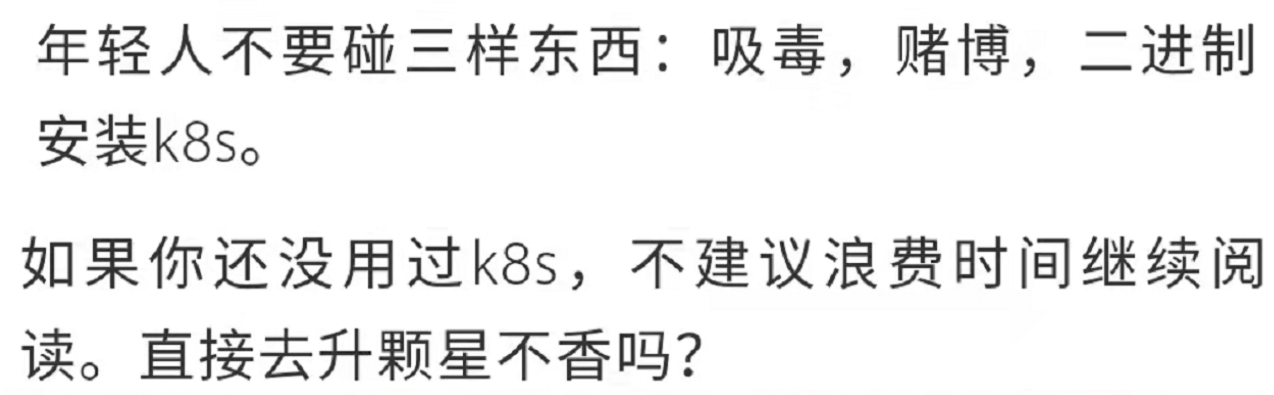
二进制部署高可用k8s集群（1.19）



> 说实在的哈，如果没学过k8s相关知识的，就别看了，劝退指南，先用kubeadm安装学吧。

1.环境初始化

规划

角色	IP	组件
master1	192.168.16.120	kube-apiserver， kube-controller-manager， kube-scheduler，kubelet，kube-proxy，docker，etcd
worker1	192.168.16.130	kubelet，kube-proxy，docker，etcd
worker2	192.168.16.140	kubelet，kube-proxy，docker，etcd

关闭防火墙和selinux

```
systemctl stop firewalld
systemctl disable firewalld

sed -i 's/enforcing/disabled/' /etc/selinux/config # 永久
setenforce 0 # 临时
```

关闭swap

```
swapoff -a # 临时
sed -ri 's/.*swap.*/#&/' /etc/fstab # 永久
```

根据规划设置主机名

```
hostnamectl set-hostname master1 && bash
hostnamectl set-hostname worker1 && bash
hostnamectl set-hostname worker2 && bash
```

在master添加host

```
cat >> /etc/hosts << EOF
192.168.16.120 master1
192.168.16.130 worker1
192.168.16.140 worker2
EOF
```

将桥接的 IPv4 流量传递到 iptables 的链

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

时间同步

```
yum install ntpdate -y
ntpdate time.windows.com
```

2.部署Etcd集群

Etcd 是一个分布式键值存储系统，Kubernetes 使用 Etcd 进行数据存储，所以先准备一个 Etcd 数据库，为解决 Etcd 单点故障，应采用集群方式部署，这里使用 3 台组建集群，可容忍 1 台机器故障，也可以使用 5 台组建集群，可容忍 2 台机器故障。

注：为了节省机器，这里与 K8s 节点机器复用。也可以独立于 k8s 集群之外部署， 只要 apiserver 能连接到就行。

节点名称	IP
etcd-1	192.168.16.120
etcd-2	192.168.16.130
etcd-3	192.168.16.140

2.1 准备 cfssl 证书生成工具

cfssl 是一个开源的证书管理工具，使用 json 文件生成证书，相比 openssl 更方 便使用。

```
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
wget https://pkg.cfssl.org/R1.2/cfssl-certinfo_linux-amd64 -O /usr/local/bin/cfssl-
certinfo
chmod +x /usr/local/bin/cfssl*
```

2.2 生成 Etcd 证书

（1）.自签证书颁发机构（CA）

创建目录，TLS目录用于保存生成的证书

```
mkdir -p /opt/TLS/{etcd,k8s}
cd /opt/TLS/etcd
```

自签CA

```
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
```

生成证书

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

会生成ca.pem和ca-key.pem文件。

（2）.使用自签CA签发Etcd HTTPS证书

创建证书申请文件:

```
cat > server-csr.json << EOF
{
    "CN": "etcd",
    "hosts": [
    "192.168.16.120",
    "192.168.16.130",
    "192.168.16.140",
    "192.168.16.150",
    "192.168.16.160"
    ],
    "key": {
        "algo": "rsa",
        "size": 2048
    },
    "names": [
        {
            "C": "CN",
            "L": "BeiJing",
            "ST": "BeiJing"
        }
    ]
}
EOF
```

注：上述文件hosts字段中IP为所有etcd节点的集群内部通信IP，一个都不能少！为了方便后期扩容可以多写几个预留的IP。

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
```

会生成server.pem和server-key.pem文件。

2.3从GitHub下载二进制文件

https://github.com/etcd-io/etcd/releases/download/v3.4.9/etcd-v3.4.9-linux-amd64.tar.gz

2.4部署Etcd集群

所有操作在master1进行，然后拷贝到其他节点即可

（1）.创建工作目录并解压二进制包

```
mkdir /opt/etcd/{bin,cfg,ssl} -p
tar zxvf etcd-v3.4.9-linux-amd64.tar.gz
mv etcd-v3.4.9-linux-amd64/{etcd,etcdctl} /opt/etcd/bin/
ln -s /opt/etcd/bin/etcd* /usr/local/bin
mkdir /data/etcd/default.etcd -p
```

（2）.创建Etcd配置文件

```
cat > /opt/etcd/cfg/etcd.conf << EOF
#[Member]
ETCD_NAME="etcd-1"
ETCD_DATA_DIR="/data/etcd/default.etcd"
ETCD_LISTEN_PEER_URLS="https://192.168.16.120:2380"
ETCD_LISTEN_CLIENT_URLS="https://192.168.16.120:2379"

#[Clustering]
ETCD_INITIAL_ADVERTISE_PEER_URLS="https://192.168.16.120:2380"
ETCD_ADVERTISE_CLIENT_URLS="https://192.168.16.120:2379"
ETCD_INITIAL_CLUSTER="etcd-1=https://192.168.16.120:2380,etcd-
2=https://192.168.16.130:2380,etcd-3=https://192.168.16.140:2380"
ETCD_INITIAL_CLUSTER_TOKEN="etcd-cluster"
ETCD_INITIAL_CLUSTER_STATE="new"
EOF
```

• 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_PEERURLS：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_INITIALCLUSTER_TOKEN：集群Token

• ETCD_INITIALCLUSTER_STATE：加入集群的当前状态，new是新集群，existing表示加入已有集群

（3）.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=/opt/etcd/cfg/etcd.conf
ExecStart=/opt/etcd/bin/etcd \
--cert-file=/opt/etcd/ssl/server.pem \
--key-file=/opt/etcd/ssl/server-key.pem \
--peer-cert-file=/opt/etcd/ssl/server.pem \
--peer-key-file=/opt/etcd/ssl/server-key.pem \
--trusted-ca-file=/opt/etcd/ssl/ca.pem \
--peer-trusted-ca-file=/opt/etcd/ssl/ca.pem \
--logger=zap
Restart=on-failure
LimitNOFILE=65536

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

（4）.拷贝刚才生成的etcd证书

```
cp /opt/TLS/etcd/ca*pem /opt/TLS/etcd/server*pem /opt/etcd/ssl/
```

（5）.将etcd目录和etcd.service拷贝到其它节点

