-file=/etc/kubernetes/pki/front-proxy-client.pem  \
      --proxy-client-key-file=/etc/kubernetes/pki/front-proxy-client-key.pem  \
      --requestheader-allowed-names=aggregator  \
      --requestheader-group-headers=X-Remote-Group  \
      --requestheader-extra-headers-prefix=X-Remote-Extra-  \
      --requestheader-username-headers=X-Remote-User
      # --token-auth-file=/etc/kubernetes/token.csv

Restart=on-failure
RestartSec=10s
LimitNOFILE=65535

[Install]
WantedBy=multi-user.target
```

9.1.2、Master02配置

```
vim /usr/lib/systemd/system/kube-apiserver.service
```

```
[Unit]
Description=Kubernetes API Server
Documentation=https://github.com/kubernetes/kubernetes
After=network.target

[Service]
ExecStart=/usr/local/bin/kube-apiserver \
      --v=2  \
      --logtostderr=true  \
      --allow-privileged=true  \
      --bind-address=0.0.0.0  \
      --secure-port=6443  \
      --insecure-port=0  \
      --advertise-address=10.4.7.108 \
      --service-cluster-ip-range=10.96.0.0/12  \
      --service-node-port-range=30000-32767  \
      --etcd-
servers=https://10.4.7.107:2379,https://10.4.7.108:2379,https://10.4.7.109:2379 \
      --etcd-cafile=/etc/etcd/ssl/etcd-ca.pem  \
      --etcd-certfile=/etc/etcd/ssl/etcd.pem  \
      --etcd-keyfile=/etc/etcd/ssl/etcd-key.pem  \
      --client-ca-file=/etc/kubernetes/pki/ca.pem  \
      --tls-cert-file=/etc/kubernetes/pki/apiserver.pem  \
      --tls-private-key-file=/etc/kubernetes/pki/apiserver-key.pem  \
      --kubelet-client-certificate=/etc/kubernetes/pki/apiserver.pem  \
      --kubelet-client-key=/etc/kubernetes/pki/apiserver-key.pem  \
      --service-account-key-file=/etc/kubernetes/pki/sa.pub  \
      --service-account-signing-key-file=/etc/kubernetes/pki/sa.key  \
      --service-account-issuer=https://kubernetes.default.svc.cluster.local \
      --kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname  \
      --enable-admission-plugins=NamespaceLifecycle,LimitRanger,ServiceAccount,DefaultStorageClass,DefaultTolerationSeconds,NodeRestriction,ResourceQuota  \
      --authorization-mode=Node,RBAC  \
      --enable-bootstrap-token-auth=true  \
      --requestheader-client-ca-file=/etc/kubernetes/pki/front-proxy-ca.pem  \
      --proxy-client-cert-file=/etc/kubernetes/pki/front-proxy-client.pem  \
      --proxy-client-key-file=/etc/kubernetes/pki/front-proxy-client-key.pem  \
      --requestheader-allowed-names=aggregator  \
      --requestheader-group-headers=X-Remote-Group  \
      --requestheader-extra-headers-prefix=X-Remote-Extra-  \
      --requestheader-username-headers=X-Remote-User
      # --token-auth-file=/etc/kubernetes/token.csv

Restart=on-failure
RestartSec=10s
LimitNOFILE=65535

[Install]
WantedBy=multi-user.target
```

9.1.3、Master03配置

```
vim /usr/lib/systemd/system/kube-apiserver.service
```

```
[Unit]
Description=Kubernetes API Server
Documentation=https://github.com/kubernetes/kubernetes
After=network.target

[Service]
ExecStart=/usr/local/bin/kube-apiserver \
      --v=2  \
      --logtostderr=true  \
      --allow-privileged=true  \
      --bind-address=0.0.0.0  \
      --secure-port=6443  \
      --insecure-port=0  \
      --advertise-address=10.4.7.109 \
      --service-cluster-ip-range=10.96.0.0/12  \
      --service-node-port-range=30000-32767  \
      --etcd-servers=https://10.4.7.107:2379,https://10.4.7.108:2379,https://10.4.7.109:2379 \
      --etcd-cafile=/etc/etcd/ssl/etcd-ca.pem  \
      --etcd-certfile=/etc/etcd/ssl/etcd.pem  \
      --etcd-keyfile=/etc/etcd/ssl/etcd-key.pem  \
      --client-ca-file=/etc/kubernetes/pki/ca.pem  \
      --tls-cert-file=/etc/kubernetes/pki/apiserver.pem  \
      --tls-private-key-file=/etc/kubernetes/pki/apiserver-key.pem  \
      --kubelet-client-certificate=/etc/kubernetes/pki/apiserver.pem  \
      --kubelet-client-key=/etc/kubernetes/pki/apiserver-key.pem  \
      --service-account-key-file=/etc/kubernetes/pki/sa.pub  \
      --service-account-signing-key-file=/etc/kubernetes/pki/sa.key  \
      --service-account-issuer=https://kubernetes.default.svc.cluster.local \
      --kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname  \
      --enable-admission-plugins=NamespaceLifecycle,LimitRanger,ServiceAccount,DefaultStorageClass,DefaultTolerationSeconds,NodeRestriction,ResourceQuota  \
      --authorization-mode=Node,RBAC  \
      --enable-bootstrap-token-auth=true  \
      --requestheader-client-ca-file=/etc/kubernetes/pki/front-proxy-ca.pem  \
      --proxy-client-cert-file=/etc/kubernetes/pki/front-proxy-client.pem  \
      --proxy-client-key-file=/etc/kubernetes/pki/front-proxy-client-key.pem  \
      --requestheader-allowed-names=aggregator  \
      --requestheader-group-headers=X-Remote-Group  \
      --requestheader-extra-headers-prefix=X-Remote-Extra-  \
      --requestheader-username-headers=X-Remote-User
      # --token-auth-file=/etc/kubernetes/token.csv

