# 搭建Kubernetes高可用集群

此文以Kubernetes 1.18.5版本为例！

如未指定，下述命令在所有节点执行！

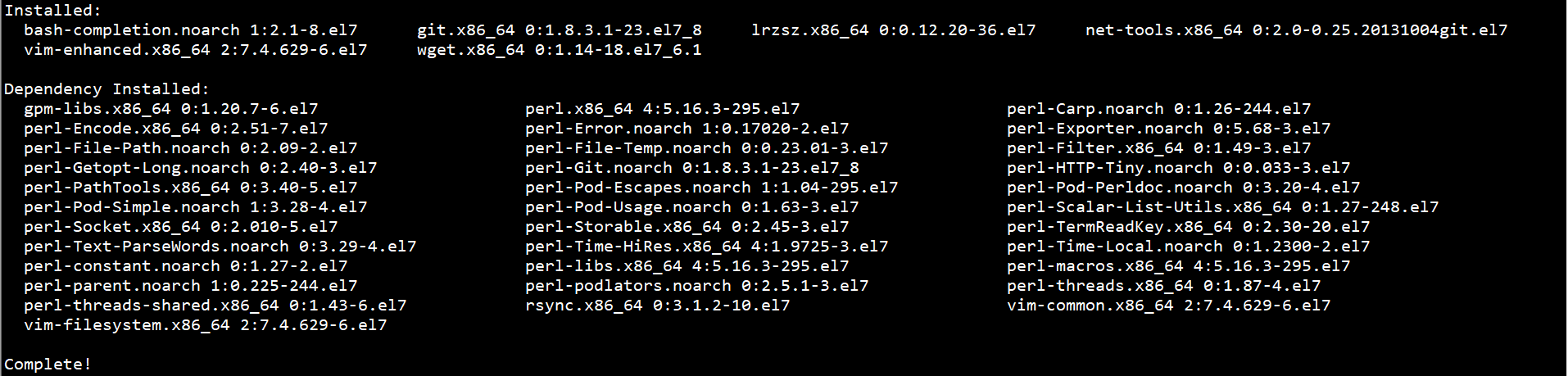
## 一、系统资源规划

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **节点名称** | **系统名称** | **CPU/内存** | **网卡** | **磁盘** | **IP地址** | **OS** |
| Master1 | master1 | 4C/8G | ens33 | 64G | 192.168.0.11 | CentOS7 |
| Master2 | master2 | 4C/8G | ens33 | 64G | 192.168.0.12 | CentOS7 |
| Master3 | master3 | 4C/8G | ens33 | 64G | 192.168.0.13 | CentOS7 |
| Worker1 | worker1 | 4C/8G | ens33 | 64G | 192.168.0.21 | CentOS7 |
| Worker2 | worker2 | 4C/8G | ens33 | 64G | 192.168.0.22 | CentOS7 |
| Worker3 | worker3 | 4C/8G | ens33 | 64G | 192.168.0.23 | CentOS7 |

## 二、系统软件安装与设置

### 1、安装基本软件

yum -y install vim git lrzsz wget net-tools bash-completion



### 2、设置名称解析

echo 192.168.0.11 master1 >> /etc/hosts

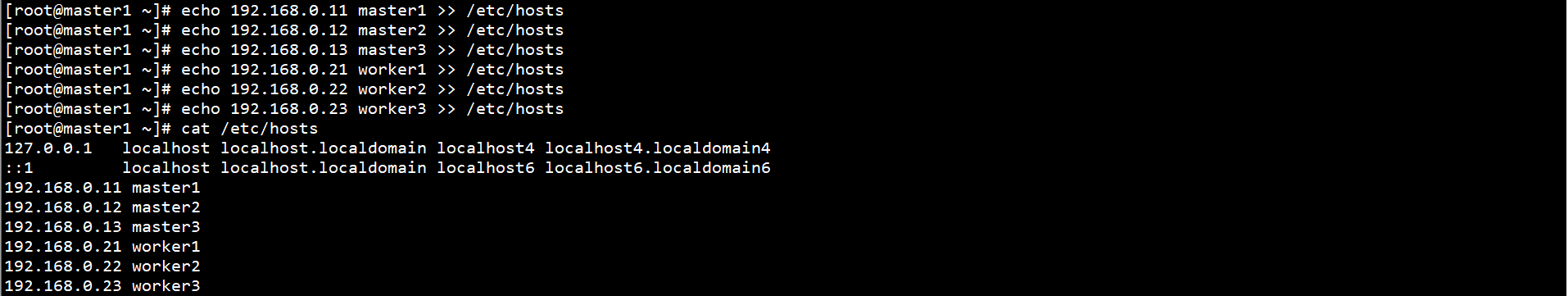
echo 192.168.0.12 master2 >> /etc/hosts

echo 192.168.0.13 master3 >> /etc/hosts

echo 192.168.0.21 worker1 >> /etc/hosts

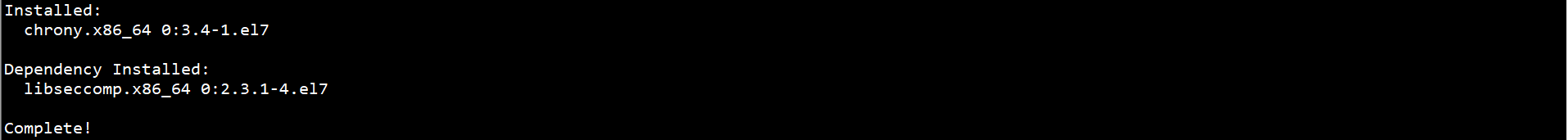
echo 192.168.0.22 worker2 >> /etc/hosts

echo 192.168.0.23 worker3 >> /etc/hosts



### 3、设置NTP

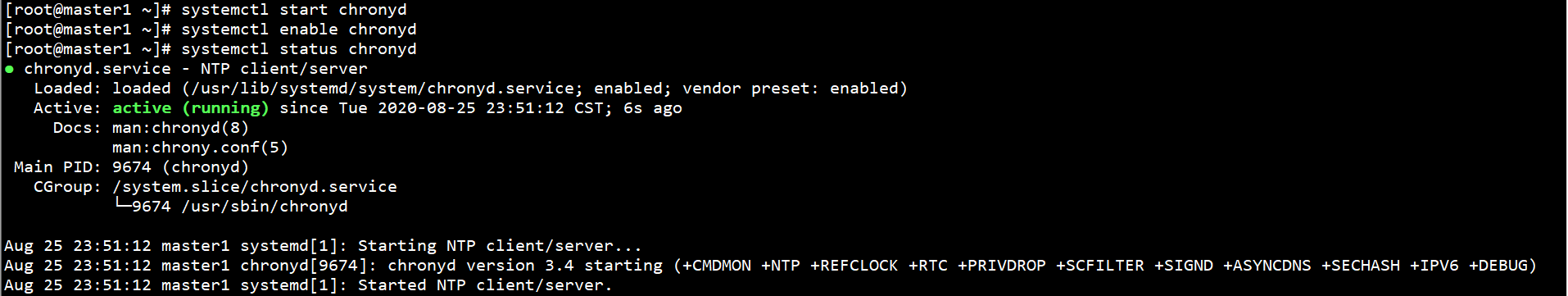
yum -y install chrony



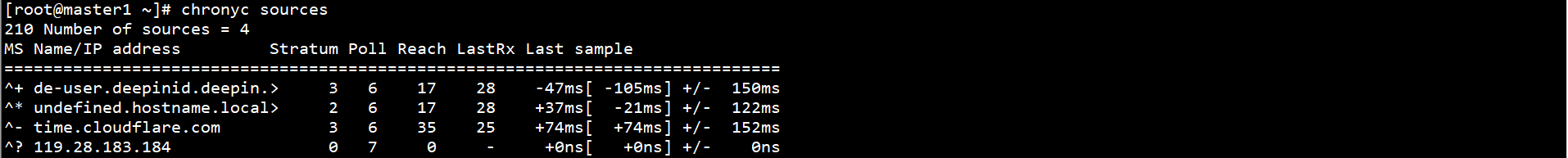
systemctl start chronyd

systemctl enable chronyd

systemctl status chronyd



chronyc sources



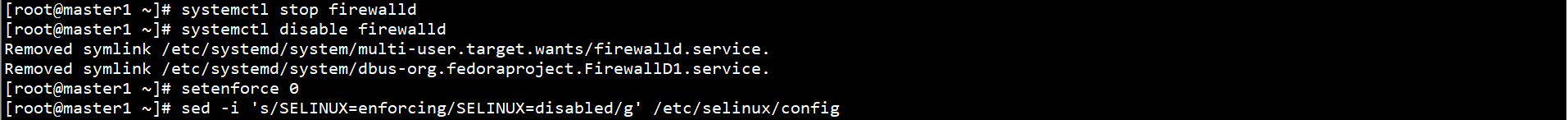
### 4、设置SELinux、防火墙

systemctl stop firewalld

systemctl disable firewalld

setenforce 0

sed -i 's/SELINUX=enforcing/SELINUX=disabled/g' /etc/selinux/config



### 5、设置网桥

配置L2网桥在转发包时会被iptables的FORWARD规则所过滤，CNI插件需要该配置

创建/etc/sysctl.d/k8s.conf文件，添加如下内容：

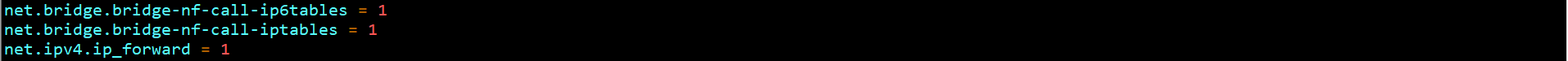
cat > /etc/sysctl.d/k8s.conf << EOF

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

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

net.ipv4.ip\_forward = 1

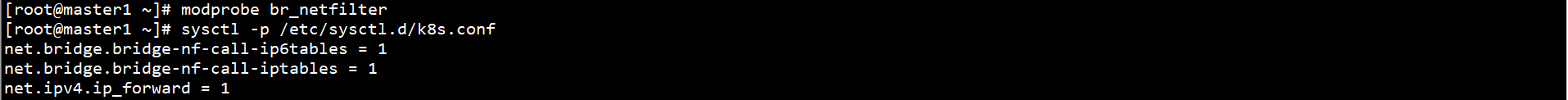
EOF



执行命令，使修改生效：

modprobe br\_netfilter

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



### 6、设置swap

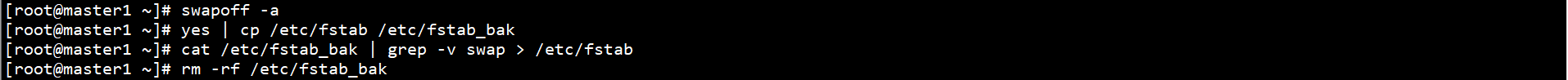
关闭系统swap分区：

swapoff -a

yes | cp /etc/fstab /etc/fstab\_bak

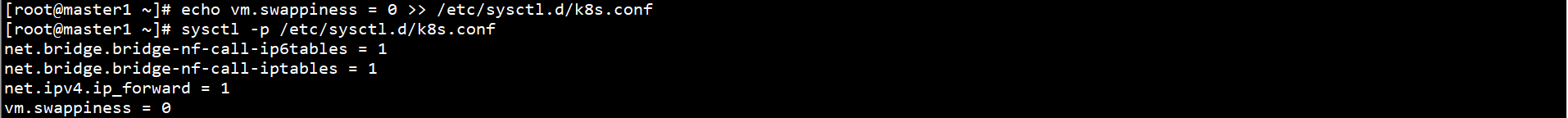
cat /etc/fstab\_bak | grep -v swap > /etc/fstab

rm -rf /etc/fstab\_bak



echo vm.swappiness = 0 >> /etc/sysctl.d/k8s.conf

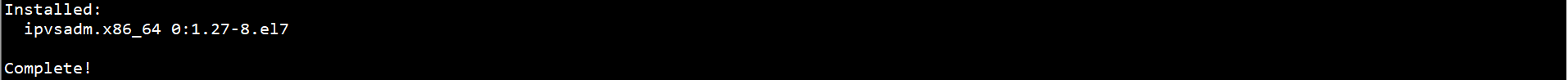
sysctl -p /etc/sysctl.d/k8s.conf



### 7、设置ipvs

安装ipvsadm ipset：

yum -y install ipvsadm ipset



创建ipvs设置脚本：

cat > /etc/sysconfig/modules/ipvs.modules << EOF

#!/bin/bash

modprobe -- ip\_vs

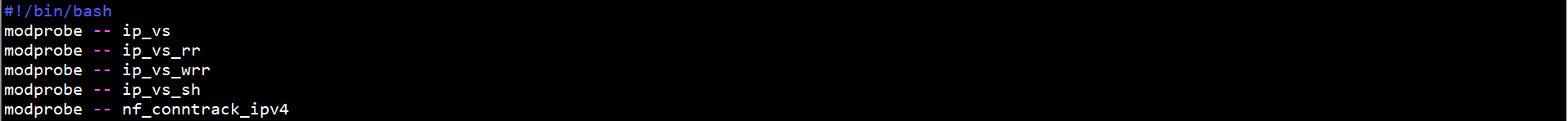
modprobe -- ip\_vs\_rr

modprobe -- ip\_vs\_wrr

modprobe -- ip\_vs\_sh

modprobe -- nf\_conntrack\_ipv4

EOF

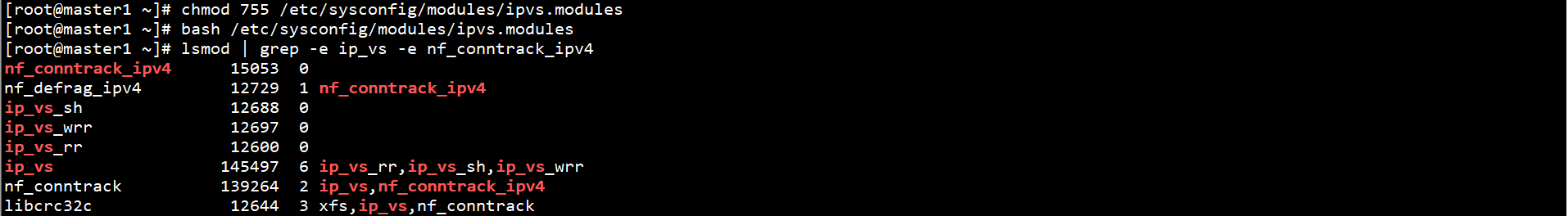


执行脚本，验证修改结果：

chmod 755 /etc/sysconfig/modules/ipvs.modules

bash /etc/sysconfig/modules/ipvs.modules

lsmod | grep -e ip\_vs -e nf\_conntrack\_ipv4

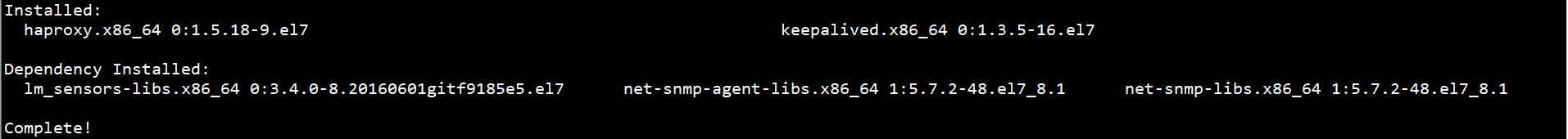


## 三、负载均衡配置

### 1、安装HAProxy和Keepalived

在所有Master节点上安装HAProxy和Keepalived：

yum -y install haproxy keepalived



在所有Master节点上创建HAProxy配置文件：

cat > /etc/haproxy/haproxy.cfg << EOF

global

log 127.0.0.1 local2

chroot /var/lib/haproxy

pidfile /var/run/haproxy.pid

maxconn 4000

user haproxy

group haproxy

daemon

stats socket /var/lib/haproxy/stats

defaults

mode tcp

log global

option tcplog

option dontlognull

option redispatch

retries 3

timeout queue 1m

timeout connect 10s

timeout client 1m

timeout server 1m

timeout check 10s

maxconn 3000

frontend k8s\_https \*:8443

mode tcp

maxconn 2000

default\_backend https\_sri

backend https\_sri

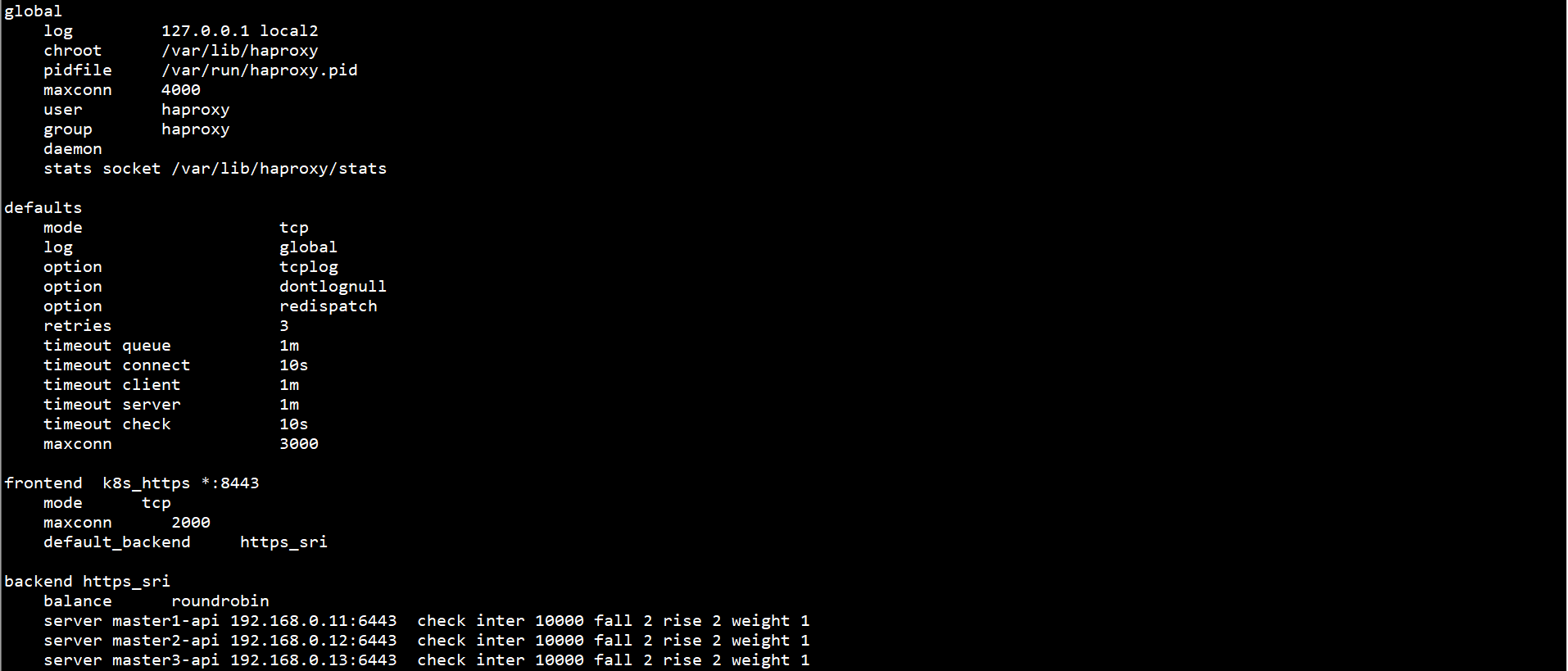
balance roundrobin

server master1-api 192.168.0.11:6443 check inter 10000 fall 2 rise 2 weight 1

server master2-api 192.168.0.12:6443 check inter 10000 fall 2 rise 2 weight 1

server master3-api 192.168.0.13:6443 check inter 10000 fall 2 rise 2 weight 1

EOF



在Master1节点上创建Keepalived配置文件：

cat > /etc/keepalived/keepalived.conf << EOF

global\_defs {

router\_id LVS\_DEVEL

}

vrrp\_script check\_haproxy {

script "/etc/keepalived/check\_haproxy.sh"