```
scp -r /opt/etcd/ root@worker1:/opt/
scp -r /opt/etcd/ root@worker2:/opt/
scp /usr/lib/systemd/system/etcd.service root@worker1:/usr/lib/systemd/system/
scp /usr/lib/systemd/system/etcd.service root@worker2:/usr/lib/systemd/system/
```

在worker节点修改etcd.conf配置文件中的节点名称和当前服务器IP

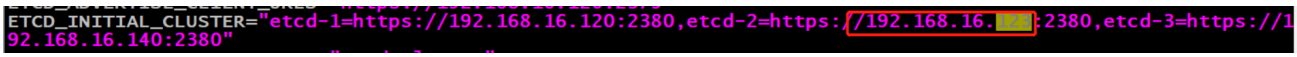
```
vim /opt/etcd/cfg/etcd.conf
```

（6）.启动

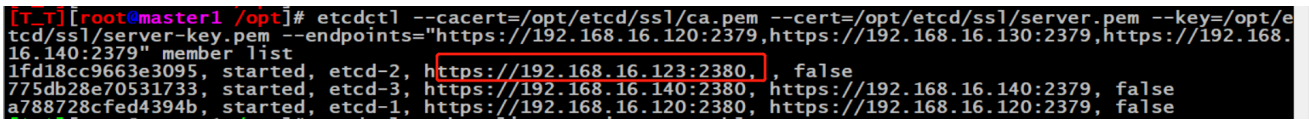
```
systemctl daemon-reload
systemctl start etcd
systemctl enable etcd
systemctl status etcd
```

如果你起来了，这个，没必要看

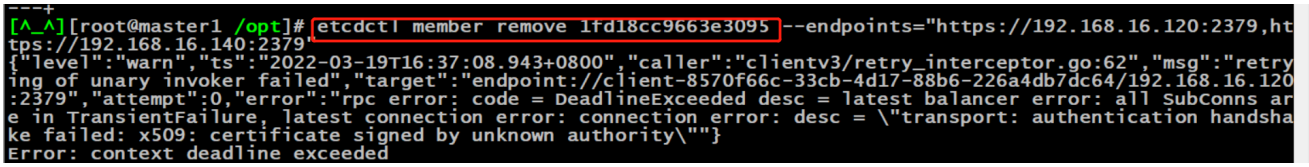
这里我报错了，因为我etcd-2的ip地址写错了



就是这个，然后启动的时候第二台报错了，在master1看etcd集群是这样子

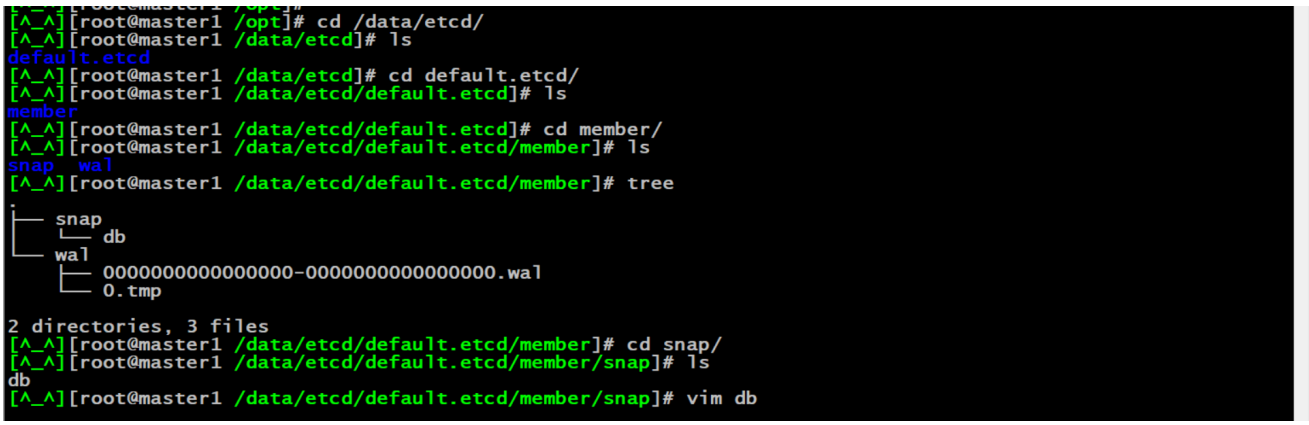


因为192.168.16.130没有指定加入到集群，所以起不来，后来尝试打算从集群中删除这个节点重新加入，上图的第一列那个1fd18cc9663e3095就是节点的id号，通过这个删除



这个的话，就是删除的命令，后面的endpoints是你要留下的节点，不过

呃。。好像没删掉，后来就想着去数据目录去看下，



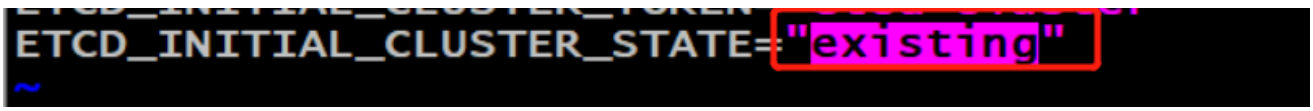
就这个文本，这个的话，二进制的，看不了

不过，有一点能看的，可以看到192.168.16.123这个ip还在，那可不就是没删掉么

然后我寻思。。。直接把这个目录删了得了，重新创建集群，然后就删了。。

然后用上面的启动命令，worker1还是起不来，

我又去worker1的etcd.conf文件看，新加入集群的话要指定这个，试一下

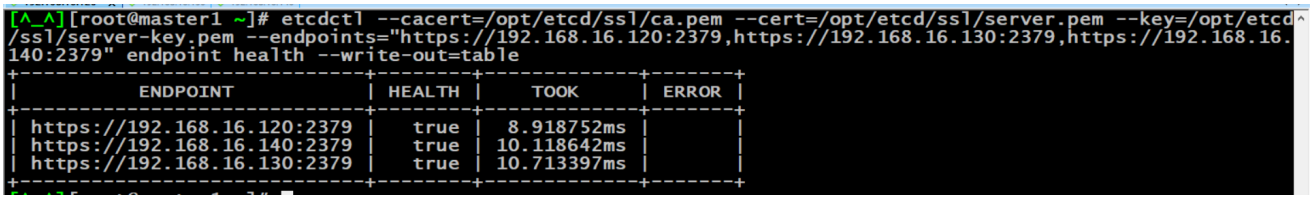


然后就好了。。。（话说我不是把集群目录都删了么。。不懂）

（7）.查看集群相关信息

```
etcdctl --cacert=/opt/etcd/ssl/ca.pem --cert=/opt/etcd/ssl/server.pem --
key=/opt/etcd/ssl/server-key.pem --
endpoints="https://192.168.16.120:2379,https://192.168.16.130:2379,https://192.168.16.1
40:2379" endpoint status --write-out=table

member list
```



终于搭好了。

3.安装docker

docker在这里是作为k8s的容器引擎，也可以使用containerd

下载

https://download.docker.com/linux/static/stable/x86_64/docker-20.10.3.tgz

所有节点都安装

3.1 解压包

```
tar zxvf docker-20.10.3.tgz
mv docker/* /usr/bin/
```

3.2 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/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
```

3.3 创建配置文件

这个是阿里云的镜像仓库，拉镜像会比国外的站点快

```
mkdir /etc/docker
cat > /etc/docker/daemon.json << EOF
{
  "registry-mirrors": ["https://b9pmyelo.mirror.aliyuncs.com"]
}
EOF
```

3.4 启动

```
systemctl daemon-reload
systemctl start docker
systemctl enable docker
systemctl status docker
```

4.部署Master节点

4.1 生成kube-apiserver证书

（1）自签证书颁发机构（CA）

```
cd /opt/TLS/k8s

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
```

生成证书

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

会生成ca.pem和ca-key.pem文件。

（2）使用自签CA签发kube-apiserver HTTPS证书

创建证书申请文件

```
cat > server-csr.json << EOF
{
    "CN": "kubernetes",
    "hosts": [
      "10.0.0.1",
      "10.254.0.1",
      "127.0.0.1",
      "192.168.16.120",
      "192.168.16.130",
      "192.168.16.140",
      "192.168.16.150",
      "192.168.16.160",
      "192.168.16.170",
      "192.168.16.180",
      "192.168.16.16",
      "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
```

注：上述文件hosts字段中IP为所有Master/LB/VIP IP，一个都不能少！为了方便后期扩容可以多写几个预留的IP。

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
```

会生成server.pem和server-key.pem文件。

4.2 从Github下载二进制文件

下载地址

https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.20.md

里面有很多包，下载一个server包就够了，包含了Master和Worker Node二进制文件。

4.3 解包

```
mkdir -p /opt/kubernetes/{bin,cfg,ssl,logs}
tar zxvf kubernetes-server-linux-amd64.tar.gz
cd kubernetes/server/bin
cp kube-apiserver kube-scheduler kube-controller-manager kubelet kube-proxy
/opt/kubernetes/bin
cp kubectl /usr/bin/
```

4.4 kube-apiserver

（1）创建配置文件

```
cat > /opt/kubernetes/cfg/kube-apiserver.conf << EOF
KUBE_APISERVER_OPTS="--logtostderr=false \\
--v=2 \\
--log-dir=/opt/kubernetes/logs \\
```