Restart=on-failure
RestartSec=10s
LimitNOFILE=65535

[Install]
WantedBy=multi-user.target
```

9.1.4、启动apiserver（所有Master节点）

```
systemctl daemon-reload && systemctl enable --now kube-apiserver


# 查看日志
tail -f /var/log/messages

# 检测kube-server状态
systemctl status kube-apiserver

# 执行结果
● kube-apiserver.service - Kubernetes API Server
   Loaded: loaded (/usr/lib/systemd/system/kube-apiserver.service; enabled; vendor preset: disabled)
   Active: active (running) since Mon 2020-12-21 03:50:34 CST; 56s ago
     Docs: https://github.com/kubernetes/kubernetes

# 以下日志不需要理会，是正常的日志
balancer_conn_wrappers.go:78] pickfirstBalancer: HandleSubConnStateChange: 0xc01124e0e0, {READY <nil>}
```

```
Dec 21 03:51:19 k8s-master01 kube-apiserver[10428]: I1221 03:51:19.699478   10428
controlbuf.go:508] transport: loopyWriter.run returning. connection error: desc =
"transport is closing"
```

9.2、配置kube-controller-manager service （所有Master节点）

注意本文档使用的k8s Pod网段为172.16.0.0/12，该网段不能和宿主机的网段、k8s Service网段的重复，请按需修改

9.2.1、三Master配置文件节点相同

```
vim /usr/lib/systemd/system/kube-controller-manager.service
```

```
[Unit]
Description=Kubernetes Controller Manager
Documentation=https://github.com/kubernetes/kubernetes
After=network.target

[Service]
ExecStart=/usr/local/bin/kube-controller-manager \
      --v=2 \
      --logtostderr=true \
      --address=127.0.0.1 \
      --root-ca-file=/etc/kubernetes/pki/ca.pem \
      --cluster-signing-cert-file=/etc/kubernetes/pki/ca.pem \
      --cluster-signing-key-file=/etc/kubernetes/pki/ca-key.pem \
      --service-account-private-key-file=/etc/kubernetes/pki/sa.key \
      --kubeconfig=/etc/kubernetes/controller-manager.kubeconfig \
      --leader-elect=true \
      --use-service-account-credentials=true \
      --node-monitor-grace-period=40s \
      --node-monitor-period=5s \
      --pod-eviction-timeout=2m0s \
      --controllers=*,bootstrapsigner,tokencleaner \
      --allocate-node-cidrs=true \
      --cluster-cidr=172.16.0.0/12 \
      --requestheader-client-ca-file=/etc/kubernetes/pki/front-proxy-ca.pem \
      --node-cidr-mask-size=24

Restart=always
RestartSec=10s

[Install]
WantedBy=multi-user.target
```

9.2.2、启动（所有matser节点）

```
systemctl daemon-reload && systemctl enable --now kube-controller-manager
```

9.2.3、查看状态（所有matser节点）

```
systemctl  status kube-controller-manager

● kube-controller-manager.service - Kubernetes Controller Manager
   Loaded: loaded (/usr/lib/systemd/system/kube-controller-manager.service; enabled;
vendor preset: disabled)
   Active: active (running) since Mon 2020-12-21 03:59:21 CST; 28s ago
     Docs: https://github.com/kubernetes/kubernetes
 Main PID: 13594 (kube-controller)
```

9.3、配置kube-scheduler service（所有Master节点）

9.3.1、所有master节点配置文件相同

```
vim /usr/lib/systemd/system/kube-scheduler.service
```

```
[Unit]
Description=Kubernetes Scheduler
Documentation=https://github.com/kubernetes/kubernetes
After=network.target

[Service]
ExecStart=/usr/local/bin/kube-scheduler \
      --v=2 \
      --logtostderr=true \
      --address=127.0.0.1 \
      --leader-elect=true \
      --kubeconfig=/etc/kubernetes/scheduler.kubeconfig

Restart=always
RestartSec=10s

[Install]
WantedBy=multi-user.target
```

9.3.2、启动

```
 systemctl daemon-reload && systemctl enable --now kube-scheduler
```

9.3.3、查看状态