interval 3000

}

vrrp\_instance VI\_1 {

state Master

interface ens33

virtual\_router\_id 80

priority 100

advert\_int 1

authentication {

auth\_type PASS

auth\_pass 111111

}

virtual\_ipaddress {

192.168.0.10/24

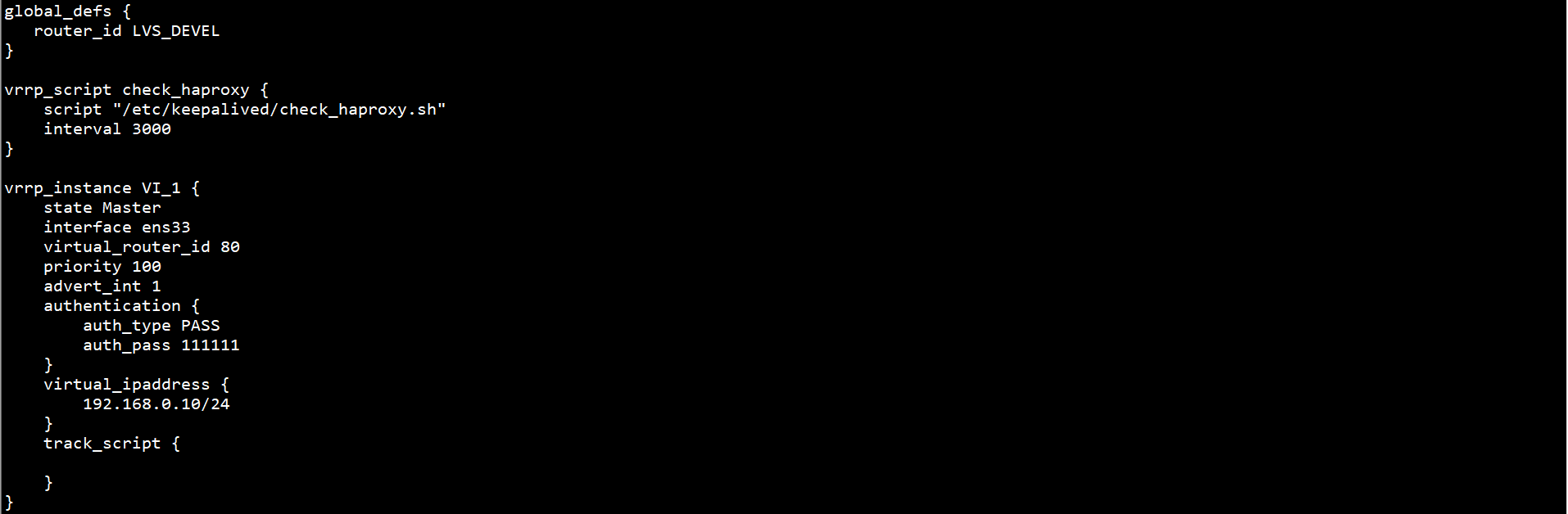
}

track\_script {

}

}

EOF



在Master2节点上创建Keepalived配置文件：

cat > /etc/keepalived/keepalived.conf << EOF

global\_defs {

router\_id LVS\_DEVEL

}

vrrp\_script check\_haproxy {

script "/etc/keepalived/check\_haproxy.sh"

interval 3000

}

vrrp\_instance VI\_1 {

state Slave

interface ens33

virtual\_router\_id 80

priority 50

advert\_int 1

authentication {

auth\_type PASS

auth\_pass 111111

}

virtual\_ipaddress {

192.168.0.10/24

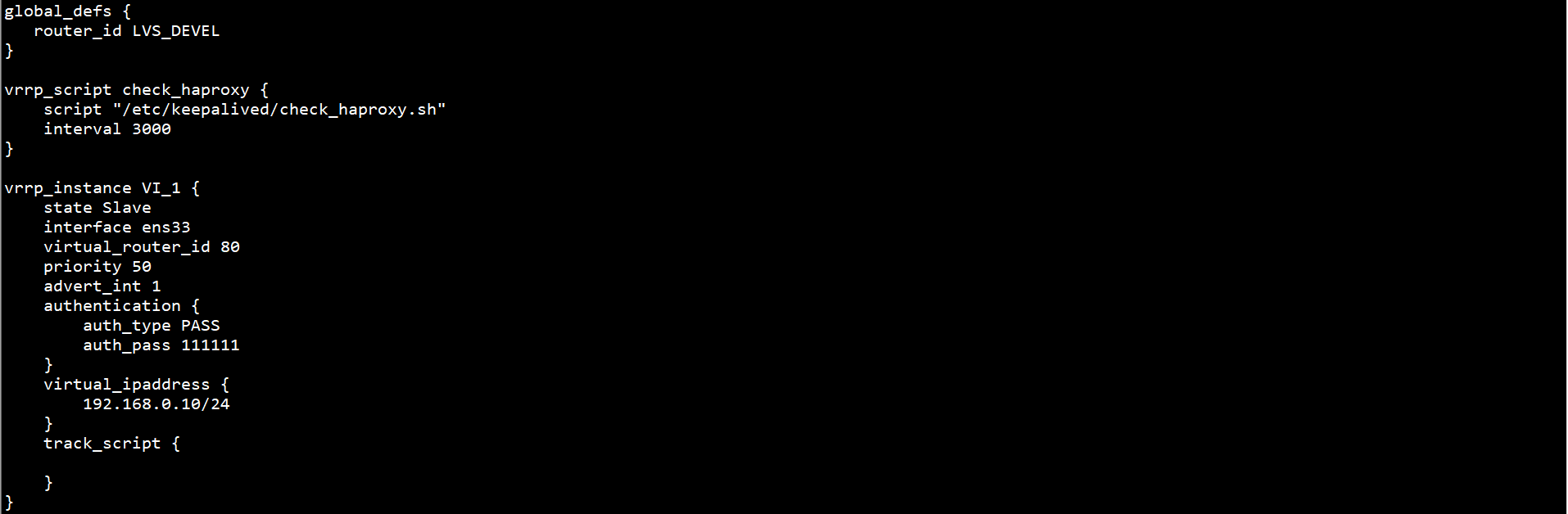
}

track\_script {

}

}

EOF



在Master3节点上创建Keepalived配置文件：

cat > /etc/keepalived/keepalived.conf << EOF

global\_defs {

router\_id LVS\_DEVEL

}

vrrp\_script check\_haproxy {

script "/etc/keepalived/check\_haproxy.sh"