```
--etcd-
servers=https://192.168.16.120:2379,https://192.168.16.130:2379,https://192.168.16.140:
2379 \\
--bind-address=192.168.16.120 \\
--secure-port=6443 \\
--advertise-address=192.168.16.120 \\
--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=/opt/kubernetes/cfg/token.csv \\
--service-node-port-range=30000-32767 \\
--kubelet-client-certificate=/opt/kubernetes/ssl/server.pem \\
--kubelet-client-key=/opt/kubernetes/ssl/server-key.pem \\
--tls-cert-file=/opt/kubernetes/ssl/server.pem  \\
--tls-private-key-file=/opt/kubernetes/ssl/server-key.pem \\
--client-ca-file=/opt/kubernetes/ssl/ca.pem \\
--service-account-key-file=/opt/kubernetes/ssl/ca-key.pem \\
--service-account-issuer=api \\
--service-account-signing-key-file=/opt/kubernetes/ssl/server-key.pem \\
--etcd-cafile=/opt/etcd/ssl/ca.pem \\
--etcd-certfile=/opt/etcd/ssl/server.pem \\
--etcd-keyfile=/opt/etcd/ssl/server-key.pem \\
--requestheader-client-ca-file=/opt/kubernetes/ssl/ca.pem \\
--proxy-client-cert-file=/opt/kubernetes/ssl/server.pem \\
--proxy-client-key-file=/opt/kubernetes/ssl/server-key.pem \\
--requestheader-allowed-names=kubernetes \\
--requestheader-extra-headers-prefix=X-Remote-Extra- \\
--requestheader-group-headers=X-Remote-Group \\
--requestheader-username-headers=X-Remote-User \\
--enable-aggregator-routing=true \\
--audit-log-maxage=30 \\
--audit-log-maxbackup=3 \\
--audit-log-maxsize=100 \\
--audit-log-path=/opt/kubernetes/logs/k8s-audit.log"
EOF
```

注：上面两个\ \ 第一个是转义符，第二个是换行符，使用转义符是为了使用EOF保留换行符。

• --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：apiserver访问kubelet客户端证书

• --tls-xxx-file：apiserver https证书

• 1.20版本必须加的参数：--service-account-issuer，--service-account-signing-key-file

• --etcd-xxxfile：连接Etcd集群证书

• --audit-log-xxx：审计日志

启动聚合层相关配置：--requestheader-client-ca-file，--proxy-client-cert-file，--proxy-client-key-file，--requestheader-allowed-names，--requestheader-extra-headers-prefix，--requestheader-group-headers，--requestheader-username-headers，--enable-aggregator-routing

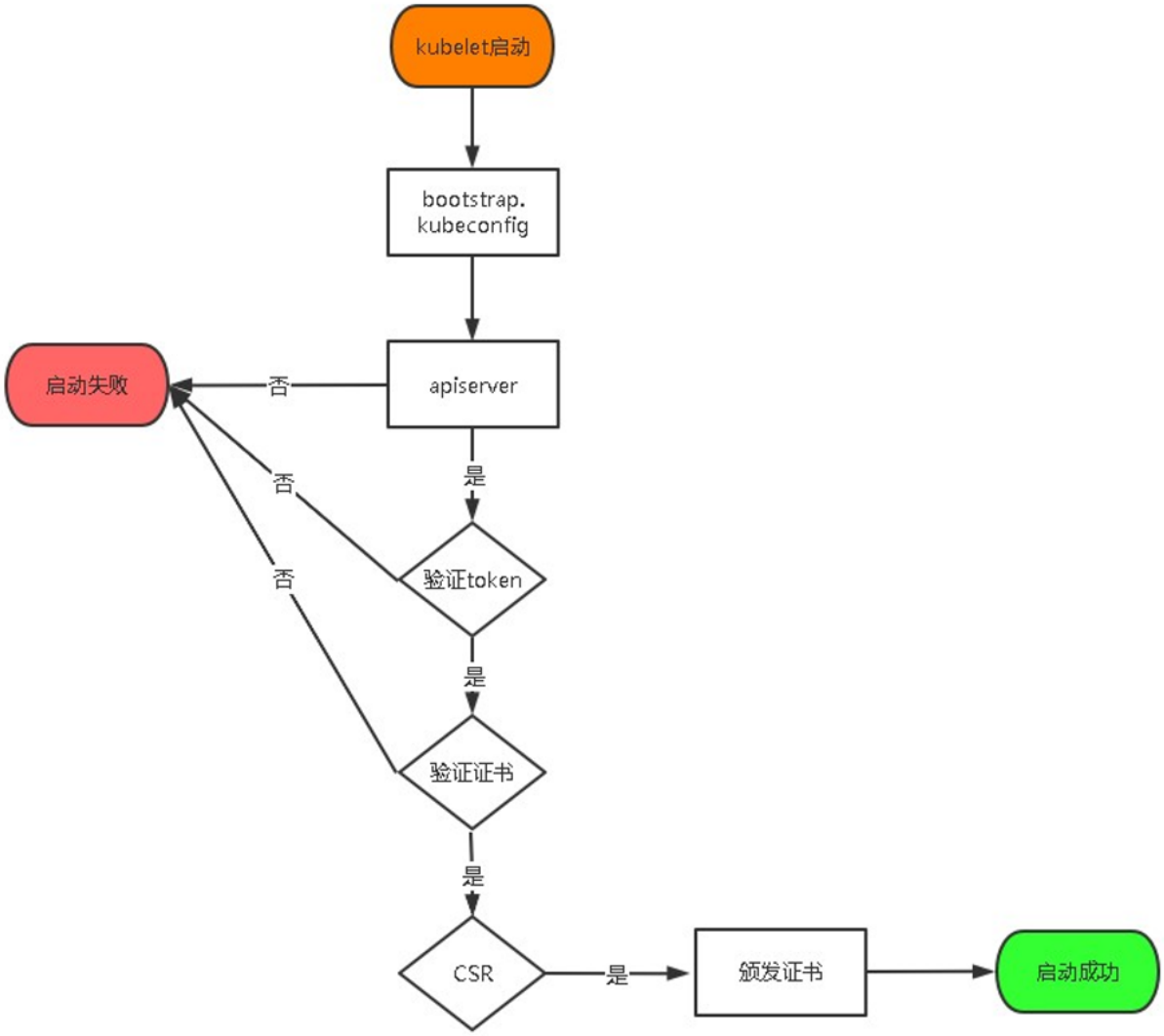
（2）拷贝之前生成的证书

```
cp /opt/TLS/k8s/*.pem /opt/kubernetes/ssl/
```

（3）启用 TLS Bootstrapping 机制

TLS Bootstraping：Master apiserver启用TLS认证后，Node节点kubelet和kube-proxy要与kube-apiserver进行通信，必须使用CA签发的有效证书才可以，当Node节点很多时，这种客户端证书颁发需要大量工作，同样也会增加集群扩展复杂度。为了简化流程，Kubernetes引入了TLS bootstraping机制来自动颁发客户端证书，kubelet会以一个低权限用户自动向apiserver申请证书，kubelet的证书由apiserver动态签署。所以强烈建议在Node上使用这种方式，目前主要用于kubelet，kube-proxy还是由我们统一颁发一个证书。

TLS bootstraping 工作流程：



创建上述配置文件中token文件:

```
cat > /opt/kubernetes/cfg/token.csv << EOF
f1332c0ceab734ae3e528816362926a0,kubelet-bootstrap,10001,"system:node-bootstrapper"
EOF
```

格式：token，用户名，UID，用户组

token也可自行生成替换:

```

```

（4）systemd管理apiserver

```
cat > /usr/lib/systemd/system/kube-apiserver.service << EOF
[Unit]
Description=Kubernetes API Server
Documentation=https://github.com/kubernetes/kubernetes

[Service]
```

```
EnvironmentFile=/opt/kubernetes/cfg/kube-apiserver.conf
ExecStart=/opt/kubernetes/bin/kube-apiserver \$KUBE_APISERVER_OPTS
Restart=on-failure

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

（5）启动

```
systemctl daemon-reload
systemctl start kube-apiserver
systemctl enable kube-apiserver
systemctl status kube-apiserver
```

4.5 kube-controller-manager

（1）创建配置文件

```
cat > /opt/kubernetes/cfg/kube-controller-manager.conf << EOF
KUBE_CONTROLLER_MANAGER_OPTS="--logtostderr=false \\
--v=2 \\
--log-dir=/opt/kubernetes/logs \\
--leader-elect=true \\
--kubeconfig=/opt/kubernetes/cfg/kube-controller-manager.kubeconfig \\
--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=/opt/kubernetes/ssl/ca.pem \\
--cluster-signing-key-file=/opt/kubernetes/ssl/ca-key.pem  \\
--root-ca-file=/opt/kubernetes/ssl/ca.pem \\
--service-account-private-key-file=/opt/kubernetes/ssl/ca-key.pem \\
--cluster-signing-duration=87600h0m0s"
EOF
```

• --kubeconfig：连接apiserver配置文件

• --leader-elect：当该组件启动多个时，自动选举（HA）

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

（2）生成kubeconfig文件

生成kube-controller-manager证书：

```
# 切换工作目录
cd /opt/TLS/k8s
```

```
# 创建证书请求文件
cat > kube-controller-manager-csr.json << EOF
{
  "CN": "system:kube-controller-manager",
  "hosts": [],
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "CN",
      "L": "BeiJing",
      "ST": "BeiJing",
      "O": "system:masters",
      "OU": "System"
    }
  ]
}
EOF

# 生成证书
cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=kubernetes
kube-controller-manager-csr.json | cfssljson -bare kube-controller-manager
```

生成kubeconfig文件：

```
KUBE_CONFIG="/opt/kubernetes/cfg/kube-controller-manager.kubeconfig"
KUBE_APISERVER="https://192.168.16.120:6443"

kubectl config set-cluster kubernetes \
  --certificate-authority=/opt/kubernetes/ssl/ca.pem \
  --embed-certs=true \
  --server=${KUBE_APISERVER} \
  --kubeconfig=${KUBE_CONFIG}
kubectl config set-credentials kube-controller-manager \
  --client-certificate=./kube-controller-manager.pem \
  --client-key=./kube-controller-manager-key.pem \
  --embed-certs=true \
  --kubeconfig=${KUBE_CONFIG}
kubectl config set-context default \
  --cluster=kubernetes \
  --user=kube-controller-manager \
  --kubeconfig=${KUBE_CONFIG}
kubectl config use-context default --kubeconfig=${KUBE_CONFIG}
```

（3）systemd管理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=/opt/kubernetes/cfg/kube-controller-manager.conf
ExecStart=/opt/kubernetes/bin/kube-controller-manager \$KUBE_CONTROLLER_MANAGER_OPTS
Restart=on-failure

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

（4）启动

```
systemctl daemon-reload
systemctl start kube-controller-manager
systemctl enable kube-controller-manager
systemctl status kube-controller-manager
```

4.6 kube-scheduler

（1）创建配置文件

```
cat > /opt/kubernetes/cfg/kube-scheduler.conf << EOF
KUBE_SCHEDULER_OPTS="--logtostderr=false \\
--v=2 \\
--log-dir=/opt/kubernetes/logs \\
--leader-elect \\
--kubeconfig=/opt/kubernetes/cfg/kube-scheduler.kubeconfig \\
--bind-address=127.0.0.1"
EOF
```