```
systemctl status kube-scheduler

● kube-scheduler.service - Kubernetes Scheduler
   Loaded: loaded (/usr/lib/systemd/system/kube-scheduler.service; enabled; vendor preset: disabled)
   Active: active (running) since Mon 2020-12-21 04:03:50 CST; 17s ago
     Docs: https://github.com/kubernetes/kubernetes
 Main PID: 10915 (kube-scheduler)
```

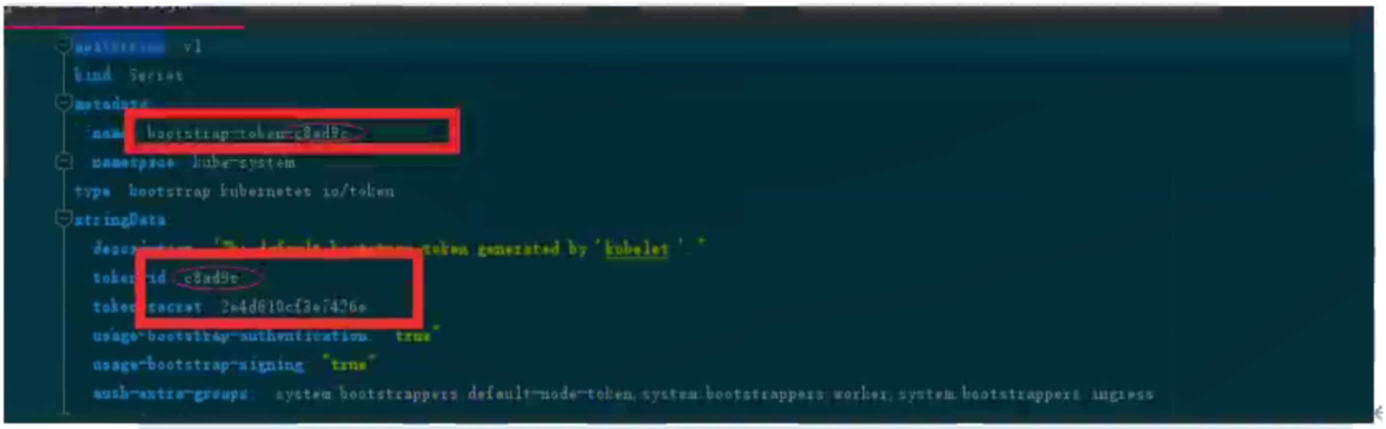
十、TLS Bootstrapping配置

10.1、创建bootstrap（Master01节点）

注意，如果不是高可用集群，10.4.7.236:8443改为master01的地址，8443改为apiserver的端口，默认是6443

```
[root@k8s-master01 ~]# cd /root/k8s-ha-install/bootstrap

kubectl config set-cluster kubernetes     --certificate-authority=/etc/kubernetes/pki/ca.pem     --embed-certs=true     --server=https://10.4.7.236:8443     --kubeconfig=/etc/kubernetes/bootstrap-kubelet.kubeconfig

kubectl config set-credentials tls-bootstrap-token-user     --token=c8ad9c.2e4d610cf3e7426e --kubeconfig=/etc/kubernetes/bootstrap-kubelet.kubeconfig

kubectl config set-context tls-bootstrap-token-user@kubernetes     --cluster=kubernetes     --user=tls-bootstrap-token-user     --kubeconfig=/etc/kubernetes/bootstrap-kubelet.kubeconfig

kubectl config use-context tls-bootstrap-token-user@kubernetes     --kubeconfig=/etc/kubernetes/bootstrap-kubelet.kubeconfig
```

 注意：如果要修改bootstrap.secret.yaml的token-id和token-secret，需要保证下图红圈内的字符串一致的，并且位数是一样的。还要保证上个命令的黄色字体：c8ad9c.2e4d610cf3e7426e与你修改的字符串要一致



```
apiVersion: v1
kind: Secret
metadata:
  name: bootstrap-token-c8ad9c
  namespace: kube-system
type: bootstrap.kubernetes.io/token
stringData:
  description: "The default bootstrap token generated by 'kubelet '."
  token-id: c8ad9c
  token-secret: 2e4d610cf3e7426e
  usage-bootstrap-authentication: "true"
  usage-bootstrap-signing: "true"
  auth-extra-groups: system:bootstrappers:default-node-token,system:bootstrappers:worker,system:bootstrappers:ingress
```

```
apiVersion: v1
kind: Secret
metadata:
  name: bootstrap-token-c8ad9c
  namespace: kube-system
type: bootstrap.kubernetes.io/token
stringData:
  description: "The default bootstrap token generated by 'kubelet '."
  token-id: c8ad9c    #这个跟metadata.name 后面那个一样
```

```
[root@k8s-master01 ~]# cd /root/k8s-ha-install/bootstrap
[root@k8s-master01 bootstrap]# mkdir -p /root/.kube ; cp
/etc/kubernetes/admin.kubeconfig /root/.kube/config

[root@k8s-master01 bootstrap]# kubectl create -f bootstrap.secret.yaml
secret/bootstrap-token-c8ad9c created
clusterrolebinding.rbac.authorization.k8s.io/kubelet-bootstrap created
clusterrolebinding.rbac.authorization.k8s.io/node-autoapprove-bootstrap created
clusterrolebinding.rbac.authorization.k8s.io/node-autoapprove-certificate-rotation
created
clusterrole.rbac.authorization.k8s.io/system:kube-apiserver-to-kubelet created
clusterrolebinding.rbac.authorization.k8s.io/system:kube-apiserver created
```

十一、Node节点配置

11.1、复制证书至Node节点

```
[root@k8s-master01 ~]# cd /etc/kubernetes/

for NODE in k8s-master02 k8s-master03 k8s-node01 k8s-node02; do
     ssh $NODE mkdir -p /etc/kubernetes/pki /etc/etcd/ssl /etc/etcd/ssl
     for FILE in etcd-ca.pem etcd.pem etcd-key.pem; do
       scp /etc/etcd/ssl/$FILE $NODE:/etc/etcd/ssl/
     done
     for FILE in pki/ca.pem pki/ca-key.pem pki/front-proxy-ca.pem bootstrap-kubelet.kubeconfig; do
       scp /etc/kubernetes/$FILE $NODE:/etc/kubernetes/${FILE}
  done
 done
```

11.2、Kubelet配置

11.2.1、创建相关目录（所有节点）

```
mkdir -p /var/lib/kubelet /var/log/kubernetes /etc/systemd/system/kubelet.service.d /etc/kubernetes/manifests/
```

11.2.2、配置kubelet service（所有节点）