interval 3000

}

vrrp\_instance VI\_1 {

state Slave

interface ens33

virtual\_router\_id 80

priority 30

advert\_int 1

authentication {

auth\_type PASS

auth\_pass 111111

}

virtual\_ipaddress {

192.168.0.10/24

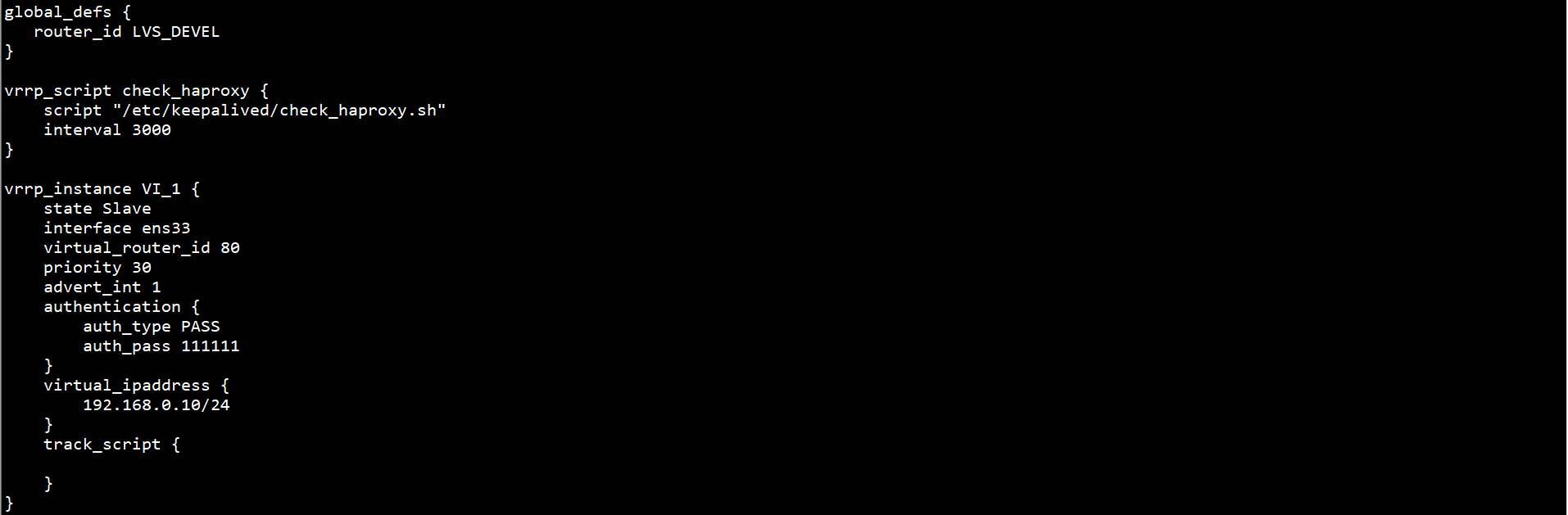
}

track\_script {

}

}

EOF



在所有Master节点上创建HAProxy检查脚本：

vim /etc/keepalived/check\_haproxy.sh

#!/bin/bash

if [ `ps -C haproxy --no-header | wc -l` == 0 ]; then

systemctl start haproxy

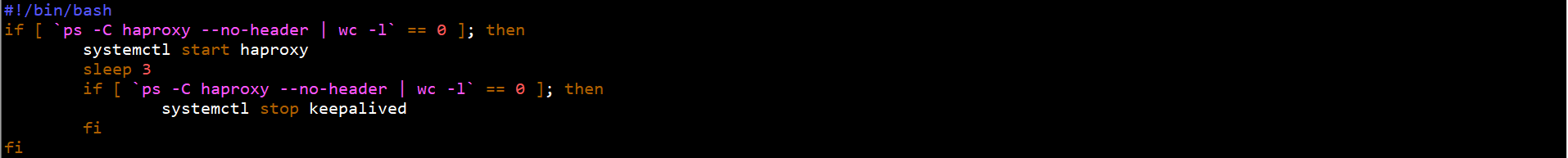
sleep 3

if [ `ps -C haproxy --no-header | wc -l` == 0 ]; then

systemctl stop keepalived

fi

fi



chmod +x /etc/keepalived/check\_haproxy.sh

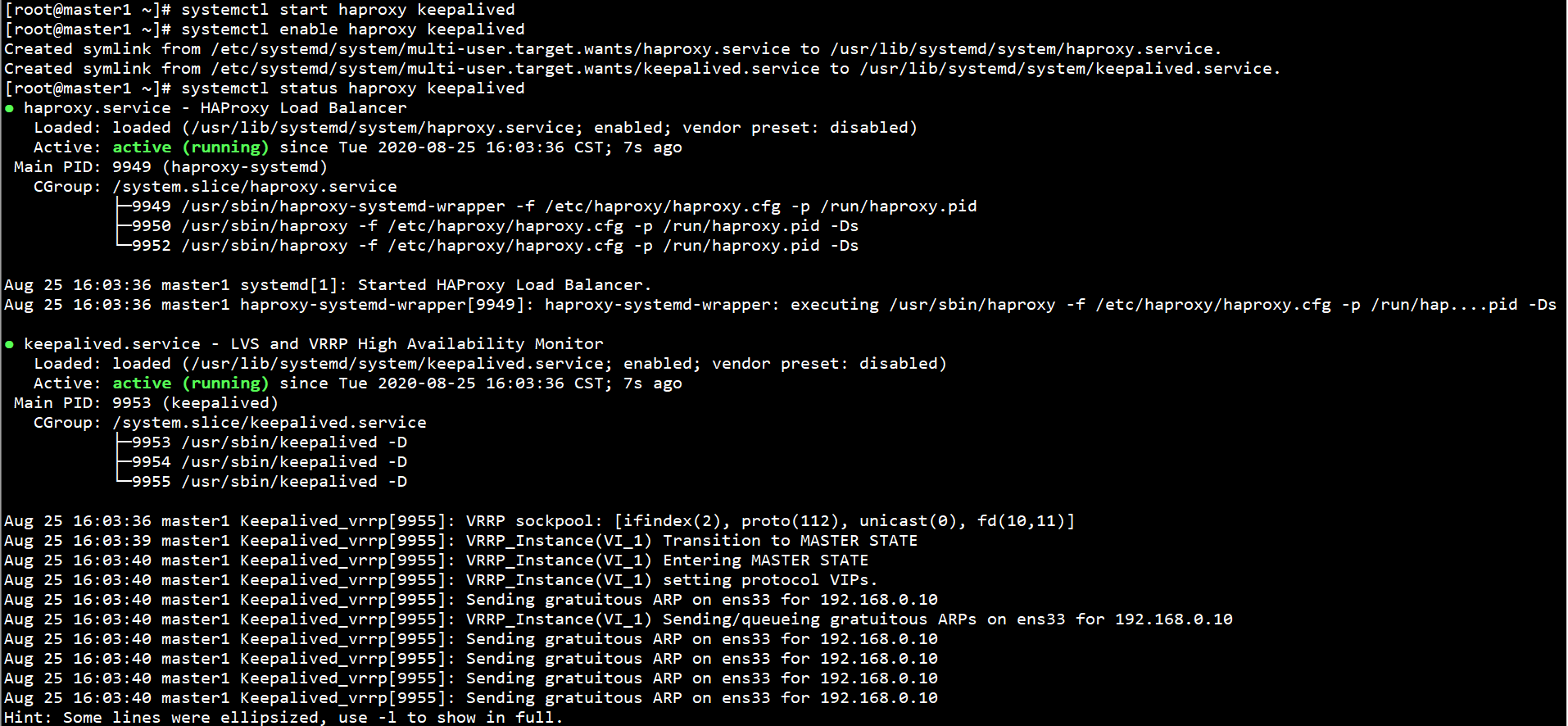


启动HAProxy和Keepalived，并设置自启动：

systemctl start haproxy keepalived

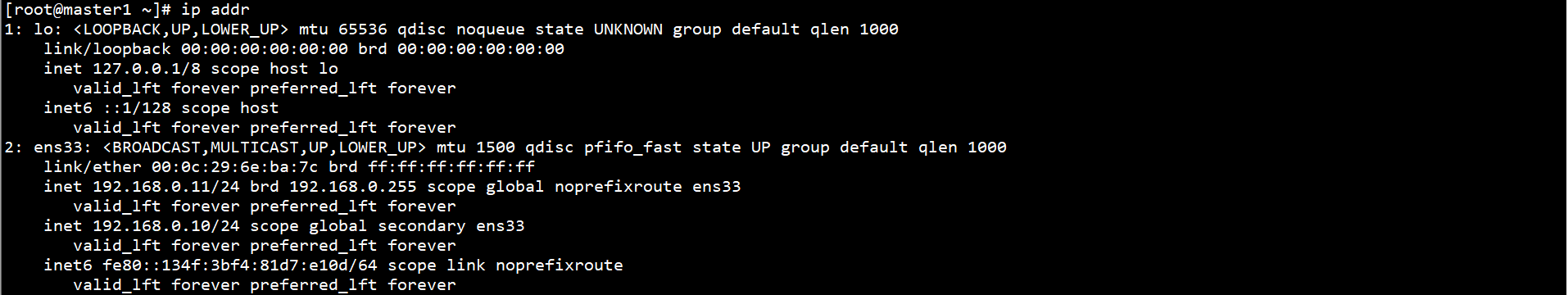
systemctl enable haproxy keepalived

systemctl status haproxy keepalived



查看keepalived工作状态：

ip addr



在ens33网卡绑定了192.168.0.10虚拟IP

## 四、Kubernetes集群配置

### 1、部署证书生成工具

在Master1节点上下载证书生成工具：

wget https://pkg.cfssl.org/R1.2/cfssl\_linux-amd64

wget https://pkg.cfssl.org/R1.2/cfssljson\_linux-amd64

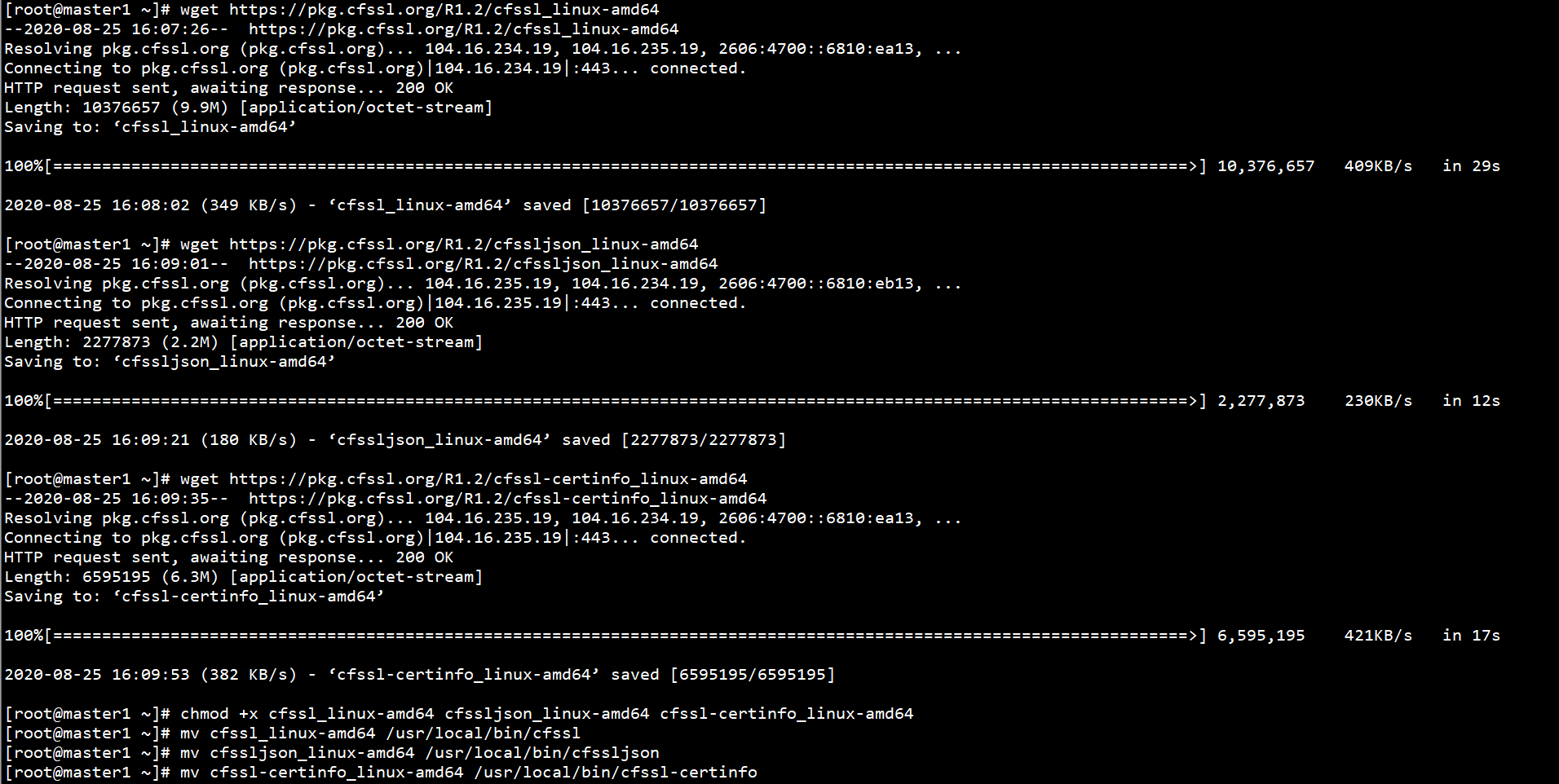
wget https://pkg.cfssl.org/R1.2/cfssl-certinfo\_linux-amd64

chmod +x cfssl\_linux-amd64 cfssljson\_linux-amd64 cfssl-certinfo\_linux-amd64

mv cfssl\_linux-amd64 /usr/local/bin/cfssl

mv cfssljson\_linux-amd64 /usr/local/bin/cfssljson

mv cfssl-certinfo\_linux-amd64 /usr/local/bin/cfssl-certinfo



### 2、部署ETCD集群

在所有Master节点上创建ETCD证书目录：

mkdir -p /etc/etcd/ssl/



在Master1节点上创建自签证书颁发机构CA：

cd /etc/etcd/ssl/



cat > ca-config.json << EOF

{

"signing": {

"default": {

"expiry": "87600h"

},

"profiles": {

"www": {

"expiry": "87600h",

"usages": [

"signing",

"key encipherment",

"server auth",

"client auth"

]

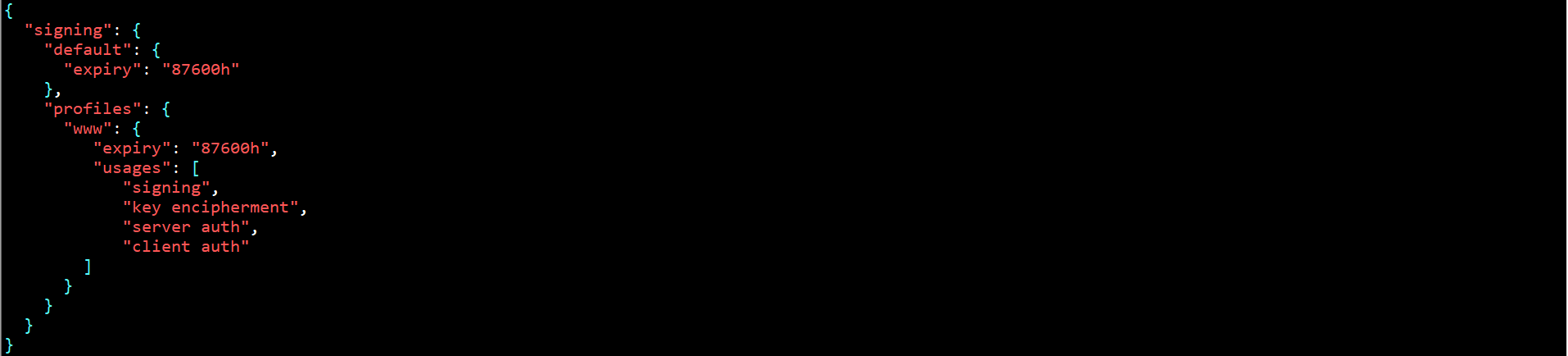
}

}

}

}

EOF



cat > ca-csr.json << EOF

{

"CN": "etcd CA",

"key": {

"algo": "rsa",

"size": 2048

},

"names": [

{

"C": "CN",

"L": "Beijing",

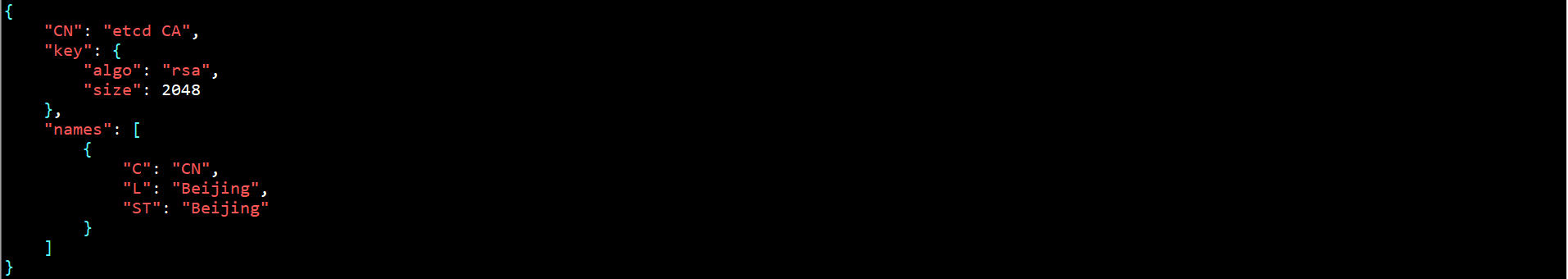
"ST": "Beijing"

}

]

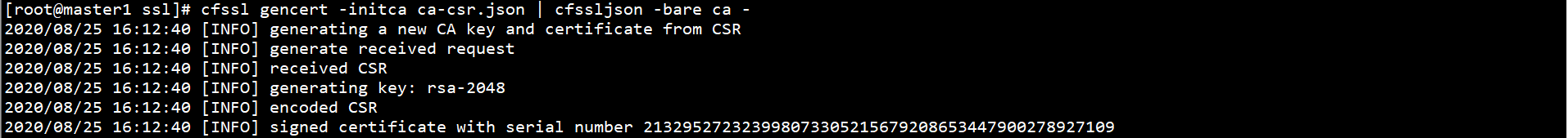
}

EOF

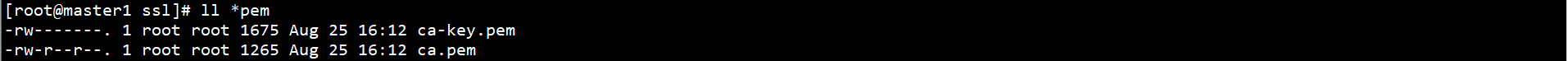


在Master1节点上生成证书：

cfssl gencert -initca ca-csr.json | cfssljson -bare ca -



ll \*pem



在Master1节点上创建ETCD HTTPS证书申请文件：

cd /etc/etcd/ssl/



cat > server-csr.json << EOF

{

"CN": "etcd",

"hosts": [

"192.168.0.11",

"192.168.0.12",

"192.168.0.13"

],

"key": {

"algo": "rsa",

"size": 2048

},

"names": [

{

"C": "CN",

"L": "BeiJing",

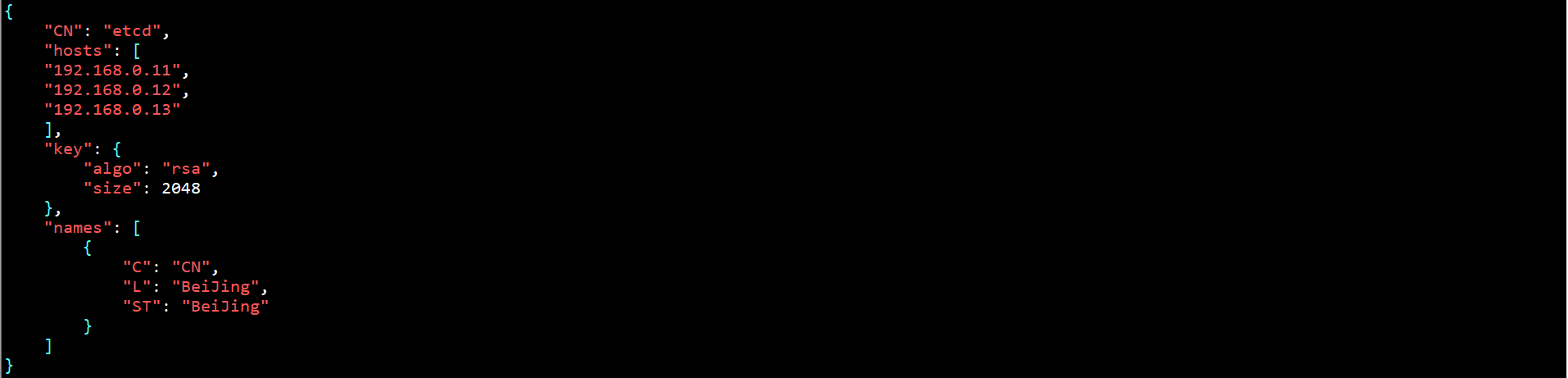
"ST": "BeiJing"

}

]

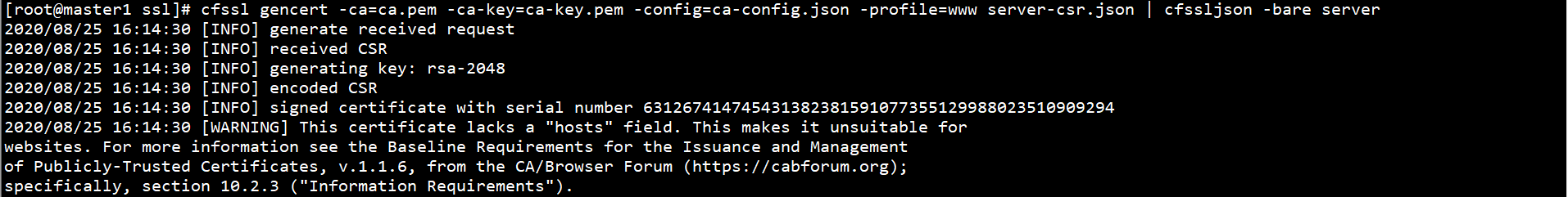
}

EOF

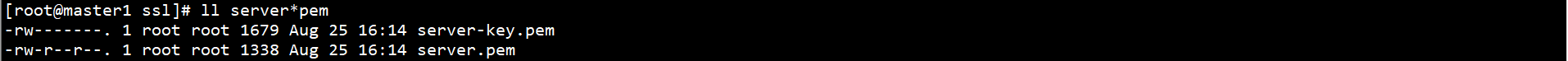


在Master1节点上生成证书：

cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=www server-csr.json | cfssljson -bare server



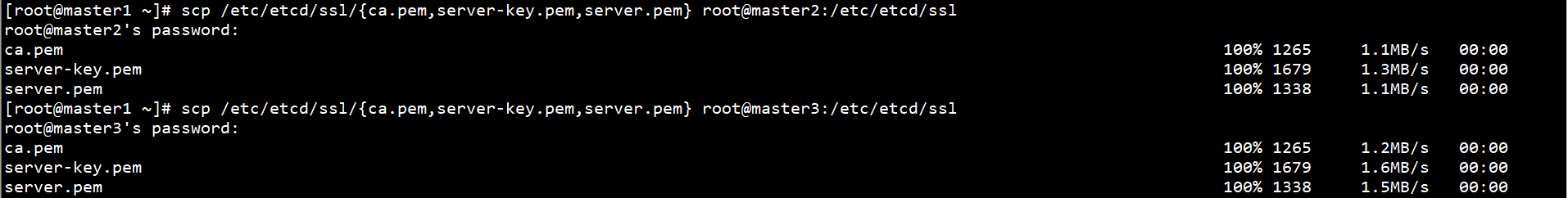
ll server\*pem



将Master1上所有生成的证书拷贝至其余Master节点：

scp /etc/etcd/ssl/{ca.pem,server-key.pem,server.pem} root@master2:/etc/etcd/ssl

scp /etc/etcd/ssl/{ca.pem,server-key.pem,server.pem} root@master3:/etc/etcd/ssl



下载ETCD二进制文件：

参考地址：<https://github.com/etcd-io/etcd/releases>

下载地址：[https://github.com/etcd-io/etcd/releases/download/v3.4.13/etcd-v3.4.13-linux-amd64.tar.gz](https://github.com/etcd-io/etcd/releases/download/v3.4.9/etcd-v3.4.9-linux-amd64.tar.gz)

在所有Master节点上创建ETCD配置文件目录：

mkdir -p /etc/etcd/cfg/



在所有Master节点上解压ETCD二进制文件至系统目录：

tar -xf /root/etcd-v3.4.13-linux-amd64.tar.gz -C /root/

mv /root/etcd-v3.4.13-linux-amd64/{etcd,etcdctl} /usr/local/bin/



在Master节点上创建ETCD配置文件：

cat > /etc/etcd/cfg/etcd.conf << EOF

#[Member]

ETCD\_NAME="etcd-1"

ETCD\_DATA\_DIR="/var/lib/etcd/default.etcd"

ETCD\_LISTEN\_PEER\_URLS="https://192.168.0.11:2380"

ETCD\_LISTEN\_CLIENT\_URLS="https://192.168.0.11:2379"

#[Clustering]

ETCD\_INITIAL\_ADVERTISE\_PEER\_URLS="https://192.168.0.11:2380"

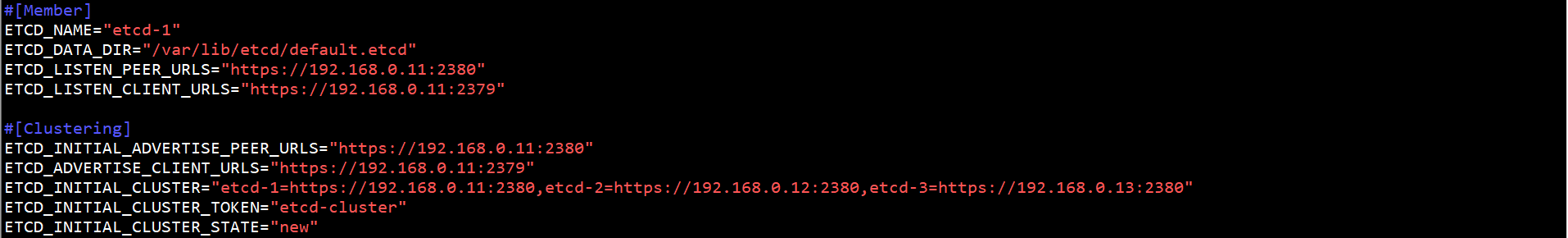
ETCD\_ADVERTISE\_CLIENT\_URLS="https://192.168.0.11:2379"