• --kubeconfig：连接apiserver配置文件

• --leader-elect：当该组件启动多个时，自动选举（HA）

（2）生成kubeconfig文件

生成kube-scheduler证书：

```
# 切换工作目录
cd /opt/TLS/k8s

# 创建证书请求文件
cat > kube-scheduler-csr.json << EOF
{
  "CN": "system:kube-scheduler",
  "hosts": [],
  "key": {
    "algo": "rsa",
```

```
    "size": 2048
  },
  "names": [
    {
      "C": "CN",
      "L": "BeiJing",
      "ST": "BeiJing",
      "O": "system:masters",
      "OU": "System"
    }
  ]
}
EOF

# 生成证书
cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=kubernetes
kube-scheduler-csr.json | cfssljson -bare kube-scheduler
```

生成kubeconfig文件：

```
KUBE_CONFIG="/opt/kubernetes/cfg/kube-scheduler.kubeconfig"
KUBE_APISERVER="https://192.168.16.120:6443"

kubectl config set-cluster kubernetes \
  --certificate-authority=/opt/kubernetes/ssl/ca.pem \
  --embed-certs=true \
  --server=${KUBE_APISERVER} \
  --kubeconfig=${KUBE_CONFIG}
kubectl config set-credentials kube-scheduler \
  --client-certificate=./kube-scheduler.pem \
  --client-key=./kube-scheduler-key.pem \
  --embed-certs=true \
  --kubeconfig=${KUBE_CONFIG}
kubectl config set-context default \
  --cluster=kubernetes \
  --user=kube-scheduler \
  --kubeconfig=${KUBE_CONFIG}
kubectl config use-context default --kubeconfig=${KUBE_CONFIG}
```

（3）systemd管理scheduler

```
cat > /usr/lib/systemd/system/kube-scheduler.service << EOF
[Unit]
Description=Kubernetes Scheduler
Documentation=https://github.com/kubernetes/kubernetes

[Service]
EnvironmentFile=/opt/kubernetes/cfg/kube-scheduler.conf
ExecStart=/opt/kubernetes/bin/kube-scheduler \$KUBE_SCHEDULER_OPTS
Restart=on-failure
```

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

（4）systemd管理scheduler

```
systemctl daemon-reload
systemctl start kube-scheduler
systemctl enable kube-scheduler
systemctl status kube-scheduler
```

4.7 kubectl

（1）生成kubectl连接集群的证书:

```
cat > /opt/TLS/k8s/admin-csr.json <<EOF
{
  "CN": "admin",
  "hosts": [],
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "CN",
      "L": "BeiJing",
      "ST": "BeiJing",
      "O": "system:masters",
      "OU": "System"
    }
  ]
}
EOF

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

（2）生成kubeconfig文件:

```
mkdir /root/.kube

KUBE_CONFIG="/root/.kube/config"
KUBE_APISERVER="https://192.168.16.120:6443"

kubectl config set-cluster kubernetes \
  --certificate-authority=/opt/kubernetes/ssl/ca.pem \
  --embed-certs=true \
  --server=${KUBE_APISERVER} \
```

```
  --kubeconfig=${KUBE_CONFIG}
kubectl config set-credentials cluster-admin \
  --client-certificate=./admin.pem \
  --client-key=./admin-key.pem \
  --embed-certs=true \
  --kubeconfig=${KUBE_CONFIG}
kubectl config set-context default \
  --cluster=kubernetes \
  --user=cluster-admin \
  --kubeconfig=${KUBE_CONFIG}
kubectl config use-context default --kubeconfig=${KUBE_CONFIG}
```

通过kubectl工具查看当前集群组件状态:

```
kubectl get cs
```

```
[^_^][root@master1 /opt/TLS/k8s]# kubectl get cs
Warning: v1 ComponentStatus is deprecated in v1.19+
NAME                 STATUS    MESSAGE             ERROR
controller-manager   Healthy   ok
scheduler            Healthy   ok
etcd-0               Healthy   {"health":"true"}
etcd-2               Healthy   {"health":"true"}
etcd-1               Healthy   {"health":"true"}
```

如上输出说明Master节点组件运行正常。

（3）授权kubelet-bootstrap用户允许请求证书

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

5. 部署Worker Node

以下操作在worker节点，如果要把master也作为工作节点的话，有些证书就不用拷贝了。

5.1 创建工作目录并拷贝二进制文件

```
mkdir -p /opt/kubernetes/{bin,cfg,ssl,logs}
scp -r /root/kubernetes/server/bin/kubectl /opt/kubernetes/bin/kubelet
/opt/kubernetes/bin/kube-proxy root@worker2:/opt/kubernetes/bin/                #拷贝可执
行文件，master1执行

cp -p /opt/kubernetes/bin/kubectl /usr/bin/
```

5.2 部署kubelet

（1）创建配置文件

```
scp -r /opt/TLS/k8s/ca.pem root@worker2:/opt/kubernetes/ssl/    #拷贝证书文件，master1执行


cat > /opt/kubernetes/cfg/kubelet.conf << EOF
KUBELET_OPTS="--logtostderr=false \\
--v=2 \\
--log-dir=/opt/kubernetes/logs \\
--hostname-override=worker2 \\
--network-plugin=cni \\
--kubeconfig=/opt/kubernetes/cfg/kubelet.kubeconfig \\
--bootstrap-kubeconfig=/opt/kubernetes/cfg/bootstrap.kubeconfig \\
--config=/opt/kubernetes/cfg/kubelet-config.yml \\
--cert-dir=/opt/kubernetes/ssl \\
--pod-infra-container-image=google/pause:latest"
EOF
```

• --hostname-override：显示名称，集群中唯一

• --network-plugin：启用CNI

• --kubeconfig：空路径，会自动生成，后面用于连接apiserver

• --bootstrap-kubeconfig：首次启动向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

（2）配置参数文件

```
cat > /opt/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: /opt/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
```

(3) 生成kubelet初次加入集群引导kubeconfig文件

```
KUBE_CONFIG="/opt/kubernetes/cfg/bootstrap.kubeconfig"
KUBE_APISERVER="https://192.168.16.120:6443" # apiserver IP:PORT
TOKEN="f1332c0ceab734ae3e528816362926a0" # 与master1的token.csv里的保持一致

# 生成 kubelet bootstrap kubeconfig 配置文件
kubectl config set-cluster kubernetes \
  --certificate-authority=/opt/kubernetes/ssl/ca.pem \
  --embed-certs=true \
  --server=${KUBE_APISERVER} \
  --kubeconfig=${KUBE_CONFIG}
kubectl config set-credentials "kubelet-bootstrap" \
  --token=${TOKEN} \
  --kubeconfig=${KUBE_CONFIG}
kubectl config set-context default \
  --cluster=kubernetes \
  --user="kubelet-bootstrap" \
  --kubeconfig=${KUBE_CONFIG}
kubectl config use-context default --kubeconfig=${KUBE_CONFIG}
```

(4) systemd管理kubelet

```
cat > /usr/lib/systemd/system/kubelet.service << EOF
[Unit]
Description=Kubernetes Kubelet
After=docker.service

[Service]
EnvironmentFile=/opt/kubernetes/cfg/kubelet.conf
ExecStart=/opt/kubernetes/bin/kubelet \$KUBELET_OPTS
Restart=on-failure
LimitNOFILE=65536

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

(5) 启动

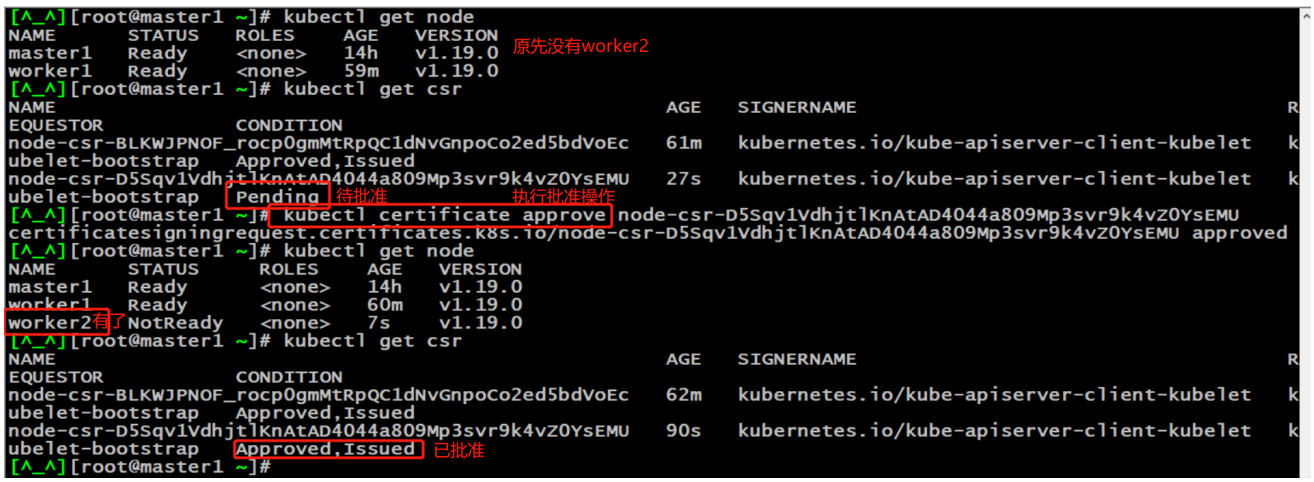
```
systemctl daemon-reload
systemctl start kubelet
systemctl enable kubelet
systemctl status kubelet
```