```
vim  /usr/lib/systemd/system/kubelet.service
```

```
[Unit]
Description=Kubernetes Kubelet
Documentation=https://github.com/kubernetes/kubernetes
After=docker.service
Requires=docker.service

[Service]
ExecStart=/usr/local/bin/kubelet

Restart=always
StartLimitInterval=0
RestartSec=10

[Install]
WantedBy=multi-user.target
```

11.2.3、配置kubelet service的配置文件（所有节点）

```
vim /etc/systemd/system/kubelet.service.d/10-kubelet.conf
```

```
[Service]
Environment="KUBELET_KUBECONFIG_ARGS=--bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.kubeconfig --kubeconfig=/etc/kubernetes/kubelet.kubeconfig"
Environment="KUBELET_SYSTEM_ARGS=--network-plugin=cni --cni-conf-dir=/etc/cni/net.d --cni-bin-dir=/opt/cni/bin"
Environment="KUBELET_CONFIG_ARGS=--config=/etc/kubernetes/kubelet-conf.yml --pod-infra-container-image=registry.cn-hangzhou.aliyuncs.com/google_containers/pause-amd64:3.2"
Environment="KUBELET_EXTRA_ARGS=--node-labels=node.kubernetes.io/node='' "
ExecStart=
ExecStart=/usr/local/bin/kubelet $KUBELET_KUBECONFIG_ARGS $KUBELET_CONFIG_ARGS $KUBELET_SYSTEM_ARGS $KUBELET_EXTRA_ARGS
```

11.2.4、kubelet的配置文件（所有节点）启动所有节点kubelet

注意：如果更改了k8s的service网段，需要更改kubelet-conf.yml 的clusterDNS:配置，改成k8s Service网段的第十个地址，比如10.96.0.10（k8s的service网段开始设置的是10.96.0.0/12）

```
vim /etc/kubernetes/kubelet-conf.yml
```

```
apiVersion: kubelet.config.k8s.io/v1beta1
kind: KubeletConfiguration
address: 0.0.0.0
port: 10250
readOnlyPort: 10255
authentication:
  anonymous:
    enabled: false
  webhook:
    cacheTTL: 2m0s
    enabled: true
  x509:
    clientCAFile: /etc/kubernetes/pki/ca.pem
authorization:
  mode: Webhook
  webhook:
    cacheAuthorizedTTL: 5m0s
    cacheUnauthorizedTTL: 30s
cgroupDriver: systemd
cgroupsPerQOS: true
clusterDNS:
```

```
- 10.96.0.10
clusterDomain: cluster.local
containerLogMaxFiles: 5
containerLogMaxSize: 10Mi
contentType: application/vnd.kubernetes.protobuf
cpuCFSQuota: true
cpuManagerPolicy: none
cpuManagerReconcilePeriod: 10s
enableControllerAttachDetach: true
enableDebuggingHandlers: true
enforceNodeAllocatable:
- pods
eventBurst: 10
eventRecordQPS: 5
evictionHard:
  imagefs.available: 15%
  memory.available: 100Mi
  nodefs.available: 10%
  nodefs.inodesFree: 5%
evictionPressureTransitionPeriod: 5m0s
failSwapOn: true
fileCheckFrequency: 20s
hairpinMode: promiscuous-bridge
healthzBindAddress: 127.0.0.1
healthzPort: 10248
httpCheckFrequency: 20s
imageGCHighThresholdPercent: 85
imageGCLowThresholdPercent: 80
imageMinimumGCAge: 2m0s
iptablesDropBit: 15
iptablesMasqueradeBit: 14
kubeAPIBurst: 10
kubeAPIQPS: 5
makeIPTablesUtilChains: true
maxOpenFiles: 1000000
maxPods: 110
nodeStatusUpdateFrequency: 10s
oomScoreAdj: -999
podPidsLimit: -1
registryBurst: 10
registryPullQPS: 5
resolvConf: /etc/resolv.conf
rotateCertificates: true
runtimeRequestTimeout: 2m0s
serializeImagePulls: true
staticPodPath: /etc/kubernetes/manifests
streamingConnectionIdleTimeout: 4h0m0s
syncFrequency: 1m0s
volumeStatsAggPeriod: 1m0s
```

11.2.5、启动kubelet（所有节点）

```
systemctl daemon-reload
systemctl enable --now kubelet
# 查看此时系统日志
tail -f /var/log/messages
```

11.2.6、查看集群状态（matser01上）

```
[root@k8s-master01 ~]# kubectl get node
NAME           STATUS     ROLES    AGE     VERSION
k8s-master01   NotReady   <none>   3m48s   v1.20.0
k8s-master02   NotReady   <none>   3m48s   v1.20.0
k8s-master03   NotReady   <none>   3m48s   v1.20.0
k8s-node01     NotReady   <none>   3m47s   v1.20.0
k8s-node02     NotReady   <none>   3m48s   v1.20.0
```

11.3、kube-proxy配置

注意，如果不是高可用集群，10.4.7.236:8443改为master01的地址，8443改为apiserver的端口，默认是6443

11.3.1、Master01执行

```
[root@k8s-master01 ~]# cd /root/k8s-ha-install

kubectl -n kube-system create serviceaccount kube-proxy

kubectl create clusterrolebinding system:kube-proxy         --clusterrole system:node-
proxier         --serviceaccount kube-system:kube-proxy