ETCD\_INITIAL\_CLUSTER="etcd-1=https://192.168.0.11:2380,etcd-2=https://192.168.0.12:2380,etcd-3=https://192.168.0.13:2380"

ETCD\_INITIAL\_CLUSTER\_TOKEN="etcd-cluster"

ETCD\_INITIAL\_CLUSTER\_STATE="new"

EOF



标红部分按节点实际名称及地址修改。

ETCD配置文件参数说明：

ETCD\_NAME：节点名称，集群中唯一

ETCD\_DATA\_DIR：数据目录

ETCD\_LISTEN\_PEER\_URLS：集群通信监听地址

ETCD\_LISTEN\_CLIENT\_URLS：客户端访问监听地址

ETCD\_INITIAL\_ADVERTISE\_PEER\_URLS：集群通告地址

ETCD\_ADVERTISE\_CLIENT\_URLS：客户端通告地址

ETCD\_INITIAL\_CLUSTER：集群节点地址

ETCD\_INITIAL\_CLUSTER\_TOKEN：集群Token

ETCD\_INITIAL\_CLUSTER\_STATE：加入集群的当前状态，new是新集群，existing表示加入已有集群

在所有Master节点上配置systemd管理ETCD：

cat > /usr/lib/systemd/system/etcd.service << EOF

[Unit]

Description=Etcd Server

After=network.target

After=network-online.target

Wants=network-online.target

[Service]

Type=notify

EnvironmentFile=/etc/etcd/cfg/etcd.conf

ExecStart=/usr/local/bin/etcd \

--cert-file=/etc/etcd/ssl/server.pem \

--key-file=/etc/etcd/ssl/server-key.pem \

--peer-cert-file=/etc/etcd/ssl/server.pem \

--peer-key-file=/etc/etcd/ssl/server-key.pem \

--trusted-ca-file=/etc/etcd/ssl/ca.pem \

--peer-trusted-ca-file=/etc/etcd/ssl/ca.pem \

--logger=zap

Restart=on-failure

LimitNOFILE=65536

[Install]

WantedBy=multi-user.target

EOF

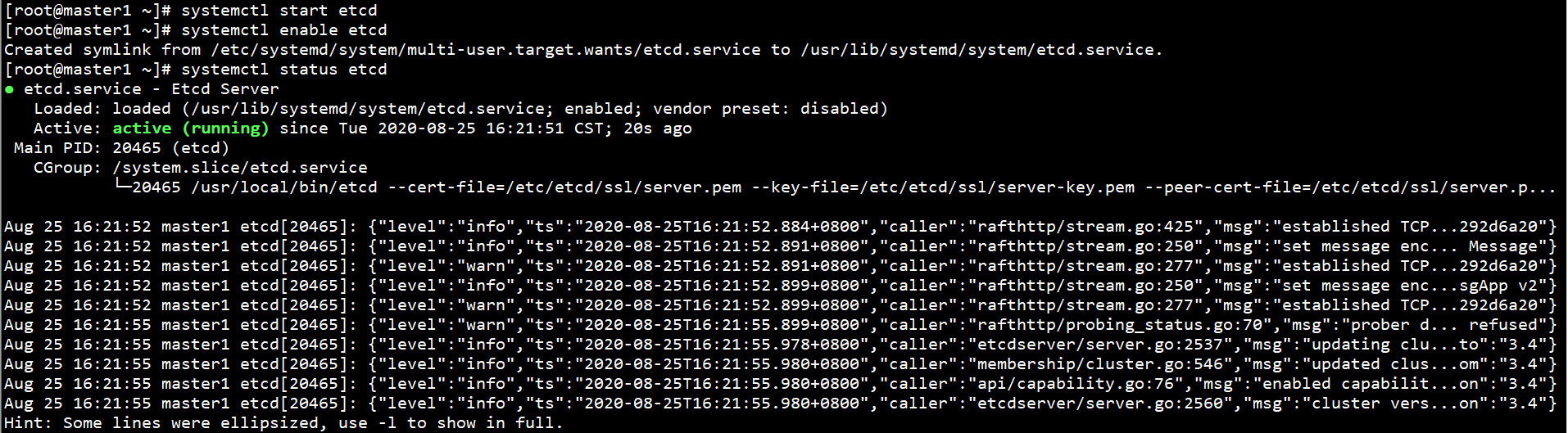


在所有Master节点上启动ETCD，并设置自启动：

systemctl start etcd

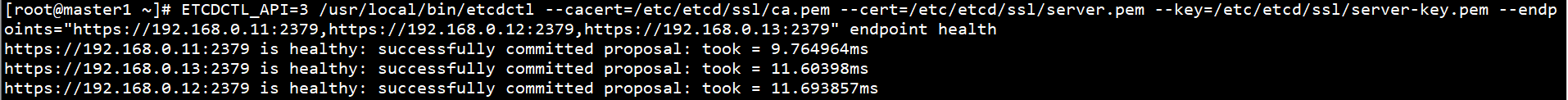
systemctl enable etcd

systemctl status etcd



在任意Master节点上查看ETCD集群状态：

ETCDCTL\_API=3 /usr/local/bin/etcdctl --cacert=/etc/etcd/ssl/ca.pem --cert=/etc/etcd/ssl/server.pem --key=/etc/etcd/ssl/server-key.pem --endpoints="https://192.168.0.11:2379,https://192.168.0.12:2379,https://192.168.0.13:2379" endpoint health



### 3、部署Docker

下载Docker二进制文件：

参考地址：https://download.docker.com/linux/static/stable/x86\_64/

下载地址：https://download.docker.com/linux/static/stable/x86\_64/docker-19.03.12.tgz

解压Docker二进制文件至系统目录：

tar -xf /root/docker-19.03.12.tgz -C /root/

mv /root/docker/\* /usr/local/bin



配置systemd管理Docker：

cat > /usr/lib/systemd/system/docker.service << EOF

[Unit]

Description=Docker Application Container Engine

Documentation=https://docs.docker.com

After=network-online.target firewalld.service

Wants=network-online.target

[Service]

Type=notify

ExecStart=/usr/local/bin/dockerd

ExecReload=/bin/kill -s HUP $MAINPID

LimitNOFILE=infinity

LimitNPROC=infinity

LimitCORE=infinity

TimeoutStartSec=0

Delegate=yes

KillMode=process

Restart=on-failure

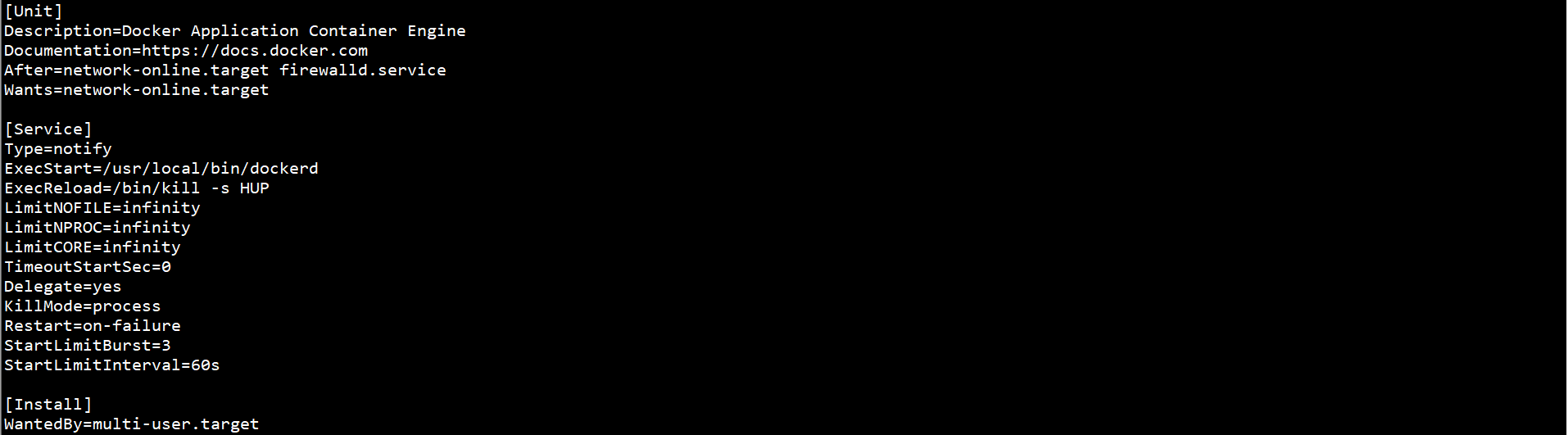
StartLimitBurst=3

StartLimitInterval=60s

[Install]

WantedBy=multi-user.target

EOF

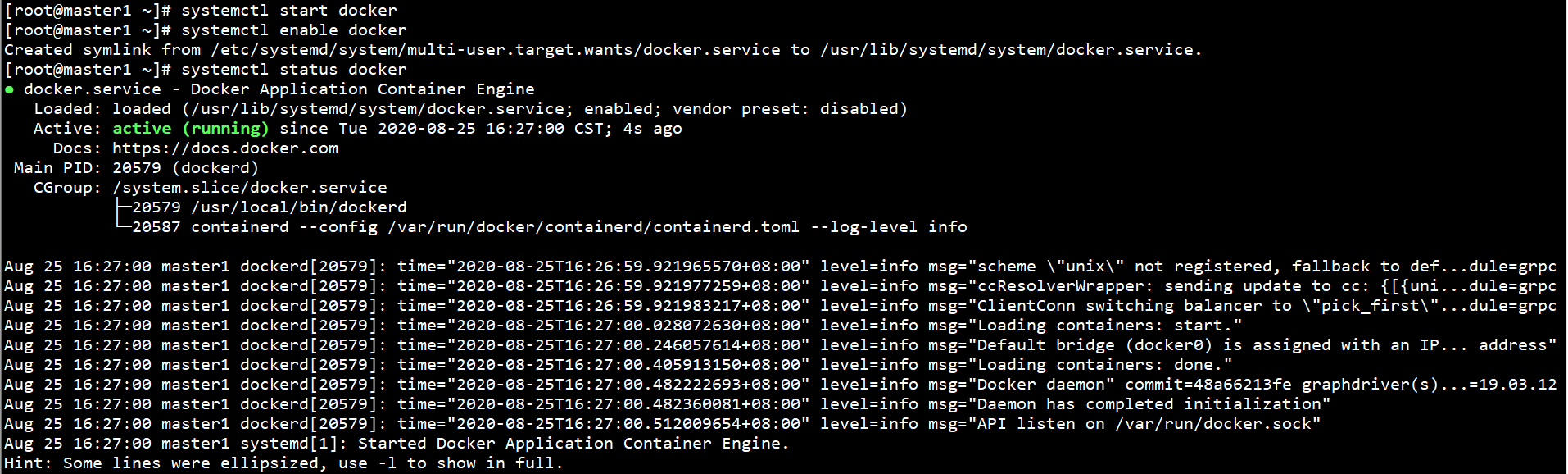


启动Docker，并设置自启动：

systemctl start docker

systemctl enable docker

systemctl status docker



### 4、部署Master节点

在所有Master节点上创建证书目录：

mkdir -p /etc/kubernetes/ssl/



在Master1节点上创建自签证书颁发机构CA：

cd /etc/kubernetes/ssl/



cat > ca-config.json << EOF

{

"signing": {

"default": {

"expiry": "87600h"

},

"profiles": {

"kubernetes": {

"expiry": "87600h",

"usages": [

"signing",

"key encipherment",

"server auth",

"client auth"

]

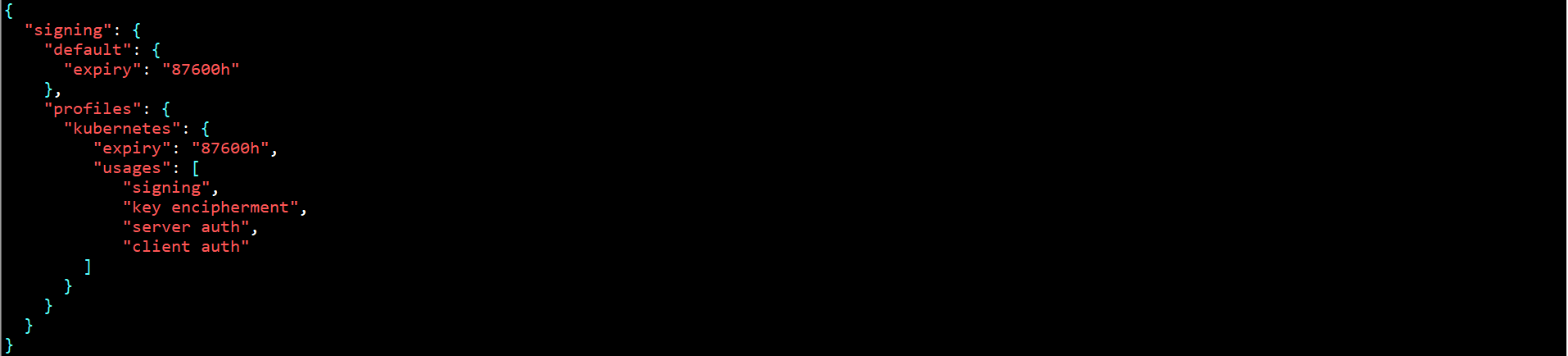
}

}

}

}

EOF



cat > ca-csr.json << EOF

{

"CN": "kubernetes",

"key": {

"algo": "rsa",

"size": 2048

},

"names": [

{

"C": "CN",

"L": "Beijing",

"ST": "Beijing",

"O": "k8s",

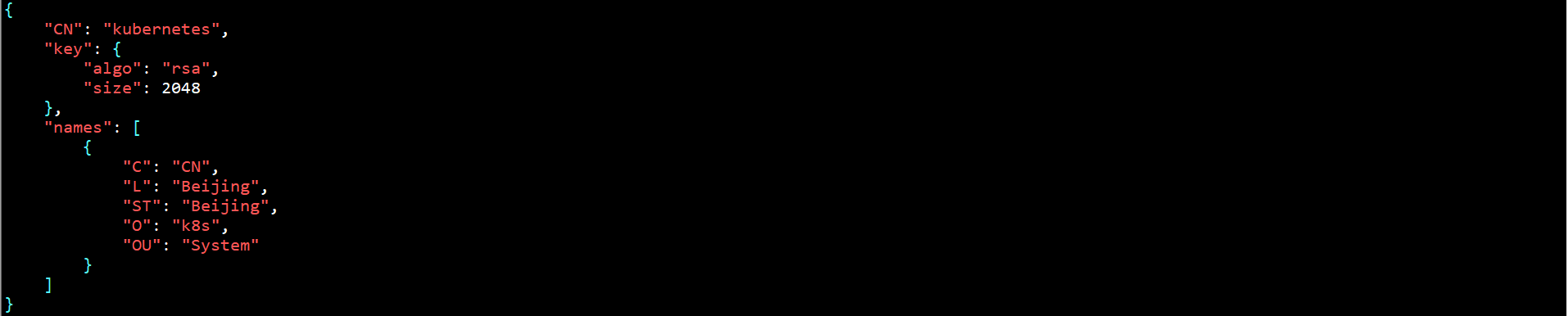
"OU": "System"

}

]

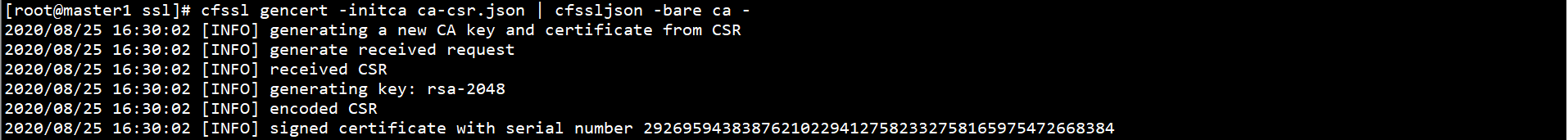
}

EOF

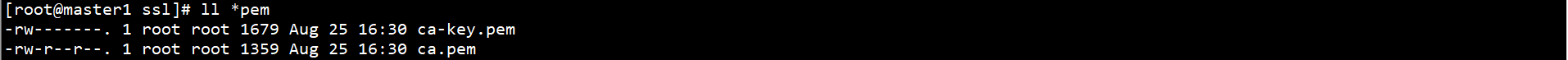


在Master1节点上生成证书：

cfssl gencert -initca ca-csr.json | cfssljson -bare ca -



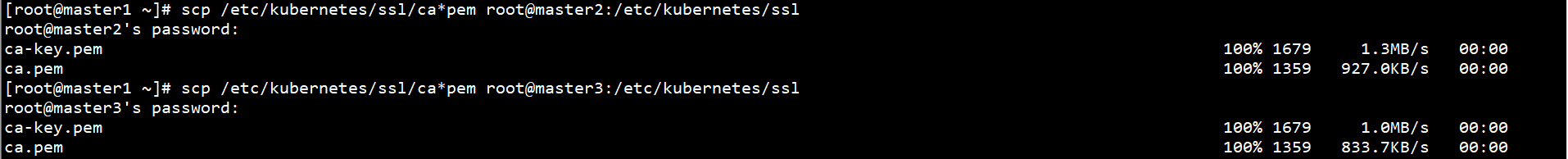
ll \*pem



将Master1上所有生成的证书拷贝至其余Master节点：

scp /etc/kubernetes/ssl/ca\*pem root@master2:/etc/kubernetes/ssl

scp /etc/kubernetes/ssl/ca\*pem root@master3:/etc/kubernetes/ssl



在Master1节点上创建kube-apiserver HTTPS证书申请文件：

cd /etc/kubernetes/ssl/



cat > server-csr.json << EOF

{

"CN": "kubernetes",

"hosts": [

"10.0.0.1",

"127.0.0.1",

"192.168.0.10",

"192.168.0.11",

"192.168.0.12",

"192.168.0.13",

"192.168.0.21",

"192.168.0.22",

"192.168.0.23",

"kubernetes",

"kubernetes.default",

"kubernetes.default.svc",

"kubernetes.default.svc.cluster",

"kubernetes.default.svc.cluster.local"