(6) 批准kubelet证书申请并加入集群

```
# 查看kubelet证书请求
kubectl get csr
NAME                                                   AGE    SIGNERNAME
REQUESTOR           CONDITION
node-csr-D5Sqv1VdhjtlKnAtAD4044a809Mp3svr9k4vZOYsEMU   27s    kubernetes.io/kube-
apiserver-client-kubelet   kubelet-bootstrap   Pending

# 批准申请
kubectl certificate approve node-csr-D5Sqv1VdhjtlKnAtAD4044a809Mp3svr9k4vZOYsEMU

# 查看节点
kubectl get node
NAME      STATUS     ROLES    AGE    VERSION
worker2   NotReady   <none>   7s     v1.19.0
```



注：由于网络插件还没有部署，节点会没有准备就绪 NotReady

5.3 部署kube-proxy

(1) 创建配置文件

```
cat > /opt/kubernetes/cfg/kube-proxy.conf << EOF
KUBE_PROXY_OPTS="--logtostderr=false \\
--v=2 \\
--log-dir=/opt/kubernetes/logs \\
--config=/opt/kubernetes/cfg/kube-proxy-config.yml"
EOF
```

（2）配置参数文件

```
cat > /opt/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: /opt/kubernetes/cfg/kube-proxy.kubeconfig
hostnameOverride: worker2
clusterCIDR: 10.244.0.0/16
EOF
```

（3）生成kube-proxy.kubeconfig文件

(在master1执行，然后拷贝到节点)

```
# 切换工作目录
cd /opt/TLS/k8s

# 创建证书请求文件
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

# 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

scp -r /opt/TLS/k8s/kube-proxy*pem root@worker2:/opt/kubernetes/ssl/    #拷贝到节点
```

生成kubeconfig文件:

```
KUBE_CONFIG="/opt/kubernetes/cfg/kube-proxy.kubeconfig"
```

```
KUBE_APISERVER="https://192.168.16.120:6443"

kubectl config set-cluster kubernetes \
  --certificate-authority=/opt/kubernetes/ssl/ca.pem \
  --embed-certs=true \
  --server=${KUBE_APISERVER} \
  --kubeconfig=${KUBE_CONFIG}
kubectl config set-credentials kube-proxy \
  --client-certificate=/opt/kubernetes/ssl/kube-proxy.pem \
  --client-key=/opt/kubernetes/ssl/kube-proxy-key.pem \
  --embed-certs=true \
  --kubeconfig=${KUBE_CONFIG}
kubectl config set-context default \
  --cluster=kubernetes \
  --user=kube-proxy \
  --kubeconfig=${KUBE_CONFIG}
kubectl config use-context default --kubeconfig=${KUBE_CONFIG}
```

（4）systemd管理kube-proxy

```
cat > /usr/lib/systemd/system/kube-proxy.service << EOF
[Unit]
Description=Kubernetes Proxy
After=network.target

[Service]
EnvironmentFile=/opt/kubernetes/cfg/kube-proxy.conf
ExecStart=/opt/kubernetes/bin/kube-proxy \$KUBE_PROXY_OPTS
Restart=on-failure
LimitNOFILE=65536

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

（5）启动

```
systemctl daemon-reload
systemctl start kube-proxy
systemctl enable kube-proxy
systemctl status kube-proxy
```

5.4 部署cni网络插件（flannel）

容器网络接口（Container Network Interface，CNI）

cni是k8s的网络接口，很多插件都可以实现这个接口

cni

通过给 Kubelet 传递 `--network-plugin=cni` 命令行选项可以选择 CNI 插件。 Kubelet 从 `--cni-conf-dir` （默认是 `/etc/cni/net.d`） 读取文件并使用 该文件中的 CNI 配置来设置各个 Pod 的网络。 CNI 配置文件必须与 CNI 规约 匹配，并且配置所引用的所有所需的 CNI 插件都应存在于 `--cni-bin-dir` （默认是 `/opt/cni/bin`）下。

如果这个目录中有多个 CNI 配置文件，kubelet 将会使用按文件名的字典顺序排列 的第一个作为配置文件。

除了配置文件指定的 CNI 插件外，Kubernetes 还需要标准的 CNI `lo` 插件，最低版本是0.2.0

```
创建cni所需目录
mkdir /opt/cni/bin -p   #可执行文件
mkdir /etc/cni/net.d -p #cni配置信息


tar -zxvf cni-plugins-linux-amd64-v0.8.2.tgz -C /opt/cni/bin/   #解压
```

master1执行

```
kubectl apply -f kube-flannel.yml       #在master部署flannel插件


kubectl get po -A       #查看部署状况
```

部署好之后，所有节点就准备就绪了

```
[^_^][root@master1 ~]# kubectl get node
NAME      STATUS   ROLES    AGE   VERSION
master1   Ready    <none>   15h   v1.19.0
worker1   Ready    <none>   75m   v1.19.0
worker2   Ready    <none>   15m   v1.19.0
```

5.5 授权apiserver访问kubelet

应用场景：例如kubectl logs

```
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
```

6. 部署Dashboard和CoreDNS

6.1 Dashboard（master操作）

```
kubectl apply -f dashboard.yaml #部署

kubectl get pods,svc -n kubernetes-dashboard    #查看信息
```

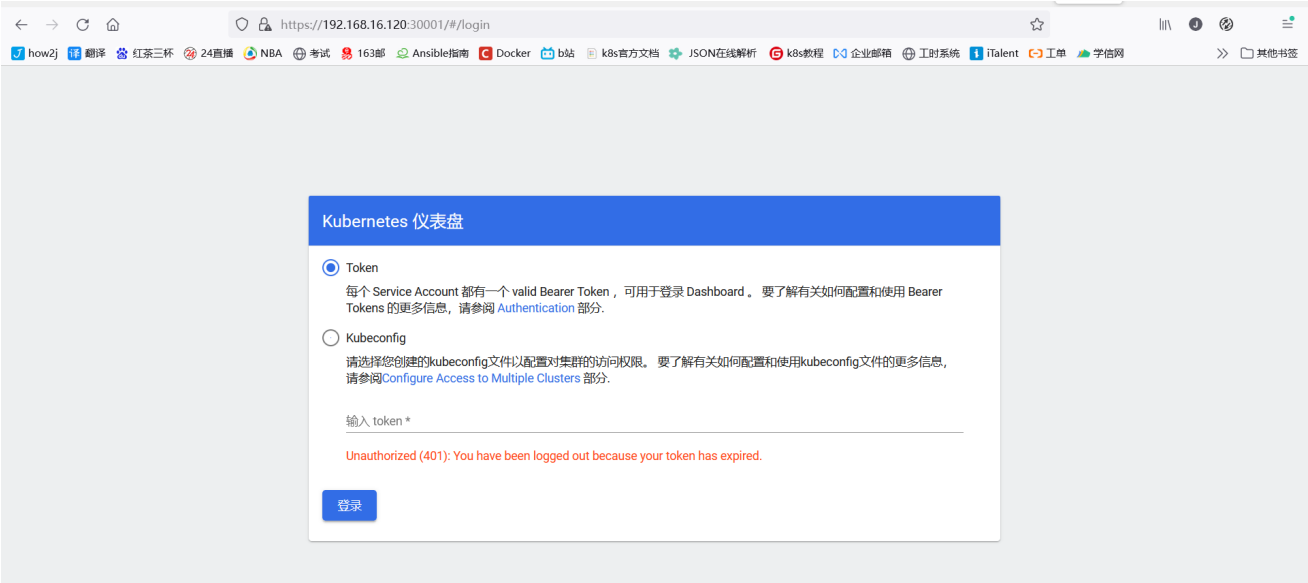
```
[^_^][root@master1 /data/kubernetes/yaml/dashboard]# kubectl get pods,svc -n kubernetes-dashboard
NAME                                        READY   STATUS    RESTARTS   AGE
pod/dashboard-metrics-scraper-7b9b99d599-8xjk8   1/1     Running   0          4m8s
pod/kubernetes-dashboard-6d4799d74-vjjcg         1/1     Running   0          4m8s