SECRET=$(kubectl -n kube-system get sa/kube-proxy \
    --output=jsonpath='{.secrets[0].name}')
JWT_TOKEN=$(kubectl -n kube-system get secret/$SECRET \
--output=jsonpath='{.data.token}' | base64 -d)
PKI_DIR=/etc/kubernetes/pki
K8S_DIR=/etc/kubernetes

kubectl config set-cluster kubernetes     --certificate-
authority=/etc/kubernetes/pki/ca.pem     --embed-certs=true     --
server=https://10.4.7.236:8443     --kubeconfig=${K8S_DIR}/kube-proxy.kubeconfig

kubectl config set-credentials kubernetes     --token=${JWT_TOKEN}     --
kubeconfig=/etc/kubernetes/kube-proxy.kubeconfig

kubectl config set-context kubernetes     --cluster=kubernetes     --user=kubernetes
  --kubeconfig=/etc/kubernetes/kube-proxy.kubeconfig

kubectl config use-context kubernetes     --kubeconfig=/etc/kubernetes/kube-
proxy.kubeconfig
```

11.3.2、发送kube-proxy的systemd Service文件发送到其他节点（master01上）

如果更改了集群Pod的网段，需要更改kube-proxy/kube-proxy.conf的clusterCIDR: 172.16.0.0/12参数为pod的网段。

```
[root@k8s-master01 ~]# vim /root/k8s-ha-install/kube-proxy/kube-proxy.conf
clusterCIDR: 172.16.0.0/12
```

分发配置文件（master01上）

```
[root@k8s-master01 ~]# cd /root/k8s-ha-install
for NODE in k8s-master01 k8s-master02 k8s-master03; do
     scp ${K8S_DIR}/kube-proxy.kubeconfig $NODE:/etc/kubernetes/kube-proxy.kubeconfig
     scp kube-proxy/kube-proxy.conf $NODE:/etc/kubernetes/kube-proxy.conf
     scp kube-proxy/kube-proxy.service $NODE:/usr/lib/systemd/system/kube-proxy.service
 done

for NODE in k8s-node01 k8s-node02; do
     scp /etc/kubernetes/kube-proxy.kubeconfig $NODE:/etc/kubernetes/kube-proxy.kubeconfig
     scp kube-proxy/kube-proxy.conf $NODE:/etc/kubernetes/kube-proxy.conf
     scp kube-proxy/kube-proxy.service $NODE:/usr/lib/systemd/system/kube-proxy.service
 done
```

11..3.3、启动kube-proxy（所有节点）

```
 systemctl daemon-reload && systemctl enable --now kube-proxy
```

十二、安装Calico

Calico的安装请必须听视频课程和最后一章升级Calico的视频

12.1、安装Calico（在master01上）

```
[root@k8s-master01 ~]# cd /root/k8s-ha-install/calico/

# 修改calico-etcd.yaml的以下位置
sed -i 's#etcd_endpoints: "http://<ETCD_IP>:<ETCD_PORT>"#etcd_endpoints: "https://10.4.7.107:2379,https://10.4.7.108:2379,https://10.4.7.109:2379"#g' calico-etcd.yaml



ETCD_CA=`cat /etc/kubernetes/pki/etcd/etcd-ca.pem | base64 | tr -d '\n'`
ETCD_CERT=`cat /etc/kubernetes/pki/etcd/etcd.pem | base64 | tr -d '\n'`
ETCD_KEY=`cat /etc/kubernetes/pki/etcd/etcd-key.pem | base64 | tr -d '\n'`
```

```
sed -i "s@# etcd-key: null@etcd-key: ${ETCD_KEY}@g; s@# etcd-cert: null@etcd-cert: ${ETCD_CERT}@g; s@# etcd-ca: null@etcd-ca: ${ETCD_CA}@g" calico-etcd.yaml


sed -i 's#etcd_ca: ""#etcd_ca: "/calico-secrets/etcd-ca"#g; s#etcd_cert: ""#etcd_cert: "/calico-secrets/etcd-cert"#g; s#etcd_key: "" #etcd_key: "/calico-secrets/etcd-key" #g' calico-etcd.yaml

# 更改此处为自己的pod网段
POD_SUBNET="172.168.0.0/12"

sed -i 's@# - name: CALICO_IPV4POOL_CIDR@- name: CALICO_IPV4POOL_CIDR@g; s@#   value: "192.168.0.0/16"@  value: '"${POD_SUBNET}"'@g' calico-etcd.yaml
```

12.2、apply

```
[root@k8s-master01 calico]# kubectl apply -f calico-etcd.yaml
# 执行结果
secret/calico-etcd-secrets created
configmap/calico-config created
clusterrole.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrolebinding.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrole.rbac.authorization.k8s.io/calico-node created
clusterrolebinding.rbac.authorization.k8s.io/calico-node created
daemonset.apps/calico-node created
serviceaccount/calico-node created
deployment.apps/calico-kube-controllers created
serviceaccount/calico-kube-controllers created
```

12.3、查看容器状态

如果容器状态异常可以使用kubectl describe 或者logs查看容器的日志

```
[root@k8s-master01 calico]# kubectl  get po -n kube-system
NAME                                       READY   STATUS     RESTARTS   AGE
calico-kube-controllers-5f6d4b864b-pq2qw   0/1     Pending    0          45s
calico-node-75blv                          0/1     Init:0/2   0          46s
calico-node-hw27b                          0/1     Init:0/2   0          46s
calico-node-k2wdf                          0/1     Init:0/2   0          46s
calico-node-l58lz                          0/1     Init:0/2   0          46s
calico-node-v2qlq                          0/1     Init:0/2   0          46s
coredns-867d46bfc6-8vzrk                   0/1     Pending    0          10m
```

十三、安装CoreDNS

13.1、安装对应版本（推荐）

master01操作

如果更改了k8s service的网段需要将coredns的serviceIP改成k8s service网段的第十个IP