],

"key": {

"algo": "rsa",

"size": 2048

},

"names": [

{

"C": "CN",

"L": "BeiJing",

"ST": "BeiJing",

"O": "k8s",

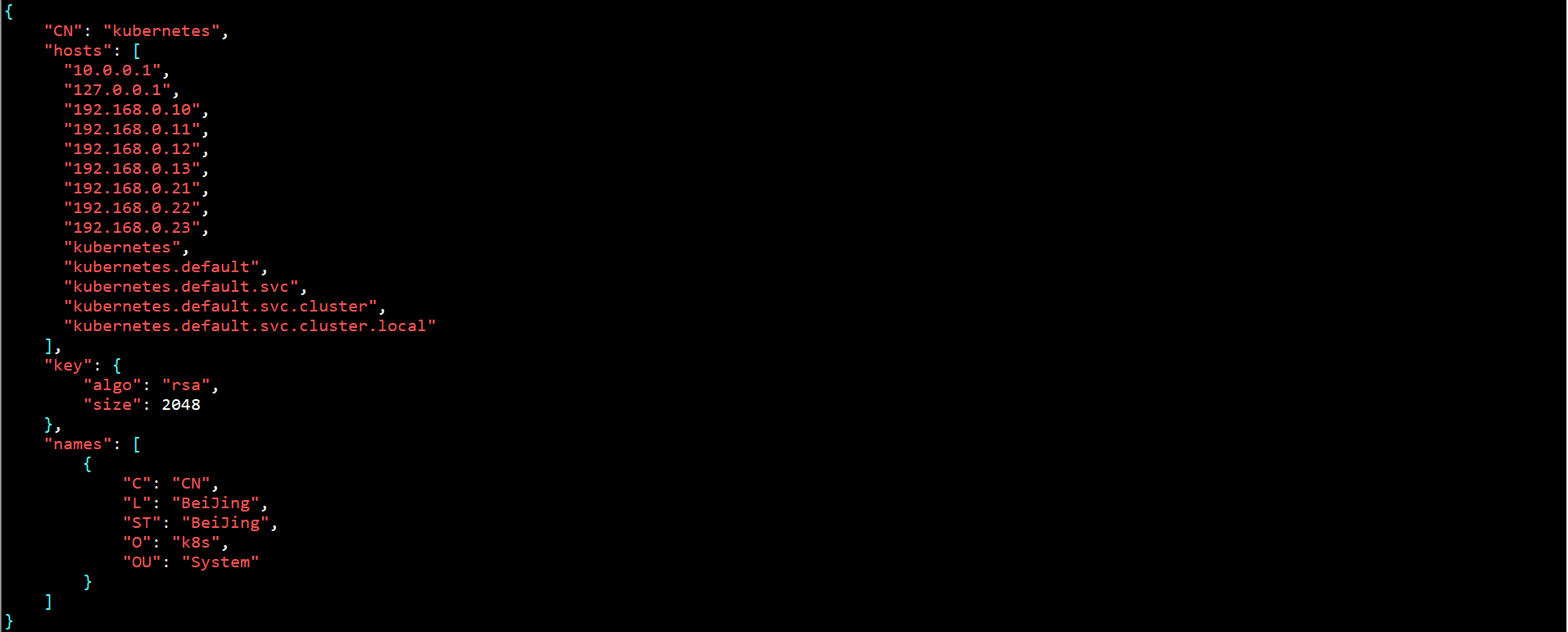
"OU": "System"

}

]

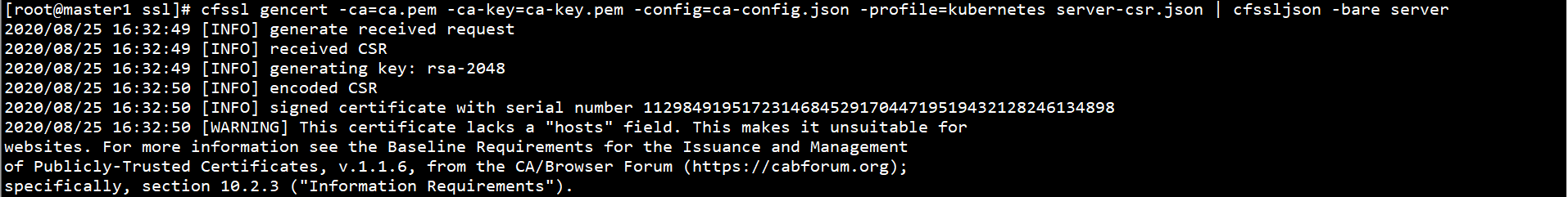
}

EOF

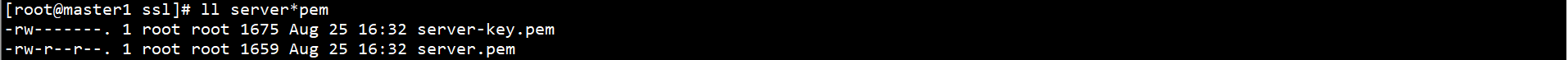


在Master1节点上生成证书：

cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=kubernetes server-csr.json | cfssljson -bare server



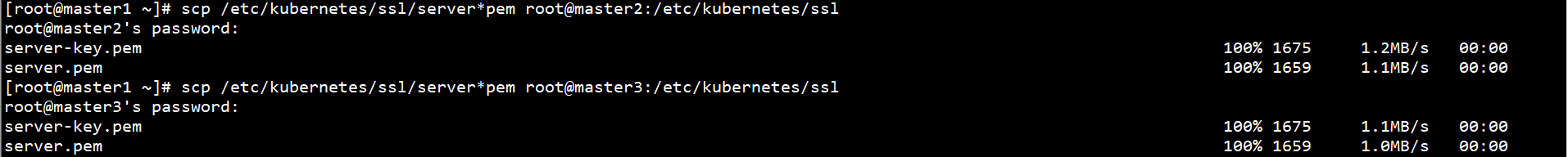
ll server\*pem



将Master1节点上所有生成的证书拷贝至其余Master节点：

scp /etc/kubernetes/ssl/server\*pem root@master2:/etc/kubernetes/ssl

scp /etc/kubernetes/ssl/server\*pem root@master3:/etc/kubernetes/ssl



在所有节点上下载Kubernetes二进制文件：

参考地址：https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.18.md

下载地址：https://dl.k8s.io/v1.18.5/kubernetes-server-linux-amd64.tar.gz

在所有Master节点上创建Kubernetes配置文件及日志目录：

mkdir -p /etc/kubernetes/cfg/

mkdir -p /var/log/kubernetes/logs/



解压Kubernetes二进制文件至系统目录：

tar -xf /root/kubernetes-server-linux-amd64.tar.gz -C /root/

mv /root/kubernetes/server/bin/{kubectl,kube-apiserver,kube-scheduler,kube-controller-manager} /usr/local/bin/



在所有Master节点上部署kube-apiserver：

在所有Master节点上创建token文件：

cat > /etc/kubernetes/cfg/token.csv << EOF

c47ffb939f5ca36231d9e3121a252940,kubelet-bootstrap,10001,"system:node-bootstrapper"

EOF



格式：token，用户名，UID，用户组

token可自行生成替换：

head -c 16 /dev/urandom | od -An -t x | tr -d ' '



在Master节点上创建kube-apiserver配置文件：

cat > /etc/kubernetes/cfg/kube-apiserver.conf << EOF

KUBE\_APISERVER\_OPTS="--logtostderr=false \\

--v=2 \\

--log-dir=/var/log/kubernetes/logs \\

--etcd-servers=https://192.168.0.11:2379,https://192.168.0.12:2379,https://192.168.0.13:2379 \\

--bind-address=192.168.0.11 \\

--secure-port=6443 \\

--advertise-address=192.168.0.11 \\

--allow-privileged=true \\

--service-cluster-ip-range=10.0.0.0/24 \\

--enable-admission-plugins=NamespaceLifecycle,LimitRanger,ServiceAccount,ResourceQuota,NodeRestriction \\

--authorization-mode=RBAC,Node \\

--enable-bootstrap-token-auth=true \\

--token-auth-file=/etc/kubernetes/cfg/token.csv \\

--service-node-port-range=30000-32767 \\

--kubelet-client-certificate=/etc/kubernetes/ssl/server.pem \\

--kubelet-client-key=/etc/kubernetes/ssl/server-key.pem \\

--tls-cert-file=/etc/kubernetes/ssl/server.pem \\

--tls-private-key-file=/etc/kubernetes/ssl/server-key.pem \\

--client-ca-file=/etc/kubernetes/ssl/ca.pem \\

--service-account-key-file=/etc/kubernetes/ssl/ca-key.pem \\

--etcd-cafile=/etc/etcd/ssl/ca.pem \\

--etcd-certfile=/etc/etcd/ssl/server.pem \\

--etcd-keyfile=/etc/etcd/ssl/server-key.pem \\

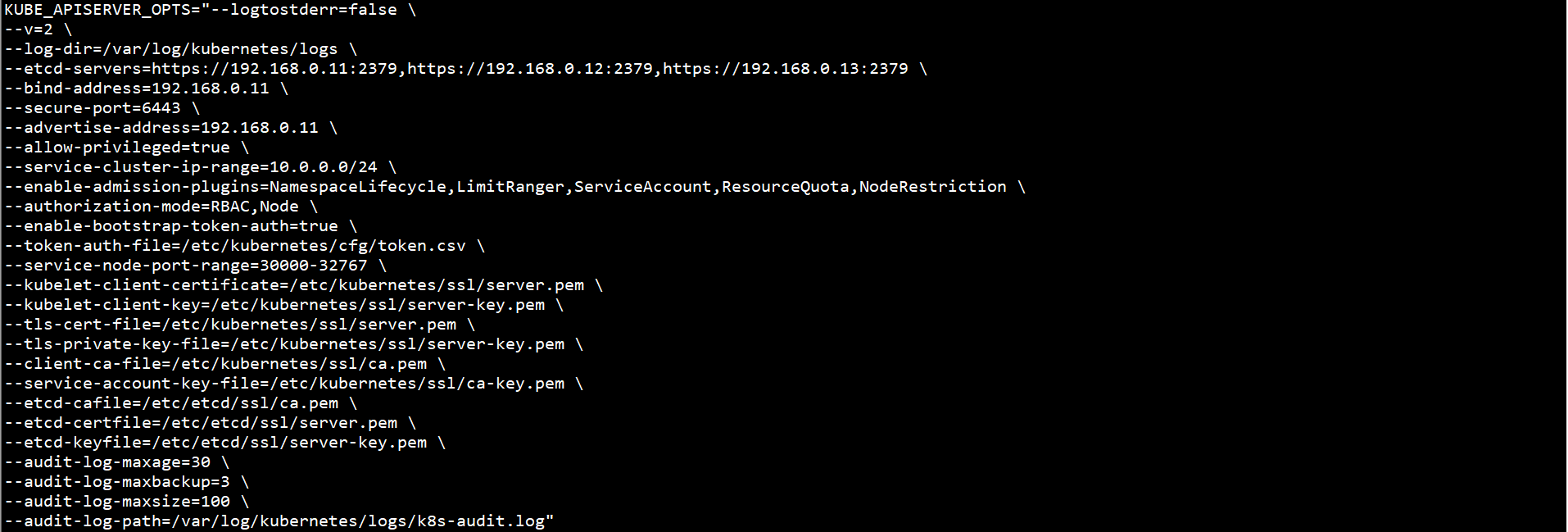
--audit-log-maxage=30 \\

--audit-log-maxbackup=3 \\

--audit-log-maxsize=100 \\

--audit-log-path=/var/log/kubernetes/logs/k8s-audit.log"

EOF



kube-apiserver配置文件参数说明：

--logtostderr：启用日志

---v：日志等级

--log-dir：日志目录

--etcd-servers：ETCD集群地址

--bind-address：监听地址

--secure-port：https安全端口

--advertise-address：集群通告地址

--allow-privileged：启用授权

--service-cluster-ip-range：Service虚拟IP地址段

--enable-admission-plugins：准入控制模块

--authorization-mode：认证授权，启用RBAC授权和节点自管理

--enable-bootstrap-token-auth：启用TLS bootstrap机制

--token-auth-file：bootstrap token文件

--service-node-port-range：Service NodePort类型默认分配端口范围

--kubelet-client-xxx：kube-apiserver访问kubelet客户端证书

--tls-xxx-file：kube-apiserver https证书

--etcd-xxxfile：连接ETCD集群证书

--audit-log-xxx：审计日志

在所有Master节点上配置systemd管理kube-apiserver：

cat > /usr/lib/systemd/system/kube-apiserver.service << EOF

[Unit]

Description=Kubernetes API Server

Documentation=https://github.com/kubernetes/kubernetes

[Service]

EnvironmentFile=/etc/kubernetes/cfg/kube-apiserver.conf

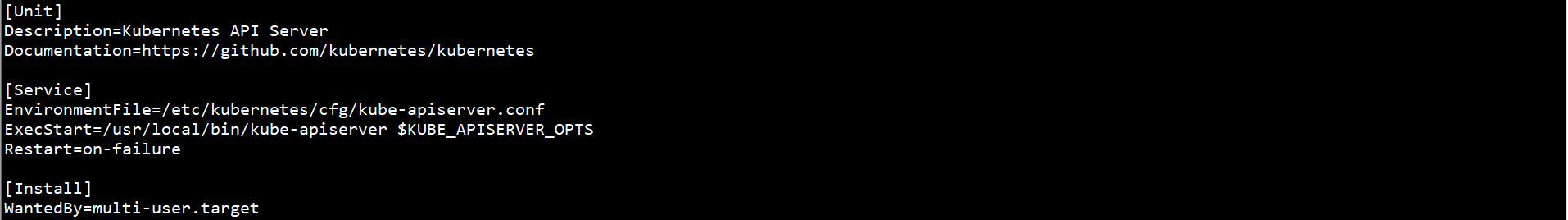
ExecStart=/usr/local/bin/kube-apiserver \$KUBE\_APISERVER\_OPTS

Restart=on-failure

[Install]

WantedBy=multi-user.target

EOF

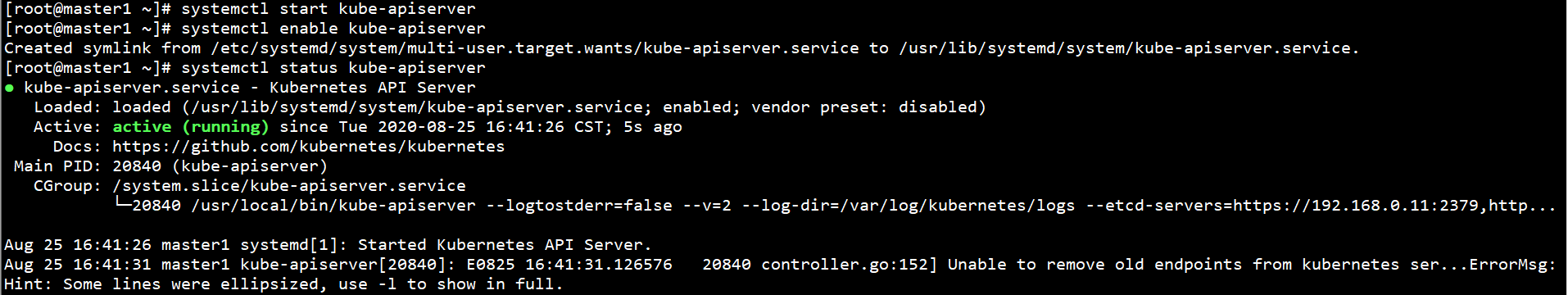


在Master节点上启动kube-apiserver，并设置自启动：

systemctl start kube-apiserver

systemctl enable kube-apiserver

systemctl status kube-apiserver



在任意Master节点上授权kubelet-bootstrap用户允许请求证书

kubectl create clusterrolebinding kubelet-bootstrap --clusterrole=system:node-bootstrapper --user=kubelet-bootstrap



在所有Master节点上部署kube-controller-manager：

在所有Master节点上创建kube-controller-manager配置文件：

cat > /etc/kubernetes/cfg/kube-controller-manager.conf << EOF

KUBE\_CONTROLLER\_MANAGER\_OPTS="--logtostderr=false \\

--v=2 \\

--log-dir=/var/log/kubernetes/logs \\

--leader-elect=true \\

--master=127.0.0.1:8080 \\

--bind-address=127.0.0.1 \\

--allocate-node-cidrs=true \\

--cluster-cidr=10.244.0.0/16 \\

--service-cluster-ip-range=10.0.0.0/24 \\

--cluster-signing-cert-file=/etc/kubernetes/ssl/ca.pem \\

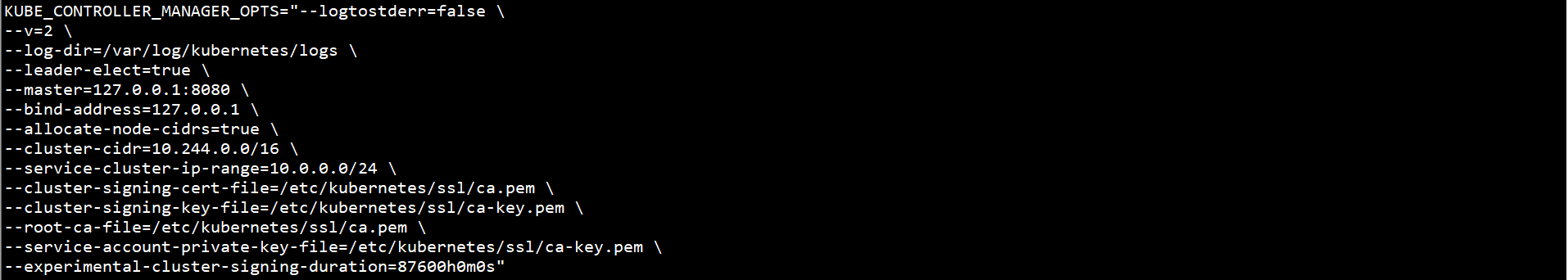
--cluster-signing-key-file=/etc/kubernetes/ssl/ca-key.pem \\