NAME                                TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)         AGE
service/dashboard-metrics-scraper   ClusterIP   10.0.0.207    <none>        8000/TCP        4m8s
service/kubernetes-dashboard        NodePort    10.0.0.123    <none>        443:30001/TCP   4m10s
```

通过宿主机ip+端口号访问

https://192.168.16.120:30001

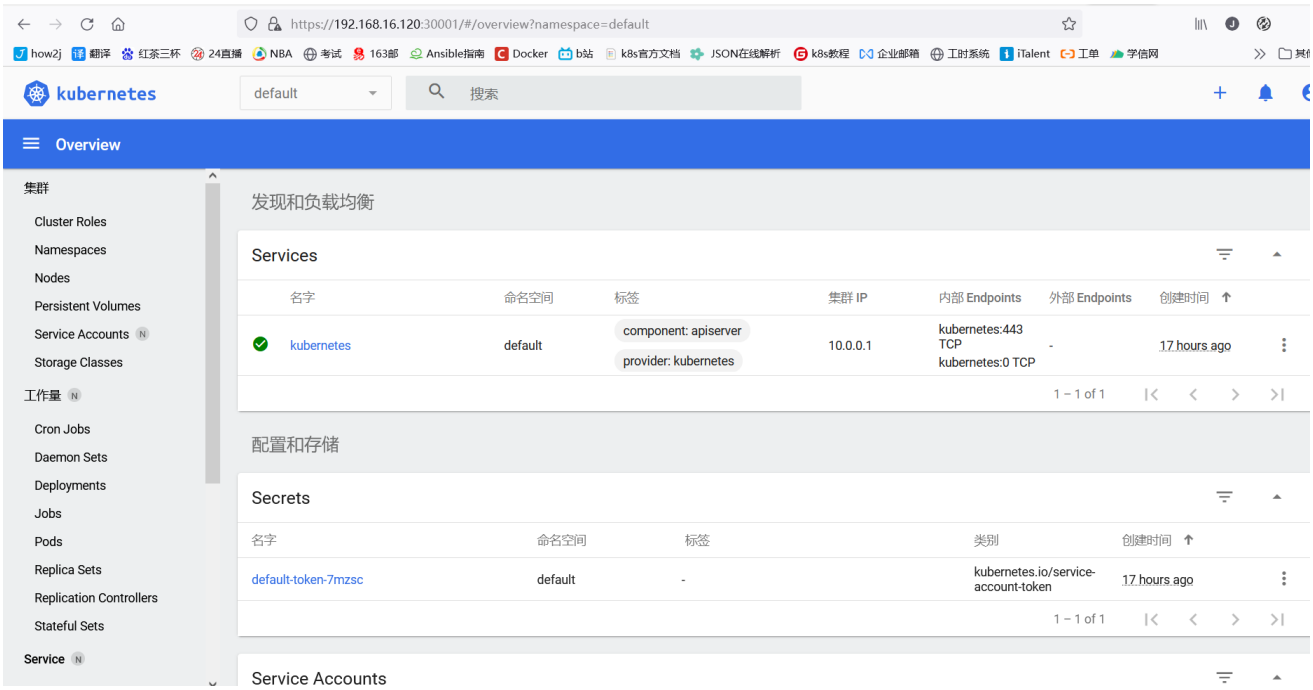
会要求你选择一个登录方式,选择token



创建service account并绑定默认cluster-admin管理员集群角色

```
kubectl create serviceaccount dashboard-admin -n kube-system
kubectl create clusterrolebinding dashboard-admin --clusterrole=cluster-admin --
serviceaccount=kube-system:dashboard-admin
kubectl describe secrets -n kube-system $(kubectl -n kube-system get secret | awk
'/dashboard-admin/{print $1}')
```

使用输出的token登录Dashboard。



6.2 CoreDNS

CoreDNS用于集群内部Service名称解析。

```
kubectl apply -f coredns.yaml
```

```
[^_^][root@master1 /data/kubernetes/yaml/coredns]# kubectl apply -f coredns.yaml
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
```

```
kubectl get pods -n kube-system
```

```
[^_^][root@master1 /data/kubernetes/yaml/coredns]# kubectl get pods -n kube-system
NAME                          READY   STATUS    RESTARTS   AGE
coredns-858f6dbf57-l6t26      1/1     Running   0          25s
kube-flannel-ds-amd64-244rj   1/1     Running   7          13h
kube-flannel-ds-amd64-fd6tj   1/1     Running   1          14h
kube-flannel-ds-amd64-x4577   1/1     Running   2          27h
```

DNS解析测试:

```
kubectl run -it --rm dns-test --image=busybox:1.28.4 sh

/ # nslookup kubernetes
Server:    10.0.0.2
Address 1: 10.0.0.2 kube-dns.kube-system.svc.cluster.local

Name:      kubernetes
Address 1: 10.0.0.1 kubernetes.default.svc.cluster.local
```

```
[^_^][root@master1 /data/kubernetes/yaml]# kubectl run -it --rm dns-test --image=busybox:1.28.4 sh
If you don't see a command prompt, try pressing enter.
/ # nslookup kubernetes
Server:    10.0.0.2
Address 1: 10.0.0.2 kube-dns.kube-system.svc.cluster.local

Name:      kubernetes
Address 1: 10.0.0.1 kubernetes.default.svc.cluster.local
/ # nslookup qq.com
Server:    10.0.0.2
Address 1: 10.0.0.2 kube-dns.kube-system.svc.cluster.local

Name:      qq.com
Address 1: 58.247.214.47
Address 2: 58.250.137.36
/ # nslookup baidu.com
Server:    10.0.0.2
Address 1: 10.0.0.2 kube-dns.kube-system.svc.cluster.local

Name:      baidu.com
Address 1: 220.181.38.148
Address 2: 220.181.38.251
/ #
```

解析没问题。

至此一个单Master集群就搭建完成了！这个环境就足以满足学习实验了，如果你的服务器配置较高，可继续扩容多Master集群。

7. 扩容多Master（高可用架构）

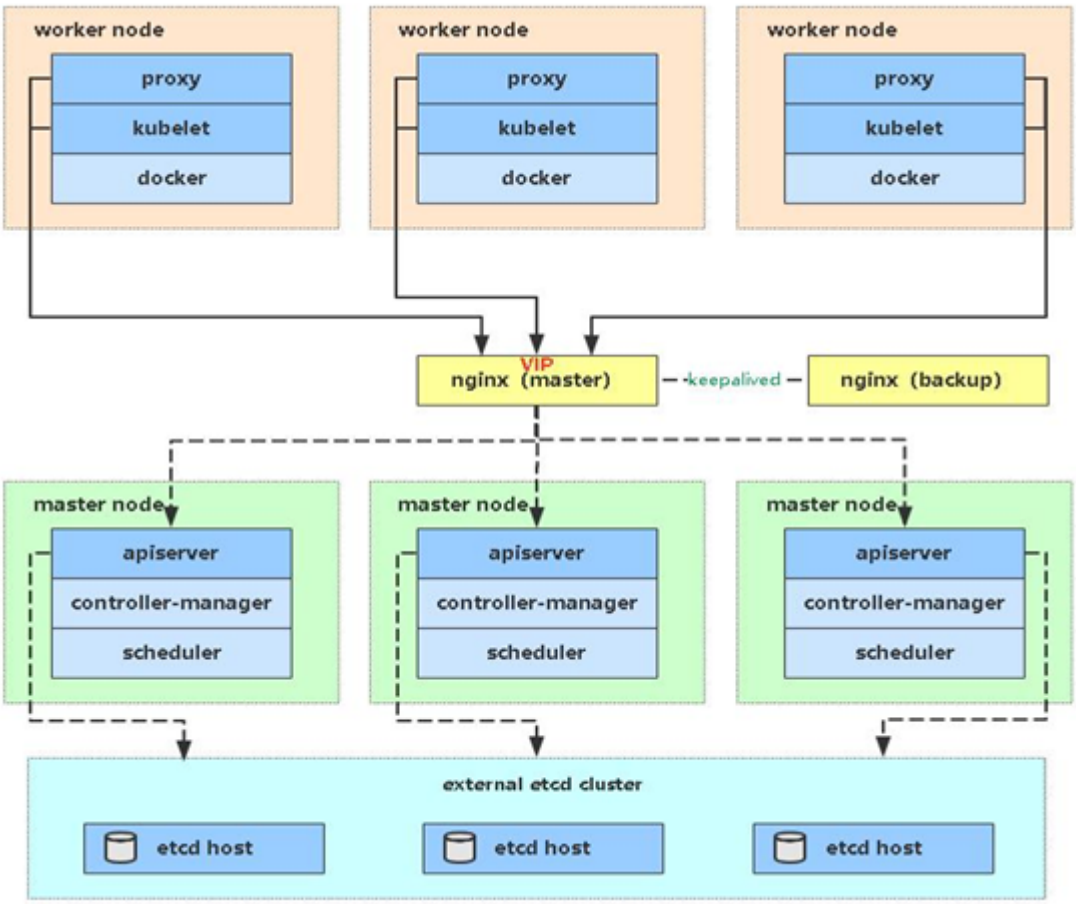
Kubernetes作为容器集群系统，通过健康检查+重启策略实现了Pod故障自我修复能力，通过调度算法实现将Pod分布式部署，并保持预期副本数，根据Node失效状态自动在其他Node拉起Pod，实现了应用层的高可用性。

针对Kubernetes集群，高可用性还应包含以下两个层面的考虑：Etcd数据库的高可用性和Kubernetes Master组件的高可用性。 而Etcd我们已经采用3个节点组建集群实现高可用，所以我们将对Master节点高可用进行说明和实施。

Master节点扮演着总控中心的角色，通过不断与工作节点上的Kubelet和kube-proxy进行通信来维护整个集群的健康工作状态。如果Master节点故障，将无法使用kubectl工具或者API做任何集群管理。

Master节点主要有三个服务kube-apiserver、kube-controller-manager和kube-scheduler，其中kube-controller-manager和kube-scheduler组件自身通过选择机制已经实现了高可用，所以Master高可用主要针对kube-apiserver组件，而该组件是以HTTP API提供服务，因此对他高可用与Web服务器类似，增加负载均衡器对其负载均衡即可，并且可水平扩容。

多Master架构图



7.1 部署Master2 Node

这里的话，机器资源不够，就部署一台学习下，生产环境最少三台。

现在需要再增加一台新服务器，作为Master2 Node，IP是192.168.16.110

Master2 与已部署的Master1所有操作一致。所以我们只需将Master1所有K8s文件拷贝过来，再修改下服务器IP和主机名启动即可。

（1）安装Docker

```
scp root@192.168.16.120:/usr/bin/docker* /usr/bin/
scp root@192.168.16.120:/usr/bin/runc  /usr/bin/
scp root@192.168.16.120:/usr/bin/containerd*  /usr/bin/
scp root@192.168.16.120:/usr/lib/systemd/system/docker.service /usr/lib/systemd/system
scp -r root@192.168.16.120:/etc/docker /etc

systemctl daemon-reload
systemctl start docker
systemctl enable docker
systemctl status docker
```

（2）创建etcd证书目录