```
cd /root/k8s-ha-install/

[root@k8s-master01 k8s-ha-install]# sed -i "s#10.96.0.10#10.96.0.10#g" CoreDNS/coredns.yaml
```

13.2、安装coredns

```
[root@k8s-master01 k8s-ha-install]# kubectl  create -f CoreDNS/coredns.yaml
# 执行结果
serviceaccount/coredns created
clusterrole.rbac.authorization.k8s.io/system:coredns created
clusterrolebinding.rbac.authorization.k8s.io/system:coredns created
configmap/coredns created
deployment.apps/coredns created
service/kube-dns created
```

13.3、安装最新版CoreDNS（不建议）

```
git clone https://github.com/coredns/deployment.git
cd deployment/kubernetes
# ./deploy.sh -s -i 10.96.0.10 | kubectl apply -f -
serviceaccount/coredns created
clusterrole.rbac.authorization.k8s.io/system:coredns created
clusterrolebinding.rbac.authorization.k8s.io/system:coredns created
configmap/coredns created
deployment.apps/coredns created
service/kube-dns created

# 查看状态
# kubectl get po -n kube-system -l k8s-app=kube-dns
NAME                       READY   STATUS    RESTARTS   AGE
coredns-85b4878f78-h29kh   1/1     Running   0          8h
```

十四、安装Metrics Server

在新版的Kubernetes中系统资源的采集均使用Metrics-server，可以通过Metrics采集节点和Pod的内存、磁盘、CPU和网络的使用率

14.1、安装metrics server

```
[root@k8s-master01 ~]# cd /root/k8s-ha-install/metrics-server-0.4.x/

[root@k8s-master01 metrics-server-0.4.x]# kubectl  create -f .

# 执行结果
serviceaccount/metrics-server created
clusterrole.rbac.authorization.k8s.io/system:aggregated-metrics-reader created
clusterrole.rbac.authorization.k8s.io/system:metrics-server created
rolebinding.rbac.authorization.k8s.io/metrics-server-auth-reader created
clusterrolebinding.rbac.authorization.k8s.io/metrics-server:system:auth-delegator
created
clusterrolebinding.rbac.authorization.k8s.io/system:metrics-server created
service/metrics-server created
deployment.apps/metrics-server created
apiservice.apiregistration.k8s.io/v1beta1.metrics.k8s.io created
```

14.2、等待metrics server启动然后查看状态

```
[root@k8s-master01 ~]# kubectl  top node
NAME           CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
k8s-master01   384m         19%    1110Mi          59%
k8s-master02   334m         16%    1086Mi          58%
k8s-master03   324m         16%    1043Mi          55%
k8s-node01     208m         10%    573Mi           30%
k8s-node02     180m         9%     534Mi           28%
```

十五、安装dashboard

Dashboard用于展示集群中的各类资源，同时也可以通过Dashboard实时查看Pod的日志和在容器中执行一些命令等。

15.1、安装指定版本dashboard

```
[root@k8s-master01 ~]# cd /root/k8s-ha-install/dashboard/

[root@k8s-master01 dashboard]# kubectl  create -f .
serviceaccount/admin-user created
clusterrolebinding.rbac.authorization.k8s.io/admin-user created
namespace/kubernetes-dashboard created
serviceaccount/kubernetes-dashboard created
service/kubernetes-dashboard created
secret/kubernetes-dashboard-certs created
secret/kubernetes-dashboard-csrf created
secret/kubernetes-dashboard-key-holder created
configmap/kubernetes-dashboard-settings created
role.rbac.authorization.k8s.io/kubernetes-dashboard created
```

```
clusterrole.rbac.authorization.k8s.io/kubernetes-dashboard created
rolebinding.rbac.authorization.k8s.io/kubernetes-dashboard created
clusterrolebinding.rbac.authorization.k8s.io/kubernetes-dashboard created
deployment.apps/kubernetes-dashboard created
service/dashboard-metrics-scraper created
deployment.apps/dashboard-metrics-scraper created
```

15.2、安装最新版

```
# 官方GitHub地址：https://github.com/kubernetes/dashboard
# 可以在官方dashboard查看到最新版dashboard

kubectl apply -f
https://raw.githubusercontent.com/kubernetes/dashboard/v2.0.3/aio/deploy/recommended.yaml

# 创建管理员用户vim admin.yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: admin-user
  namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: admin-user
  annotations:
    rbac.authorization.kubernetes.io/autoupdate: "true"
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: admin-user
  namespace: kube-system

# 安装
kubectl apply -f admin.yaml -n kube-system
```

15.3、登录dashboard

```
# 更改dashboard的svc为NodePort
[root@k8s-master01 ~]# kubectl edit svc kubernetes-dashboard -n kubernetes-dashboard
将ClusterIP更改为NodePort

# 查看端口号
[root@k8s-master01 ~]# kubectl get svc kubernetes-dashboard -n kubernetes-dashboard
NAME                   TYPE       CLUSTER-IP     EXTERNAL-IP   PORT(S)         AGE
kubernetes-dashboard   NodePort   10.96.77.112   <none>        443:30902/TCP   4m10s

# 根据自己的实例端口号，通过任意安装了kube-proxy的宿主机或者VIP的IP+端口即可访问到dashboard
页面访问：https://10.4.7.236:30902/
```

15.4、查看token值

```
[root@k8s-master01 ~]# kubectl -n kube-system describe secret $(kubectl -n kube-system
get secret | grep admin-user | awk '{print $1}')
```

十六、集群验证

16.1、安装busybox（master01上）