--root-ca-file=/etc/kubernetes/ssl/ca.pem \\

--service-account-private-key-file=/etc/kubernetes/ssl/ca-key.pem \\

--experimental-cluster-signing-duration=87600h0m0s"

EOF



kube-controller-manager配置文件参数说明：

--master：通过本地非安全本地端口8080连接kube-apiserver

--leader-elect：当该组件启动多个时，自动选举

--cluster-signing-cert-file/--cluster-signing-key-file：自动为kubelet颁发证书的CA，与kube-apiserver保持一致

在所有Master节点上配置systemd管理kube-controller-manager：

cat > /usr/lib/systemd/system/kube-controller-manager.service << EOF

[Unit]

Description=Kubernetes Controller Manager

Documentation=https://github.com/kubernetes/kubernetes

[Service]

EnvironmentFile=/etc/kubernetes/cfg/kube-controller-manager.conf

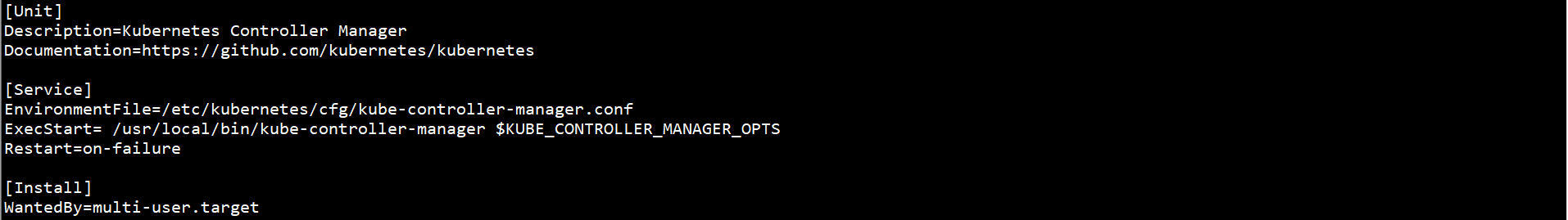
ExecStart= /usr/local/bin/kube-controller-manager \$KUBE\_CONTROLLER\_MANAGER\_OPTS

Restart=on-failure

[Install]

WantedBy=multi-user.target

EOF

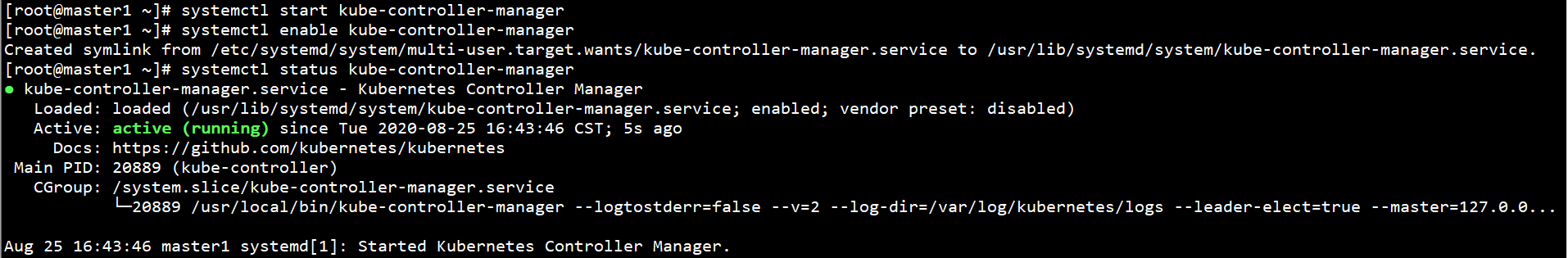


在所有Master节点上启动kube-controller-manager，并设置自启动：

systemctl start kube-controller-manager

systemctl enable kube-controller-manager

systemctl status kube-controller-manager



在所有Master节点上部署kube-scheduler：

在所有Master节点上创建kube-scheduler配置文件：

cat > /etc/kubernetes/cfg/kube-scheduler.conf << EOF

KUBE\_SCHEDULER\_OPTS="--logtostderr=false \\

--v=2 \\

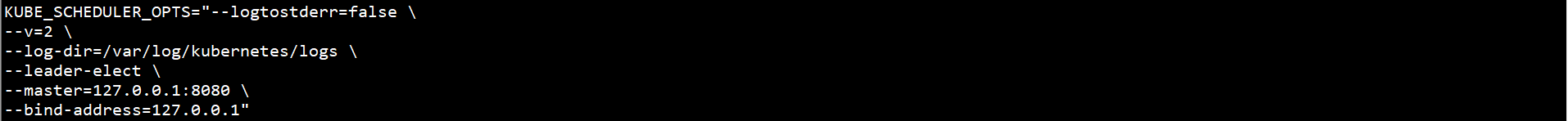
--log-dir=/var/log/kubernetes/logs \\

--leader-elect \\

--master=127.0.0.1:8080 \\

--bind-address=127.0.0.1"

EOF



kube-scheduler配置文件参数说明：

--master：通过本地非安全本地端口8080连接kube-apiserver

--leader-elect：当该组件启动多个时，自动选举

在所有Master节点上配置systemd管理kube-scheduler：

cat > /usr/lib/systemd/system/kube-scheduler.service << EOF

[Unit]

Description=Kubernetes Scheduler

Documentation=https://github.com/kubernetes/kubernetes

[Service]

EnvironmentFile=/etc/kubernetes/cfg/kube-scheduler.conf

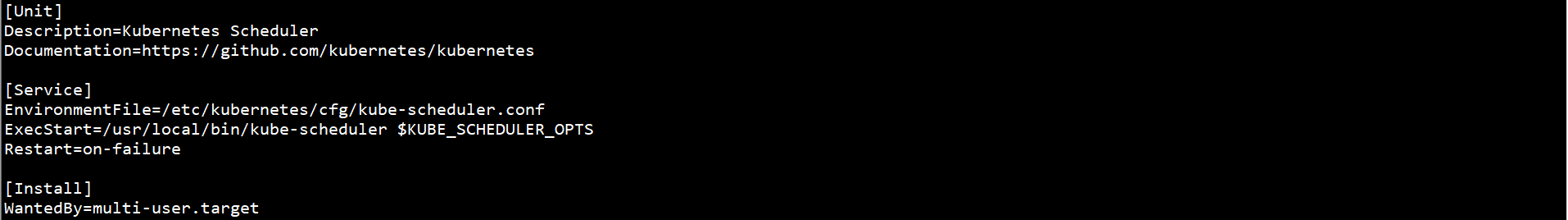
ExecStart=/usr/local/bin/kube-scheduler \$KUBE\_SCHEDULER\_OPTS

Restart=on-failure

[Install]

WantedBy=multi-user.target

EOF

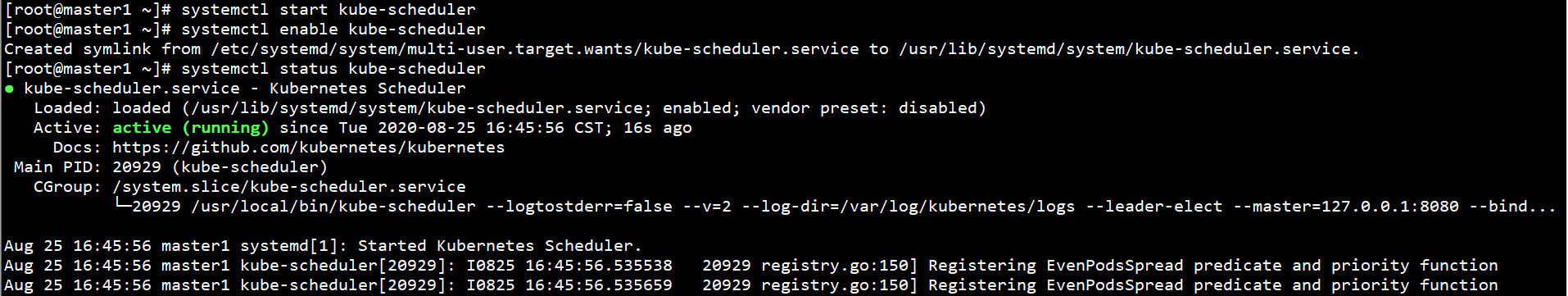


在所有Master节点上启动kube-scheduler，并设置自启动：

systemctl start kube-scheduler

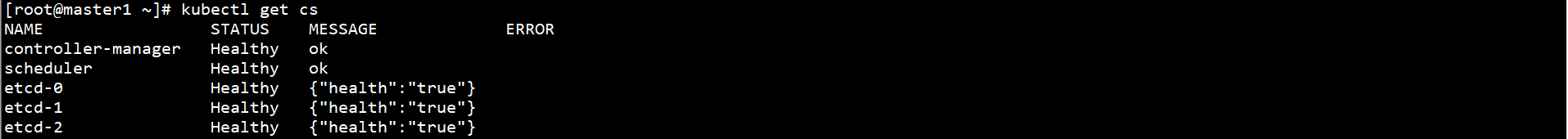
systemctl enable kube-scheduler

systemctl status kube-scheduler



在任意Master节点上查看集群状态：

kubectl get cs



### 5、部署Worker节点

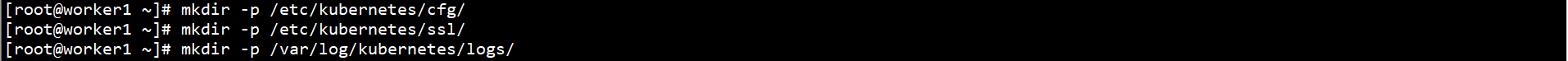
Master节点同时作为Worker节点，需配置kubelet和kube-proxy。

在所有Worker节点上创建Kubernetes配置文件及日志目录：

mkdir -p /etc/kubernetes/cfg/

mkdir -p /etc/kubernetes/ssl/

mkdir -p /var/log/kubernetes/logs/

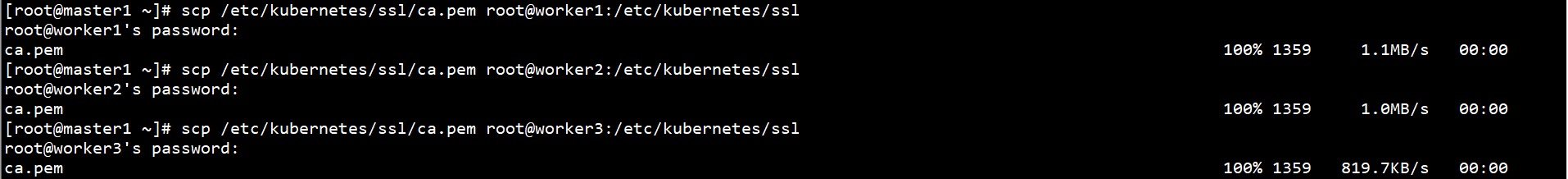


将Master1上所有生成的证书拷贝至Worker节点：

scp /etc/kubernetes/ssl/ca.pem root@worker1:/etc/kubernetes/ssl

scp /etc/kubernetes/ssl/ca.pem root@worker2:/etc/kubernetes/ssl

scp /etc/kubernetes/ssl/ca.pem root@worker3:/etc/kubernetes/ssl



在所有Worker节点上解压Kubernetes二进制文件：

tar -xf /root/kubernetes-server-linux-amd64.tar.gz -C /root/



在所有节点上将Kubernetes二进制文件至系统目录：

mv /root/kubernetes/server/bin/{kubelet,kube-proxy} /usr/local/bin



在所有节点上部署kubelet：

在所有节点上创建kubelet配置文件：

cat > /etc/kubernetes/cfg/kubelet.conf << EOF

KUBELET\_OPTS="--logtostderr=false \\

--v=2 \\

--log-dir=/var/log/kubernetes/logs \\

--hostname-override=master1 \\

--network-plugin=cni \\

--kubeconfig=/etc/kubernetes/cfg/kubelet.kubeconfig \\

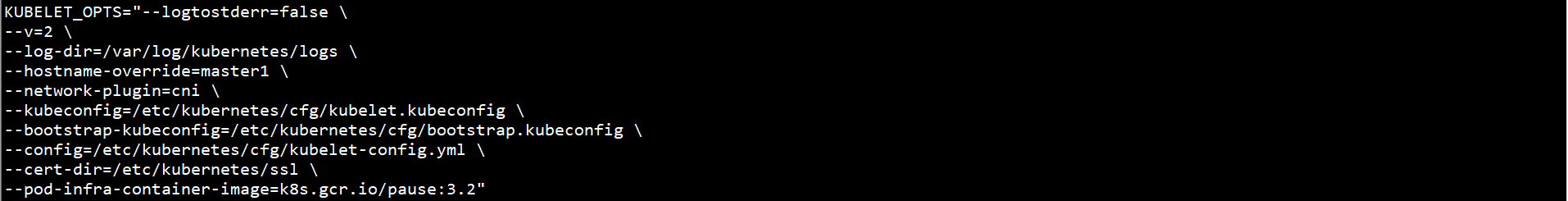
--bootstrap-kubeconfig=/etc/kubernetes/cfg/bootstrap.kubeconfig \\

--config=/etc/kubernetes/cfg/kubelet-config.yml \\

--cert-dir=/etc/kubernetes/ssl \\

--pod-infra-container-image=k8s.gcr.io/pause:3.2"

EOF



kubelet配置文件参数说明：

--hostname-override：显示名称，集群中唯一

--network-plugin：启用CNI

--kubeconfig：指定路径及名称，自动生成，用于连接kube-apiserver

--bootstrap-kubeconfig：首次启动向kube-apiserver申请证书

--config：配置参数文件

--cert-dir：kubelet证书生成目录

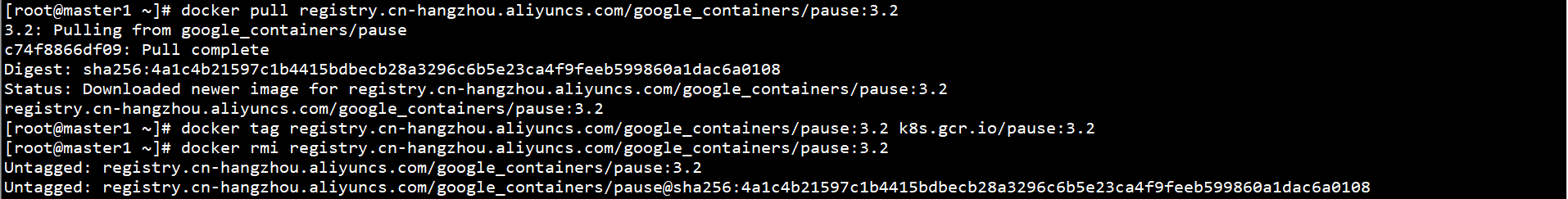
--pod-infra-container-image：管理Pod网络容器的镜像

k8s.gcr.io/pause:3.2无法直接下载，需通过阿里云镜像仓库下载：

docker pull registry.cn-hangzhou.aliyuncs.com/google\_containers/pause:3.2

docker tag registry.cn-hangzhou.aliyuncs.com/google\_containers/pause:3.2 k8s.gcr.io/pause:3.2

docker rmi registry.cn-hangzhou.aliyuncs.com/google\_containers/pause:3.2



在所有节点上创建kubelet配置文件：

cat > /etc/kubernetes/cfg/kubelet-config.yml << EOF

kind: KubeletConfiguration

apiVersion: kubelet.config.k8s.io/v1beta1

address: 0.0.0.0

port: 10250

readOnlyPort: 10255

cgroupDriver: cgroupfs

clusterDNS:

- 10.0.0.2

clusterDomain: cluster.local

failSwapOn: false

authentication:

anonymous:

enabled: false

webhook:

cacheTTL: 2m0s

enabled: true

x509:

clientCAFile: /etc/kubernetes/ssl/ca.pem

authorization:

mode: Webhook

webhook:

cacheAuthorizedTTL: 5m0s

cacheUnauthorizedTTL: 30s

evictionHard:

imagefs.available: 15%

memory.available: 100Mi

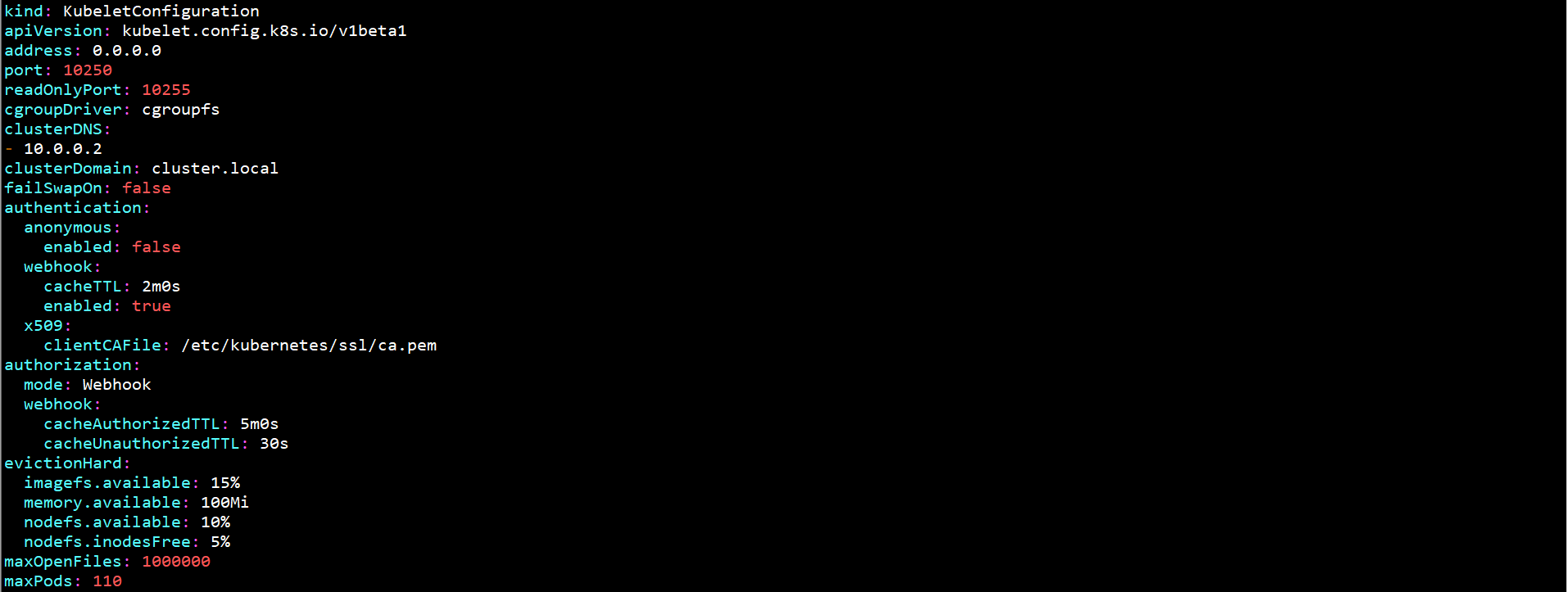
nodefs.available: 10%

nodefs.inodesFree: 5%

maxOpenFiles: 1000000

maxPods: 110

EOF



在Master1节点上生成bootstrap.kubeconfig文件：

KUBE\_APISERVER="https://192.168.0.10:8443"