```
mkdir -p /opt/etcd/ssl
```

（3）拷贝文件

拷贝Master1上所有K8s文件和etcd证书到Master2：

```
scp root@192.168.16.120:/opt/kubernetes /opt
scp -r root@192.168.16.120:/opt/etcd/ssl /opt/etcd
scp -r root@192.168.16.120:/usr/lib/systemd/system/kube* /usr/lib/systemd/system
scp -r root@192.168.16.120:/usr/bin/kubectl /usr/bin/
scp -r root@192.168.16.120:/root/.kube /root
```

（4）删除证书文件

删除kubelet证书和kubeconfig文件，这个只作为master节点，不需要kubelet，启动kubelet的话会申请加入集群

```
rm -f /opt/kubernetes/cfg/kubelet.kubeconfig
rm -f /opt/kubernetes/ssl/kubelet*
```

（5）修改配置文件IP和主机名

修改apiserver、kubelet和kube-proxy配置文件为本地IP：

```
vi /opt/kubernetes/cfg/kube-apiserver.conf
--bind-address=192.168.16.110 \
--advertise-address=192.168.16.110 \

vi /opt/kubernetes/cfg/kube-controller-manager.kubeconfig
    server: https://192.168.16.110:6443

vi /opt/kubernetes/cfg/kube-scheduler.kubeconfig
    server: https://192.168.16.110:6443

vi ~/.kube/config
    server: https://192.168.16.110:6443
```

```
vi /opt/kubernetes/cfg/kubelet.conf
--hostname-override=master2 \

vi /opt/kubernetes/cfg/kube-proxy-config.yml
hostnameOverride: master2
```

（6）启动

```
systemctl daemon-reload
systemctl start kube-apiserver kube-controller-manager kube-scheduler
systemctl enable kube-apiserver kube-controller-manager kube-scheduler
systemctl status kube-apiserver kube-controller-manager kube-scheduler
```

启动没问题，查看node或pod，这里报错了

```
[root@master2 opt]# kubectl get node
Unable to connect to the server: x509: certificate is valid for 10.0.0.1, 10.254.0.1,
127.0.0.1, 192.168.16.120, 192.168.16.130, 192.168.16.140, 192.168.16.150,
192.168.16.160, 192.168.16.170, 192.168.16.180, 192.168.16.16, not 192.168.16.110
```

这里是说，当前服务器的IP地址没有通过证书的认证，因为我们前面没有写192.168.16.110这个ip，所以说前面的证书文件一定要多预留几个IP，这里改为192.168.16.150。

上面的配置文件都改下

```
[root@master2 ~]# kubectl get node
NAME      STATUS   ROLES    AGE   VERSION
master1   Ready    <none>   38h   v1.19.0
worker1   Ready    <none>   25h   v1.19.0
worker2   Ready    <none>   24h   v1.19.0
[root@master2 ~]# kubectl get po -A
NAMESPACE              NAME                                         READY   STATUS    RESTARTS   AGE
default                dns-test                                     1/1     Running   1          11h
kube-system            coredns-858f6dbf57-l6t26                     1/1     Running   1          11h
kube-system            kube-flannel-ds-amd64-244rj                  1/1     Running   8          24h
kube-system            kube-flannel-ds-amd64-fd6tj                  1/1     Running   2          25h
kube-system            kube-flannel-ds-amd64-x4577                  1/1     Running   3          38h
kubernetes-dashboard   dashboard-metrics-scraper-7b9b99d599-8xjk8   1/1     Running   1          11h
kubernetes-dashboard   kubernetes-dashboard-6d4799d74-vjjcg         1/1     Running   1          11h
[root@master2 ~]#
```

这个是最后的效果，master2本身没有加入集群，如果想加入就把kubelet和kube-proxy配置文件的ip和主机名改一下启动就行，然后批准证书。 算了 弄一下吧。。。

（7）启动kubelet和kube-proyx

```
systemctl daemon-reload
systemctl start kubelet kube-proxy
systemctl enable kubelet kube-proxy
systemctl status kubelet kube-proxy
```

（8）批准kubelet证书申请

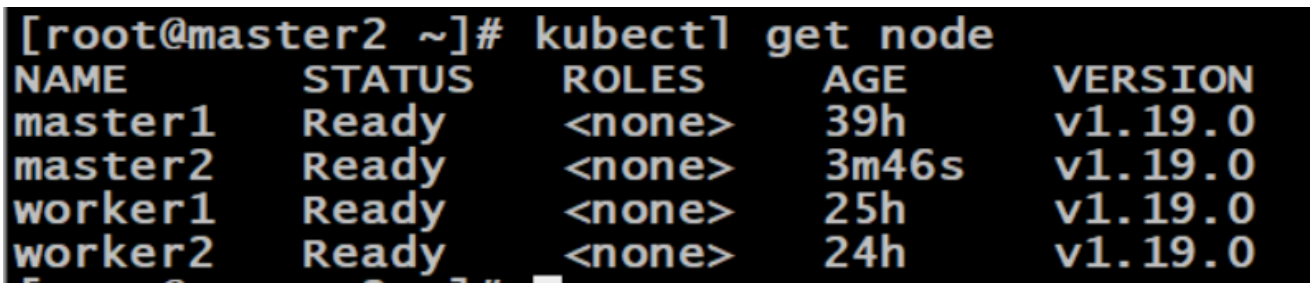
```
# 查看证书请求
kubectl get csr
# 授权请求
kubectl certificate approve node-csr-LeMmDEvdgUmZ4lhQJ-WZCODcz3vVy1YCaq_GjPBYmLw
# 查看Node
kubectl get node
```

NotReady，cni没安装

（9）部署cni

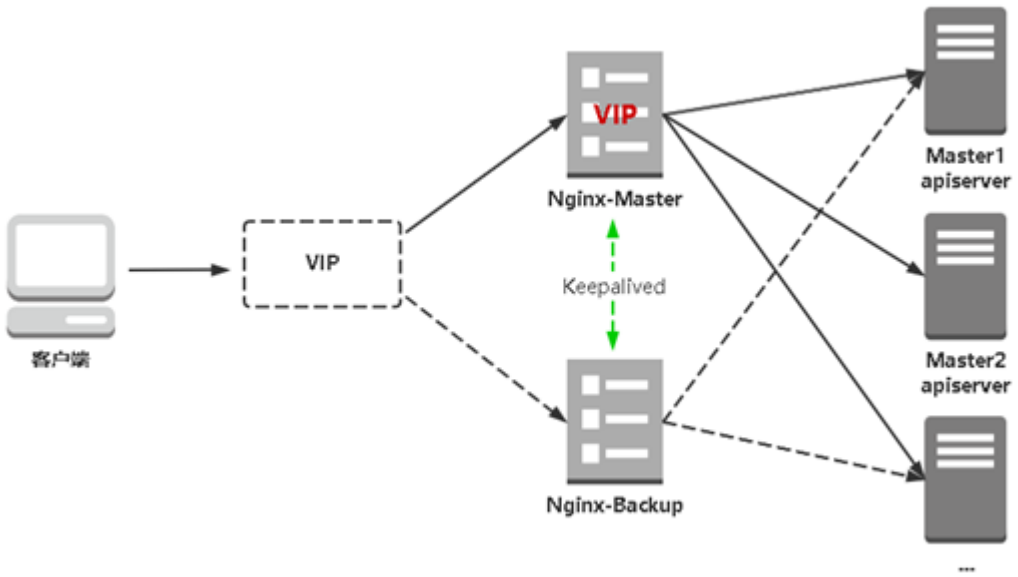
```
创建cni所需目录
mkdir /opt/cni/bin -p   #可执行文件
mkdir /etc/cni/net.d -p #cni配置信息

scp -r root@192.168.16.120:/opt/cni/bin/ /opt/cni/
```



7.2 部署Nginx+Keepalived高可用负载均衡器

kube-apiserver高可用架构图：



- Nginx是一个主流Web服务和反向代理服务器，这里用四层实现对apiserver实现负载均衡。

  4层用的是NAT技术，请求进来的时候，nginx修改数据包里面的目标和源IP和端口，然后把数据包发向目标服务器，服务器处理完成后，nginx再做一次修改，返回给请求的客户端。

  7层代理：需要读取并解析http请求内容，然后根据具体内容(url,参数，cookie,请求头)然后转发到相应的服务器，转发的过程是：建立和目标机器的连接，然后转发请求，收到响应数据在转发给请求客户端。

理论上4层要比7层快，因为7层代理需要解析数据包的具体内容，需要消耗额外的cpu。这里我们的nginx只是负责转发请求，并不需要处理数据包的具体信息，所以采用四层。

注1：为了节省机器，这里与K8s Master节点机器复用。也可以独立于k8s集群之外部署，只要nginx与apiserver能通信就行。

注2：如果你是在公有云上，一般都不支持keepalived，那么你可以直接用它们的负载均衡器产品，直接负载均衡多台Master kube-apiserver，架构与上面一样。

- Keepalived是一个主流高可用软件，基于VIP绑定实现服务器双机热备，在上述拓扑中，Keepalived主要根据Nginx运行状态判断是否需要故障转移（漂移VIP），例如当Nginx主节点挂掉，VIP会自动绑定在Nginx备节点，从而保证VIP一直可用，实现Nginx高可用。

在两台Master节点操作：

（1）安装软件包（主/备）