```
cat<<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: busybox
  namespace: default
spec:
  containers:
  - name: busybox
    image: busybox:1.28
    command:
      - sleep
      - "3600"
    imagePullPolicy: IfNotPresent
  restartPolicy: Always
EOF
```

16.2、验证步骤（matser01上）

```
1.  Pod必须能解析Service
2.  Pod必须能解析跨namespace的Service
3.  每个节点都必须要能访问Kubernetes的kubernetes svc 443和kube-dns的service 53
4.  Pod和Pod之间要能通
  a)  同namespace能通信
  b)  跨namespace能通信
  c)  跨机器能通信
```

16.3、步骤演示（matser01上）

```
# 首先查看po是否安装成功
[root@k8s-master01 ~]# kubectl get po
NAME      READY   STATUS    RESTARTS   AGE
busybox   1/1     Running   0          3m11s

# 查看svc是否正常
[root@k8s-master01 ~]# kubectl get svc
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   163m

# 查看Pod是否能能解析Service
[root@k8s-master01 ~]# kubectl exec  busybox -n default -- nslookup kubernetes
Server:    10.96.0.10
Address 1: 10.96.0.10 kube-dns.kube-system.svc.cluster.local

Name:      kubernetes
Address 1: 10.96.0.1 kubernetes.default.svc.cluster.local

# 查看Pod是否能解析跨namespace的Service
[root@k8s-master01 ~]# kubectl exec  busybox -n default -- nslookup kube-dns.kube-system
Server:    10.96.0.10
Address 1: 10.96.0.10 kube-dns.kube-system.svc.cluster.local

Name:      kube-dns.kube-system
Address 1: 10.96.0.10 kube-dns.kube-system.svc.cluster.local

# 跟我以上结果一致就成功了
```

16.4、使用telnet命令验证

```
# 所有节点安装telnet命令,有的话忽略
yum install -y telnet

# 所有机器 10.96.0.1  443  kubernetes svc 443
# 所有机器 10.96.0.10 53   kube-dns的service 53
# 不会自动断开就是成功了
telnet 10.96.0.1 443
telnet 10.96.0.10 53

Trying 10.96.0.1...
Connected to 10.96.0.1.
Escape character is '^]'.
```

16.5、使用curl命令验证（所有机器）

```
[root@k8s-master01 ~]# curl 10.96.0.10:53
curl: (52) Empty reply from server
```

16.6、容器验证（master01上）

```
[root@k8s-master01 ~]# kubectl get po -n kube-system
NAME                                       READY   STATUS    RESTARTS   AGE
calico-kube-controllers-5f6d4b864b-pq2qw   1/1     Running   0          62m
calico-node-75blv                          1/1     Running   0          62m
calico-node-hw27b                          1/1     Running   0          62m
calico-node-k2wdf                          1/1     Running   0          62m
calico-node-l58lz                          1/1     Running   0          62m
calico-node-v2qlq                          1/1     Running   0          62m
coredns-867d46bfc6-8vzrk                   1/1     Running   0          72m
metrics-server-595f65d8d5-kgn8c            1/1     Running   0          60m

[root@k8s-master01 ~]# kubectl get po -n kube-system -owide
NAME                                       READY   STATUS    RESTARTS   AGE   IP
       NODE           NOMINATED NODE   READINESS GATES
calico-kube-controllers-5f6d4b864b-pq2qw   1/1     Running   0          63m
10.4.7.107   k8s-master01   <none>           <none>
calico-node-75blv                          1/1     Running   0          63m
10.4.7.110   k8s-node01     <none>           <none>
calico-node-hw27b                          1/1     Running   0          63m
10.4.7.108   k8s-master02   <none>           <none>
calico-node-k2wdf                          1/1     Running   0          63m
10.4.7.107   k8s-master01   <none>           <none>
calico-node-l58lz                          1/1     Running   0          63m
10.4.7.109   k8s-master03   <none>           <none>
calico-node-v2qlq                          1/1     Running   0          63m
10.4.7.111   k8s-node02     <none>           <none>
coredns-867d46bfc6-8vzrk                   1/1     Running   0          73m
172.161.125.2   k8s-node01     <none>           <none>
```

```
metrics-server-595f65d8d5-kgn8c           1/1     Running   0          62m
172.161.125.1   k8s-node01     <none>           <none>

# 能进去就ok
[root@k8s-master01 ~]# kubectl exec -it calico-node-v2qlq -n  kube-system  -- sh
sh-4.4#

# 进入node01，然后能ping通node02就行
[root@k8s-master01 ~]# kubectl exec -it calico-node-v2qlq -n  kube-system  -- bash
[root@k8s-node02 /]# ping 10.4.7.111
PING 10.4.7.111 (10.4.7.111) 56(84) bytes of data.
64 bytes from 10.4.7.111: icmp_seq=1 ttl=64 time=0.123 ms
64 bytes from 10.4.7.111: icmp_seq=2 ttl=64 time=0.090 ms
^C
--- 10.4.7.111 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 46ms
rtt min/avg/max/mdev = 0.090/0.106/0.123/0.019 ms
```

十七、生产环境关键性配置

17.1、关键性配置请参考视频，**不要直接配置！**