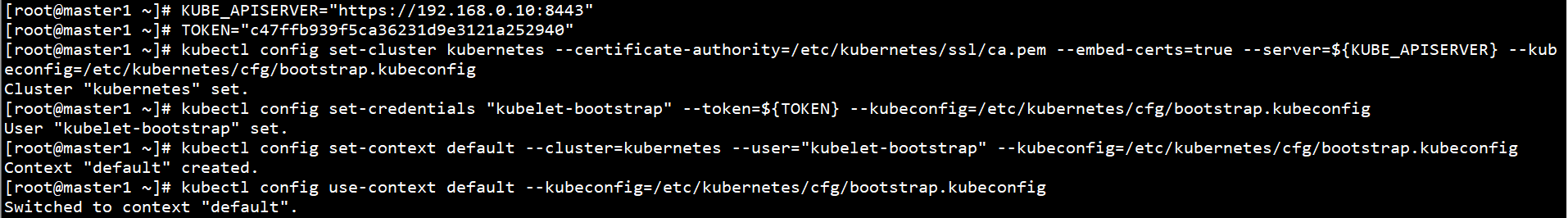
TOKEN="c47ffb939f5ca36231d9e3121a252940"

kubectl config set-cluster kubernetes --certificate-authority=/etc/kubernetes/ssl/ca.pem --embed-certs=true --server=${KUBE\_APISERVER} --kubeconfig=/etc/kubernetes/cfg/bootstrap.kubeconfig

kubectl config set-credentials "kubelet-bootstrap" --token=${TOKEN} --kubeconfig=/etc/kubernetes/cfg/bootstrap.kubeconfig

kubectl config set-context default --cluster=kubernetes --user="kubelet-bootstrap" --kubeconfig=/etc/kubernetes/cfg/bootstrap.kubeconfig

kubectl config use-context default --kubeconfig=/etc/kubernetes/cfg/bootstrap.kubeconfig



将Master1节点上所有生成的bootstrap.kubeconfig文件拷贝至其余节点：

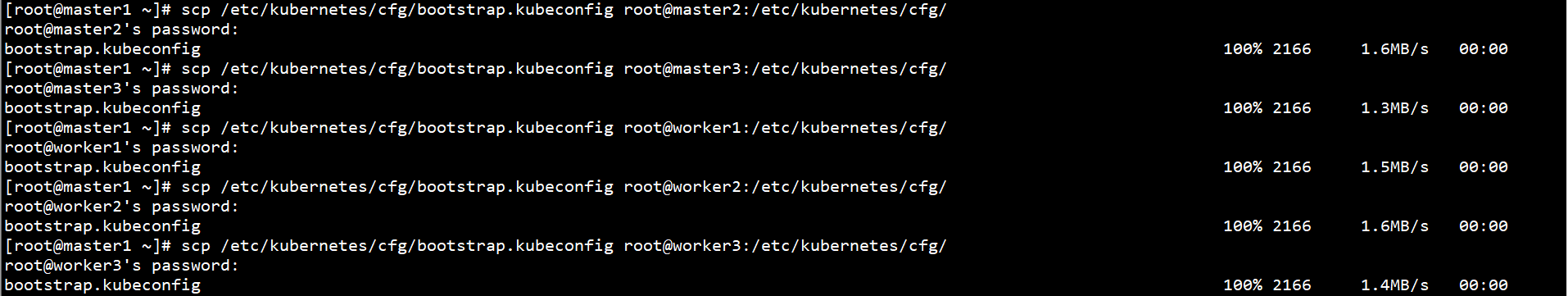
scp /etc/kubernetes/cfg/bootstrap.kubeconfig root@master2:/etc/kubernetes/cfg/

scp /etc/kubernetes/cfg/bootstrap.kubeconfig root@master3:/etc/kubernetes/cfg/

scp /etc/kubernetes/cfg/bootstrap.kubeconfig root@worker1:/etc/kubernetes/cfg/

scp /etc/kubernetes/cfg/bootstrap.kubeconfig root@worker2:/etc/kubernetes/cfg/

scp /etc/kubernetes/cfg/bootstrap.kubeconfig root@worker3:/etc/kubernetes/cfg/



在所有节点上配置systemd管理kubelet：

cat > /usr/lib/systemd/system/kubelet.service << EOF

[Unit]

Description=Kubernetes Kubelet

After=docker.service

[Service]

EnvironmentFile=/etc/kubernetes/cfg/kubelet.conf

ExecStart=/usr/local/bin/kubelet \$KUBELET\_OPTS

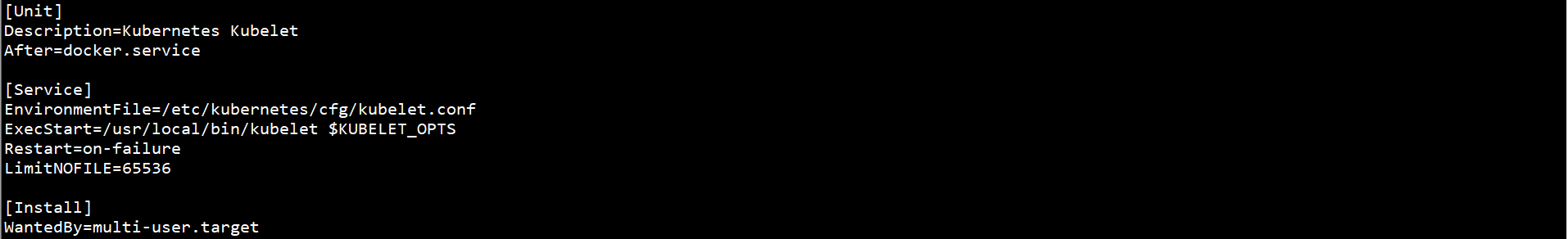
Restart=on-failure

LimitNOFILE=65536

[Install]

WantedBy=multi-user.target

EOF

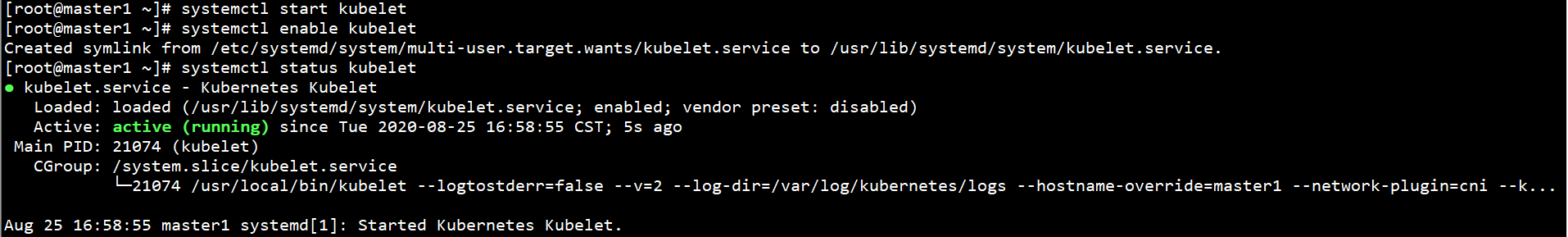


在所有节点上启动kubelet，并设置自启动：

systemctl start kubelet

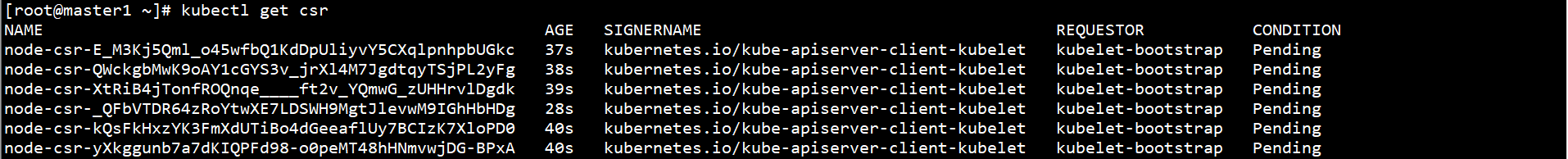
systemctl enable kubelet

systemctl status kubelet

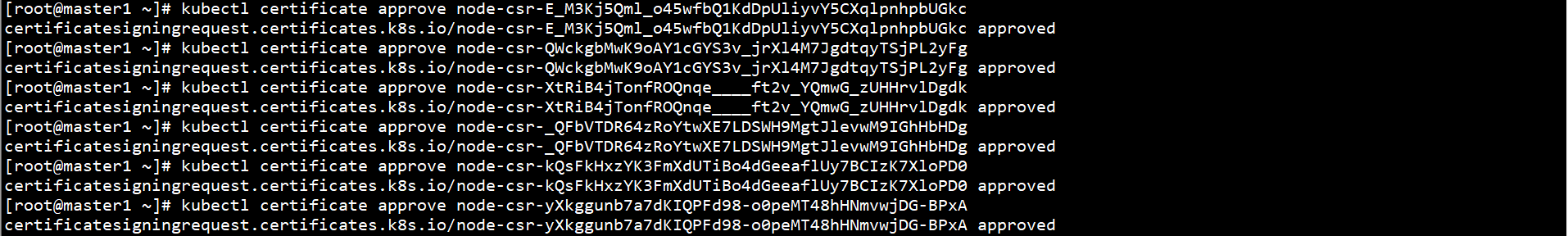


在任意Master节点上批准kubelet证书申请：

kubectl get csr



kubectl certificate approve node-csr-xxx



在所有节点上部署kube-proxy：

在所有节点上创建kube-proxy配置文件：

cat > /etc/kubernetes/cfg/kube-proxy.conf << EOF

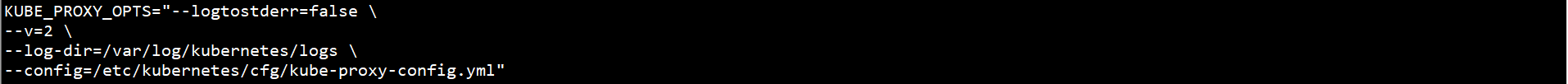
KUBE\_PROXY\_OPTS="--logtostderr=false \\

--v=2 \\

--log-dir=/var/log/kubernetes/logs \\

--config=/etc/kubernetes/cfg/kube-proxy-config.yml"

EOF



在所有节点上创建kube-proxy配置文件：

cat > /etc/kubernetes/cfg/kube-proxy-config.yml << EOF

kind: KubeProxyConfiguration

apiVersion: kubeproxy.config.k8s.io/v1alpha1

bindAddress: 0.0.0.0

metricsBindAddress: 0.0.0.0:10249

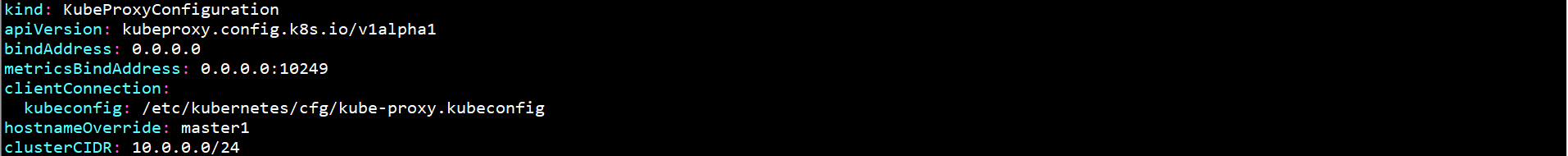
clientConnection:

kubeconfig: /etc/kubernetes/cfg/kube-proxy.kubeconfig

hostnameOverride: master1

clusterCIDR: 10.0.0.0/24

EOF



在Master1节点上创建kube-proxy证书申请文件：

cd /etc/kubernetes/ssl/



cat > kube-proxy-csr.json << EOF

{

"CN": "system:kube-proxy",

"hosts": [],

"key": {

"algo": "rsa",

"size": 2048

},

"names": [

{

"C": "CN",

"L": "BeiJing",

"ST": "BeiJing",

"O": "k8s",

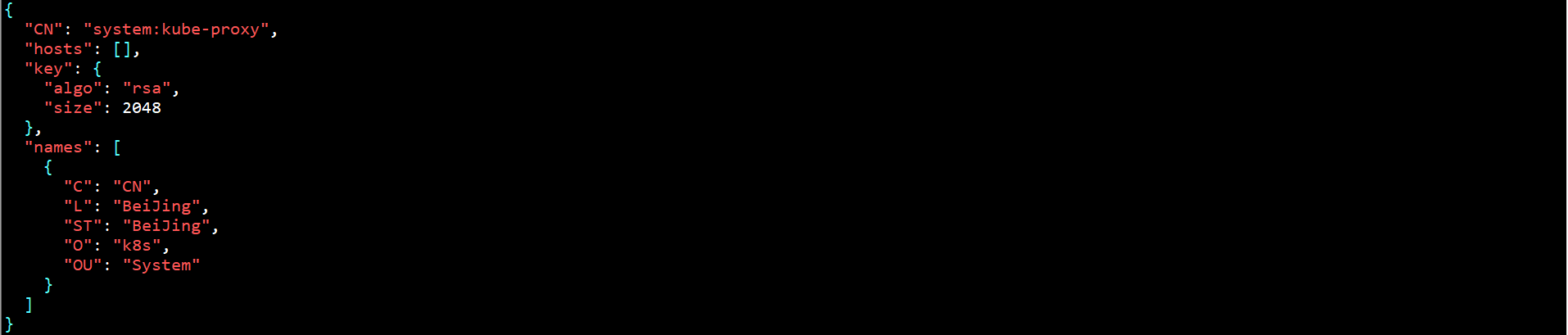
"OU": "System"

}

]

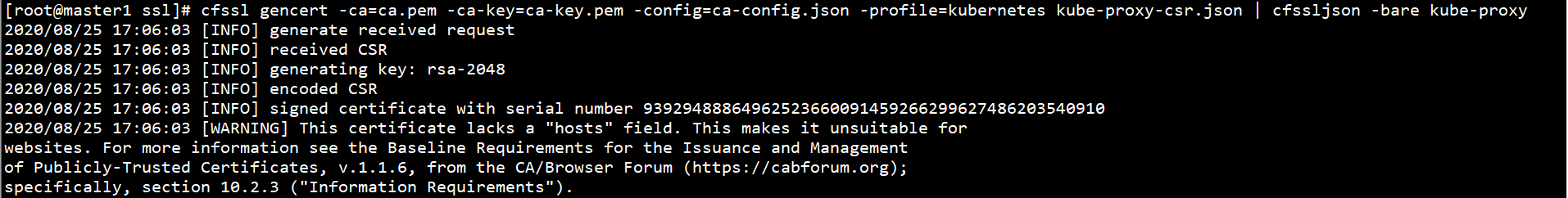
}

EOF

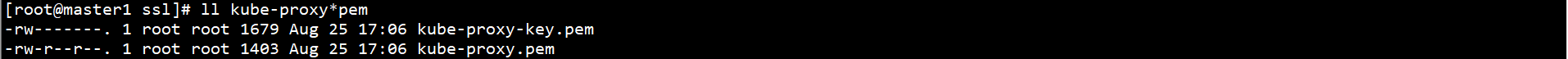


在Master1节点上生成证书：

cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=kubernetes kube-proxy-csr.json | cfssljson -bare kube-proxy



ll kube-proxy\*pem



在Master1节点上创建kube-proxy.kubeconfig文件：

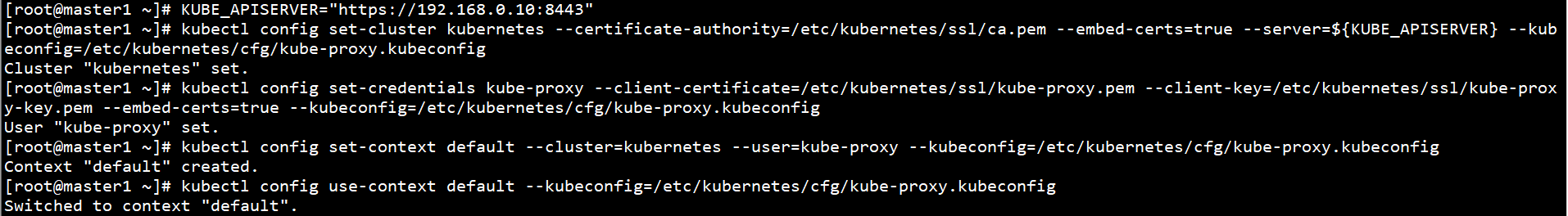
KUBE\_APISERVER="https://192.168.0.10:8443"

kubectl config set-cluster kubernetes --certificate-authority=/etc/kubernetes/ssl/ca.pem --embed-certs=true --server=${KUBE\_APISERVER} --kubeconfig=/etc/kubernetes/cfg/kube-proxy.kubeconfig

kubectl config set-credentials kube-proxy --client-certificate=/etc/kubernetes/ssl/kube-proxy.pem --client-key=/etc/kubernetes/ssl/kube-proxy-key.pem --embed-certs=true --kubeconfig=/etc/kubernetes/cfg/kube-proxy.kubeconfig