```
yum install epel-release -y
yum install nginx keepalived nginx-mod-stream -y
```

（2）Nginx配置文件（主/备一样）

```
cat > /etc/nginx/nginx.conf << "EOF"
user nginx;
worker_processes auto;
error_log /var/log/nginx/error.log;
pid /run/nginx.pid;

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

events {
    worker_connections 1024;
}

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

    log_format  main  '$remote_addr $upstream_addr - [$time_local] $status
$upstream_bytes_sent';

    access_log  /var/log/nginx/k8s-access.log  main;

    upstream k8s-apiserver {
       server 192.168.16.120:6443;   # Master1 APISERVER IP:PORT
       server 192.168.16.150:6443;   # Master2 APISERVER IP:PORT
    }

    server {
       listen 16443; # 由于nginx与master节点复用，这个监听端口不能是6443，否则会冲突，一般的话独立
部署就是6443就可以
       proxy_pass k8s-apiserver;
    }
}

http {
    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;

    sendfile            on;
    tcp_nopush          on;
    tcp_nodelay         on;
    keepalive_timeout   65;
    types_hash_max_size 2048;

    include             /etc/nginx/mime.types;
    default_type        application/octet-stream;

    server {
        listen       80 default_server;
        server_name  _;

        location / {
```

```
        }
    }
}
EOF
```

（3）keepalived配置文件（Nginx Master）

```
cat > /etc/keepalived/keepalived.conf << EOF
global_defs {
   notification_email {
     acassen@firewall.loc
     failover@firewall.loc
     sysadmin@firewall.loc
   }
   notification_email_from Alexandre.Cassen@firewall.loc
   smtp_server 127.0.0.1
   smtp_connect_timeout 30
   router_id NGINX_MASTER
}

vrrp_script check_nginx {
    script "/etc/keepalived/check_nginx.sh"
#    interval 2  #检测脚本执行的间隔
}

vrrp_instance VI_1 {
    state MASTER
    interface ens33  # 修改为实际网卡名
    virtual_router_id 51 # VRRP 路由 ID实例，每个实例是唯一的
    priority 100    # 优先级，备服务器设置 90
    advert_int 1    # 指定VRRP 心跳包通告间隔时间，默认1秒
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    # 虚拟IP
    virtual_ipaddress {
        192.168.16.66/24                # 多个虚拟ip换行即可
    }
    track_script {
        check_nginx             #（调用检测脚本）
    }
}
EOF
```

- vrrp_script：指定检查nginx工作状态脚本（根据nginx状态判断是否故障转移）
- virtual_ipaddress：虚拟IP（VIP）

准备上述配置文件中检查nginx运行状态的脚本：

```
cat > /etc/keepalived/check_nginx.sh  << "EOF"
#!/bin/bash
count=$(ss -antp |grep 16443 |egrep -cv "grep|$$")

if [ "$count" -eq 0 ];then
    exit 1
else
    exit 0
fi
EOF
chmod +x /etc/keepalived/check_nginx.sh
```

(4) keepalived配置文件（Nginx Backup）

```
cat > /etc/keepalived/keepalived.conf << EOF
global_defs {
   notification_email {
     acassen@firewall.loc
     failover@firewall.loc
     sysadmin@firewall.loc
   }
   notification_email_from Alexandre.Cassen@firewall.loc
   smtp_server 127.0.0.1
   smtp_connect_timeout 30
   router_id NGINX_BACKUP
}

vrrp_script check_nginx {
    script "/etc/keepalived/check_nginx.sh"
#    interval 2  #检测脚本执行的间隔
}

vrrp_instance VI_1 {
    state BACKUP
    interface ens33
    virtual_router_id 51 # VRRP 路由 ID实例，每个实例是唯一的
    priority 90
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        192.168.16.66/24
    }
    track_script {
        check_nginx             #（调用检测脚本）
    }
}
EOF
```

准备上述配置文件中检查nginx运行状态的脚本：

```
cat > /etc/keepalived/check_nginx.sh  << "EOF"
#!/bin/bash
count=$(ss -antp |grep 16443 |egrep -cv "grep|$$")

if [ "$count" -eq 0 ];then
    exit 1
else
    exit 0
fi
EOF
chmod +x /etc/keepalived/check_nginx.sh
```

注：keepalived根据脚本返回状态码（0为工作正常，非0不正常）判断是否故障转移。

（5）启动

```
systemctl daemon-reload
systemctl restart nginx keepalived
systemctl enable nginx keepalived
systemctl status nginx keepalived
```

（6）查看keepalived工作状态

```
ip a #查看vip所在节点
```

```
[^_^][root@master1 ~]# ip a|grep -A3 ens33
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:0c:29:97:7d:75 brd ff:ff:ff:ff:ff:ff
    inet 192.168.16.120/24 brd 192.168.16.255 scope global noprefixroute ens33
       valid_lft forever preferred_lft forever
    inet 192.168.16.66/24 scope global secondary ens33
       valid_lft forever preferred_lft forever
    inet6 fe80::1bec:f8ad:4995:c5a2/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

可以看到，在master1的ens33网卡绑定了 虚拟IP，说明工作正常。

（7）Nginx+Keepalived高可用测试

关闭主节点Nginx，测试VIP是否漂移到备节点服务器。

在Nginx Backup，ip addr命令查看已成功绑定VIP。

关闭主节点Keepalived，测试VIP是否漂移到备节点服务器。

在Nginx Backup，ip addr命令查看已成功绑定VIP。

（8）访问负载均衡器测试

找K8s集群中任意一个节点，使用curl查看K8s版本测试，使用VIP访问：

```
curl -k https://192.168.16.66:16443/version
{
  "major": "1",
  "minor": "19",
  "gitVersion": "v1.19.0",
  "gitCommit": "e19964183377d0ec2052d1f1fa930c4d7575bd50",
  "gitTreeState": "clean",
  "buildDate": "2020-08-26T14:23:04Z",
  "goVersion": "go1.15",
  "compiler": "gc",
  "platform": "linux/amd64"
}
```

可以正确获取到K8s版本信息，说明负载均衡器搭建正常。该请求数据流程：curl -> vip(nginx) -> apiserver

通过查看Nginx日志也可以看到转发apiserver IP：

```
for i in {1..20}; do curl -k https://192.168.16.66:16443/version;done   #请求20次

tail /var/log/nginx/k8s-access.log -f          #查看日志
```

```
[^_^][root@master1 /etc/nginx]# tail /var/log/nginx/k8s-access.log -f
192.168.16.150 192.168.16.120:6443 - [21/Mar/2022:14:26:46 +0800] 200 423
192.168.16.150 192.168.16.120:6443 - [21/Mar/2022:14:26:46 +0800] 200 423
192.168.16.150 192.168.16.150:6443 - [21/Mar/2022:14:26:47 +0800] 200 423
192.168.16.150 192.168.16.120:6443 - [21/Mar/2022:14:26:47 +0800] 200 423
192.168.16.150 192.168.16.150:6443 - [21/Mar/2022:14:26:47 +0800] 200 423
192.168.16.150 192.168.16.120:6443 - [21/Mar/2022:14:26:47 +0800] 200 423
192.168.16.150 192.168.16.150:6443 - [21/Mar/2022:14:26:47 +0800] 200 423
192.168.16.150 192.168.16.120:6443 - [21/Mar/2022:14:26:47 +0800] 200 423
192.168.16.150 192.168.16.150:6443 - [21/Mar/2022:14:26:47 +0800] 200 423
```

到此还没结束，还有下面最关键的一步。

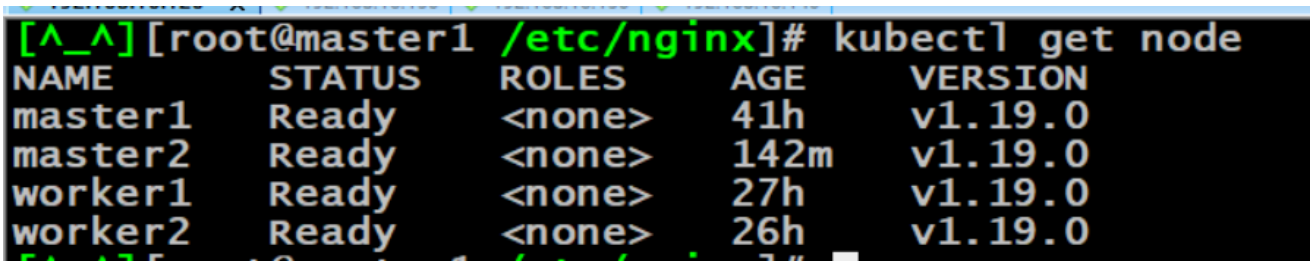
7.3 修改所有Worker Node连接LB VIP

试想下，虽然我们增加了Master2 Node和负载均衡器，但是我们是从单Master架构扩容的，也就是说目前所有的Worker Node组件连接都还是Master1 Node，如果不改为连接VIP走LB，那么Master还是单点故障。

因此接下来就是要改所有Worker Node（kubectl get node命令查看到的节点）组件配置文件，由原来192.168.16.120修改为192.168.16.66（VIP）。

在所有Worker Node执行：

```
```


[^_^][root@master1 /etc/nginx]# kubectl get node
NAME STATUS ROLES AGE VERSION
master1 Ready <none> 41h v1.19.0
master2 Ready <none> 142m v1.19.0
worker1 Ready <none> 27h v1.19.0
worker2 Ready <none> 26h v1.19.0