```
# 所有节点都改
vim /etc/docker/daemon.json
{  "registry-mirrors": [
    "https://registry.docker-cn.com",
    "http://hub-mirror.c.163.com",
    "https://docker.mirrors.ustc.edu.cn"
  ],
 "exec-opts": ["native.cgroupdriver=systemd"],
 "max-concurrent-downloads": 10,
 "max-concurrent-uploads": 5,
 "log-opts": {    "max-size": "300m",    "max-file": "2"  },
 "live-restore": true
 }

max-concurrent-downloads # 下载并发数
max-concurrent-uploads   # 上传并发数
max-size         # 日志文件最大到多少切割 （此处是300m）
max-file         # 日志文件保留个数 （此处是2个）
live-restore       # 开启这个参数，重启docker不会影响上面的参数

# 所有节点改完重启docker
systemctl daemon-reload && systemctl restart docker


vim /usr/lib/systemd/system/kube-controller-manager.service
 # 找个位置加上,在三个master节点
 --experimental-cluster-signing-duration=876000h0m0s \
 # 改完重启
systemctl daemon-reload && systemctl restart kube-controller-manager


# 所有节点，更换成以下的配置文件
[root@k8s-node02 ~]# cat /etc/systemd/system/kubelet.service.d/10-kubelet.conf

[Service]
Environment="KUBELET_KUBECONFIG_ARGS=--bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.kubeconfig --kubeconfig=/etc/kubernetes/kubelet.kubeconfig"
Environment="KUBELET_SYSTEM_ARGS=--network-plugin=cni --cni-conf-dir=/etc/cni/net.d --cni-bin-dir=/opt/cni/bin"
Environment="KUBELET_CONFIG_ARGS=--config=/etc/kubernetes/kubelet-conf.yml --pod-infra-container-image=registry.cn-hangzhou.aliyuncs.com/google_containers/pause-amd64:3.2"
Environment="KUBELET_EXTRA_ARGS=--node-labels=node.kubernetes.io/node='' --tls-cipher-suites=TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384    --image-pull-progress-deadline=30m "
ExecStart=
ExecStart=/usr/local/bin/kubelet $KUBELET_KUBECONFIG_ARGS $KUBELET_CONFIG_ARGS $KUBELET_SYSTEM_ARGS $KUBELET_EXTRA_ARGS
# 所有节点、添加如下配置-----  注意请更具生成环境配置
vim /etc/kubernetes/kubelet-conf.yml

rotateServerCertificates: true
allowedUnsafeSysctls:
 - "net.core*"
 - "net.ipv4.*"
kubeReserved:
  cpu: "10m"
  memory: 10Mi
  ephemeral-storage: 10Mi
systemReserved:
  cpu: "1"
  memory: 20Mi
  ephemeral-storage: 1Gi


# 改完重启
systemctl daemon-reload && systemctl restart kubelet

# 查看日志没报错就行
[root@k8s-master01 ~]# tail -f /var/log/messages
# 角色名字更改
[root@k8s-master01 ~]# kubectl label node k8s-master01 node-role.kubernetes.io/matser=''
node/k8s-master01 labeled
[root@k8s-master01 ~]# kubectl get node
NAME           STATUS   ROLES    AGE    VERSION
```

```
k8s-master01   Ready    matser   129m   v1.20.0   # 成功更改
k8s-master02   Ready    <none>   129m   v1.20.0
k8s-master03   Ready    <none>   129m   v1.20.0
k8s-node01     Ready    <none>   129m   v1.20.0
k8s-node02     Ready    <none>   129m   v1.20.0
```

十八、安装总结

```
1、  kubeadm
2、  二进制
3、  自动化安装
  a)  Ansible
    i.  Master节点安装不需要写自动化。
    ii. 添加Node节点，playbook。
4、  安装需要注意的细节
  a)  上面的细节配置
  b)  生产环境中etcd一定要和系统盘分开，一定要用ssd硬盘。
  c)  Docker数据盘也要和系统盘分开，有条件的话可以使用ssd硬盘
```

十九、Bootstrapping

Bootstrapping CSR申请和证书颁发原理

1.kubelet启动

2.kubele t查找 kubelet.kubeconfig 文件，假设没有这个文件

3.kubelet 会查找本地 bootstrap-kubelet.kubeconfig

4.kubelet 读取 bootstrap.kubeconfig 文件，检索apiserver 的 url 和一个token

5.kubelet 链接 apiserver, 使用这个token 进行认证

a) apiserver 会识别tokenid， apiserver 会查找该 tokenid 对应的 bootstrap 的要给 secret

b) 找这个 secret 中的一个字段， apiserver 把这个 token 识别成一个 username，名称是 system:bootstrap:，属于system:bootstrappers这个组，这个组具有申请csr的权限, 该组的权限绑定在一个叫system:node-bootstrapper的 clusterrole； clusterrole k8s 集群级别的权限控制，它作用整个k8s集群。

c) CSR： 相当于申请表，可以拿着这个申请表去申请我们的证书。

6.经过上面的认证，kubelet 就有了一个创建和检索 CSR的权限。

7.kubelet 为自己创建一个CSR，名称为 kubernetes.io/kube-apiserver-clinet-kubelet

8.CSR 被允许有两种方式：

a) k8s 管理员使用 kubectl 手动的颁发证书

b) 如果配置了相关权限，kube-controller-manager 会自动同意。

9. CSR 被同意后， controller-manager 创建 kubelet的证书文件
10. controller-manager 将证书更新至 csr的 status字段
11. kubelet 从 apiserver 获取证书
12. kubelet 从获取到的 ey 和证书文件 创建 kubelet.kubeconfig
13. kubelet 启动完成并正常工作
14. 可选：如果配置了自动续期，kubelet 会在证书文件过期的时候利用之前的 kubeconfig 文件去申请一个新的证书，相当于续约。
15. 新的证书被同意或签发，取决于我们的配置。