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

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



将Master1节点上所有生成的证书拷贝至Worker节点：

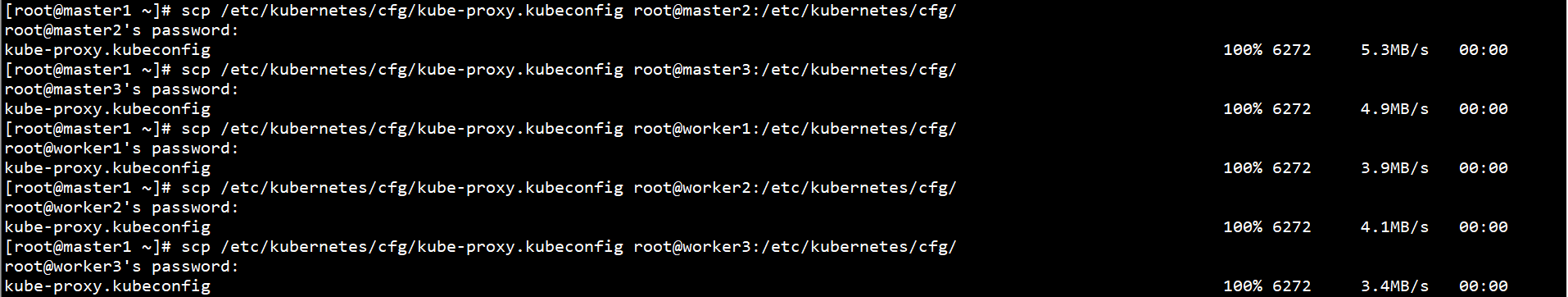
scp /etc/kubernetes/cfg/kube-proxy.kubeconfig root@master2:/etc/kubernetes/cfg/

scp /etc/kubernetes/cfg/kube-proxy.kubeconfig root@master3:/etc/kubernetes/cfg/

scp /etc/kubernetes/cfg/kube-proxy.kubeconfig root@worker1:/etc/kubernetes/cfg/

scp /etc/kubernetes/cfg/kube-proxy.kubeconfig root@worker2:/etc/kubernetes/cfg/

scp /etc/kubernetes/cfg/kube-proxy.kubeconfig root@worker3:/etc/kubernetes/cfg/



在所有节点上配置systemd管理kube-proxy：

cat > /usr/lib/systemd/system/kube-proxy.service << EOF

[Unit]

Description=Kubernetes Proxy

After=network.target

[Service]

EnvironmentFile=/etc/kubernetes/cfg/kube-proxy.conf

ExecStart=/usr/local/bin/kube-proxy \$KUBE\_PROXY\_OPTS

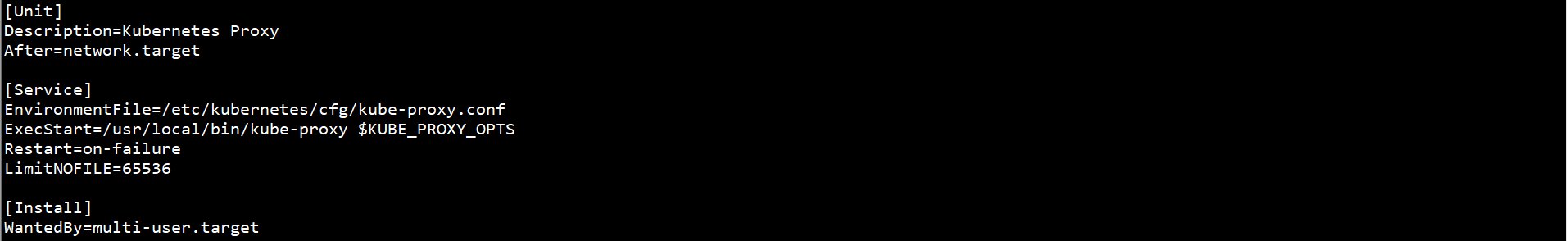
Restart=on-failure

LimitNOFILE=65536

[Install]

WantedBy=multi-user.target

EOF



在所有节点上启动kube-apiserver，并设置自启动：

systemctl start kube-proxy

systemctl enable kube-proxy

systemctl status kube-proxy



### 6、部署CNI网络（flannel）

下载CNI二进制文件：

参考地址：<https://github.com/containernetworking/plugins/releases/>

下载地址：https://github.com/containernetworking/plugins/releases/download/v0.8.6/cni-plugins-linux-amd64-v0.8.6.tgz

在所有节点上创建CNI二进制文件工作目录：

mkdir -p /opt/cni/bin



在所有节点上解压CNI二进制文件至工作目录：

tar -xf /root/cni-plugins-linux-amd64-v0.8.6.tgz -C /opt/cni/bin



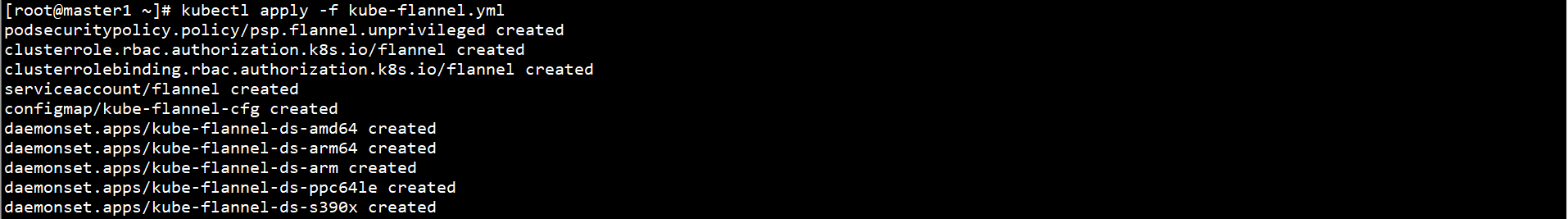
在Master1节点上部署CNI网络：

下载flannel部署文件：

下载地址：https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml

部署CNI网络：

kubectl apply -f kube-flannel.yml



在任意Master节点上授权kube-apiserver访问kubelet：

cat > apiserver-to-kubelet-rbac.yaml << EOF

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRole

metadata:

annotations:

rbac.authorization.kubernetes.io/autoupdate: "true"

labels:

kubernetes.io/bootstrapping: rbac-defaults

name: system:kube-apiserver-to-kubelet

rules:

- apiGroups:

- ""

resources:

- nodes/proxy

- nodes/stats

- nodes/log

- nodes/spec

- nodes/metrics

- pods/log

verbs:

- "\*"

---

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRoleBinding

metadata:

name: system:kube-apiserver

namespace: ""

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: system:kube-apiserver-to-kubelet

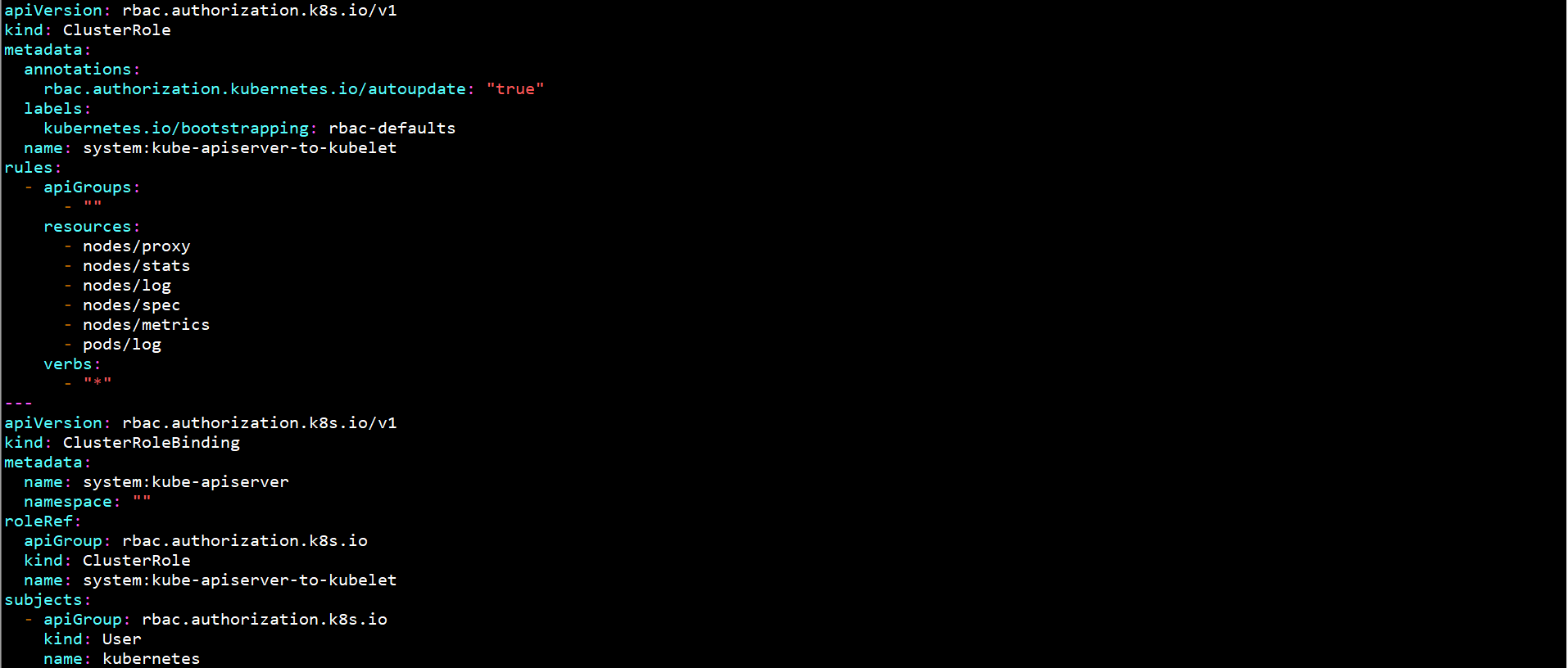
subjects:

- apiGroup: rbac.authorization.k8s.io

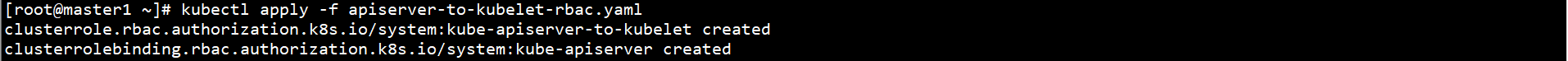
kind: User

name: kubernetes

EOF

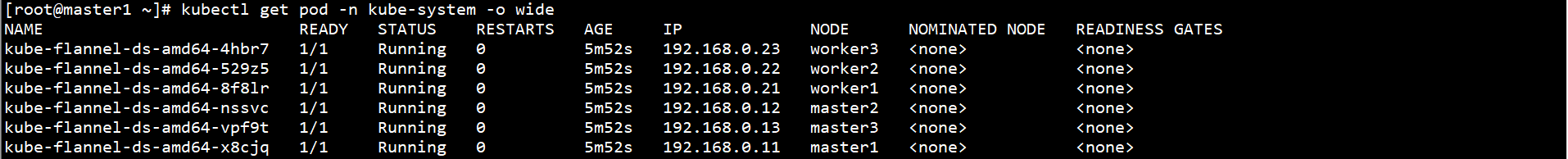


kubectl apply -f apiserver-to-kubelet-rbac.yaml



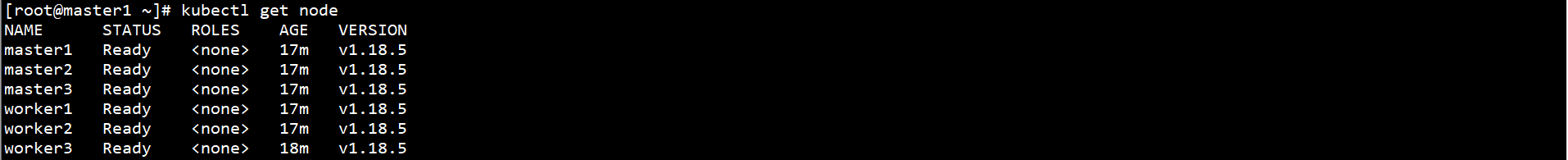
在任意Master节点上查看Pod状态：

kubectl get pod -n kube-system



在任意Master节点上查看Node状态：

kubectl get node



### 7、部署CoreDNS

在Master1节点上解压kubernetes-src.tar.gz文件：

tar -xf /root/kubernetes/kubernetes-src.tar.gz -C /root/kubernetes/



修改/root/kubernetes/cluster/addons/dns/coredns/transforms2sed.sed文件中$DNS\_SERVER\_IP、$DNS\_DOMAIN、$DNS\_MEMORY\_LIMIT参数，如下：

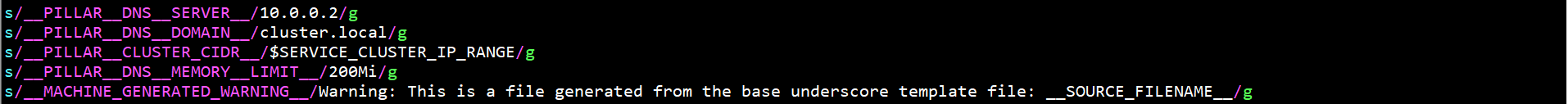
s/\_\_PILLAR\_\_DNS\_\_SERVER\_\_/10.0.0.2/g

s/\_\_PILLAR\_\_DNS\_\_DOMAIN\_\_/cluster.local/g

s/\_\_PILLAR\_\_CLUSTER\_CIDR\_\_/$SERVICE\_CLUSTER\_IP\_RANGE/g

s/\_\_PILLAR\_\_DNS\_\_MEMORY\_\_LIMIT\_\_/200Mi/g

s/\_\_MACHINE\_GENERATED\_WARNING\_\_/Warning: This is a file generated from the base underscore template file: \_\_SOURCE\_FILENAME\_\_/g



使用模板文件生成CoreDNS配置文件coredns.yaml：

cd /root/kubernetes/cluster/addons/dns/coredns/



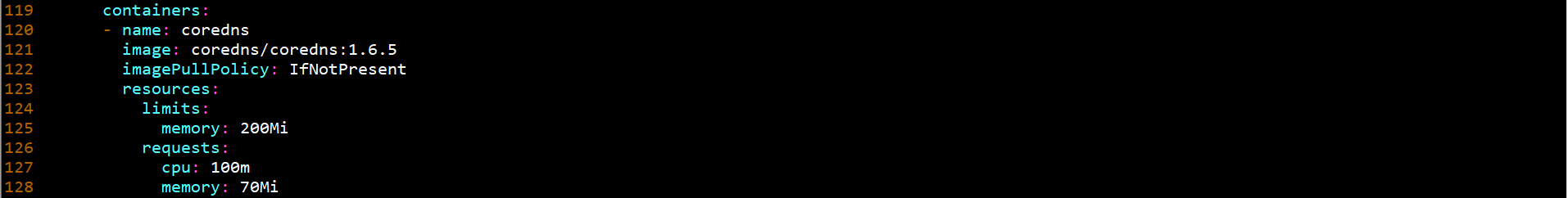
sed -f transforms2sed.sed coredns.yaml.base > coredns.yaml



修改CoreDNS配置文件coredns.yaml

修改image部分参数

image: coredns/coredns:1.6.5



删除capabilities部分：

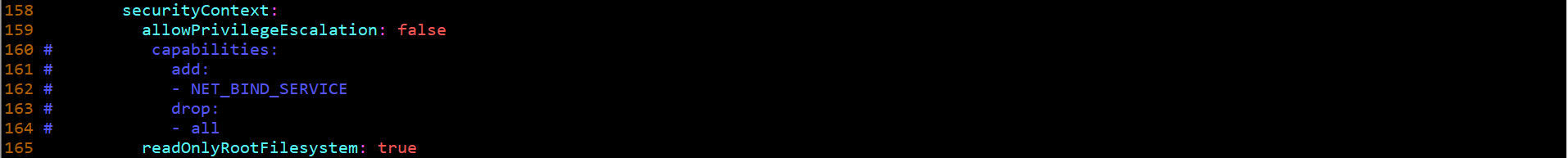
capabilities:

add:

- NET\_BIND\_SERVICE

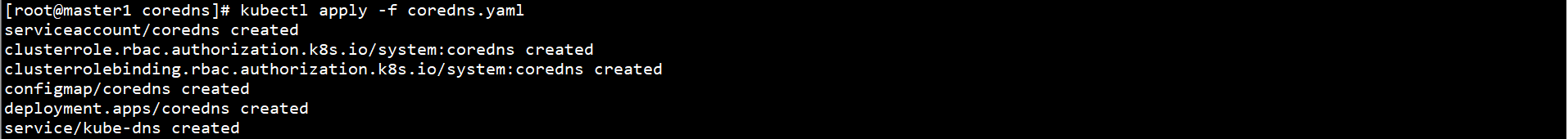
drop:

- all



在Master1节点上部署CoreDNS：

kubectl apply -f coredns.yaml



在任意Master节点上查看Pod状态：

kubectl get pod -n kube-system

