Kubernetes(二进制)高可用部署教程

`作者：shouzhi@duan`

一、部署特点

- 二进制部署需要大家手动部署，下载集群中所有相关部署组件，从而让大家深入了解各组件的角色功能。
- 生产级别的高可用部署方式，避免其他工具部署方式（kube-admin、kubespray）带来相关问题。
    - 部署过程中个组件版本兼容问题。
    - 部署过程中组件下载超时或者失败的问题。
    - 部署过程中工具脚本(ansible)自动化错误问题，难以介入解决。
    - 证书过期问题等。
- 高可用不依赖haproxy、keepalived。采用本地代理方式，简单优雅。

二、适合群体

- 对kubernetes系统有一定的基础认知。
- 想深入学习kubernetes。
- 对kubernetes二进制部署有强烈的兴趣。
- 正在接触或者学习部署生产级别kubernetes高可用集群。

三、部署教程

1-基础环境准备

1.1服务器说明

1.1.1 节点要求

- 节点数>=3台
- CPU>=2
- 内存>=2G
- 关闭安全组，允许节点之间任意端口访问，以及ipip隧道协议通讯

1.1.2 服务器分配说明

系统类型	IP	角色	CPU	内存/G	HostName
Ubuntu	192.168.10.121	master	8	15	node-1
Ubuntu	192.168.10.122	master|worker	8	15	node-2
Ubuntu	192.168.10.123	worker	8	15	node-3

1.2系统设置（所有集群机器都要操作）

1.2.1域名映射

```
vi /etc/hosts
#kubernetes
192.168.10.121 node-1
192.168.10.122 node-2
192.168.10.123 node-3
```

1.2.2下载相关软件包

```
socat conntrack ipvsadm ipset jq sysstat curl iptables libseccomp yum-utils
```

1.2.3关闭防火墙、selinux、swap，重置iptables

```
# 关闭selinux
$ setenforce 0
$ sed -i '/SELINUX/s/enforcing/disabled/' /etc/selinux/config
# 关闭防火墙
$ systemctl stop firewalld && systemctl disable firewalld

# 设置iptables规则
$ iptables -F && iptables -X && iptables -F -t nat && iptables -X -t nat &&
iptables -P FORWARD ACCEPT
# 关闭swap
$ swapoff -a && free -h

# 关闭dnsmasq(否则可能导致容器无法解析域名)
$ service dnsmasq stop && systemctl disable dnsmasq
```

1.2.3kubernetes参数配置

```
# 制作配置文件
$ cat > /etc/sysctl.d/kubernetes.conf <<EOF
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
net.ipv4.ip_nonlocal_bind = 1
net.ipv4.ip_forward = 1
vm.swappiness = 0
vm.overcommit_memory = 1
EOF
# 生效文件
$ sysctl -p /etc/sysctl.d/kubernetes.conf
```

1.2.4免密登录配置

```
# 看看是否已经存在rsa公钥
$ cat ~/.ssh/id_rsa.pub

# 如果不存在就创建一个新的
$ ssh-keygen -t rsa

# 把id_rsa.pub文件内容copy到其他机器的授权文件中
$ cat ~/.ssh/id_rsa.pub
```

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQDGVY93IOuYzT7C1Z6ZsyqNjDYMzF9QsiRE2rYVjtN+yQ18zVEm
wPGVvHrJgqx7TDd2ir2cfMi9whpADA65L/LHDub2PK1OSB5OMdS2gaMFoSkoCtzlz+nkkEHOYGIRBbJU
J944Ha8MWSwWEHd0K/7F+F1Y2DpPMfRT4Ohaond2oKYnDA0r8LnOOJSMdMprGBnVtRdSR+8fxgJadGhb
JReLjyJRdrMzW1cvJUXfp2DeR68jS7fxOd2vEV8+8S679aJvIwc+3X51WNYaKHx0I4fRMMvFusIFPZxD
6G9h6Lm+mzVpFIgFfopfcyQ3QRO4sqSKexbRoCHm8YXNlq3RuKbB root@fabric1

# 在其他节点执行下面命令（包括worker节点）
$ echo "<file_content>" >> ~/.ssh/authorized_keys


echo "ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQDGVY93IOuYzT7C1Z6ZsyqNjDYMzF9QsiRE2rYVjtN+yQ18zVEm
wPGVvHrJgqx7TDd2ir2cfMi9whpADA65L/LHDub2PK1OSB5OMdS2gaMFoSkoCtzlz+nkkEHOYGIRBbJU
J944Ha8MWSwWEHd0K/7F+F1Y2DpPMfRT4Ohaond2oKYnDA0r8LnOOJSMdMprGBnVtRdSR+8fxgJadGhb
JReLjyJRdrMzW1cvJUXfp2DeR68jS7fxOd2vEV8+8S679aJvIwc+3X51WNYaKHx0I4fRMMvFusIFPZxD
6G9h6Lm+mzVpFIgFfopfcyQ3QRO4sqSKexbRoCHm8YXNlq3RuKbB root@fabric1" >>
~/.ssh/authorized_keys
```

1.3准备kubernetes软件包

> 6个软件包

```
master节点组件：
 1、kube-apiserver
 2、kube-controller-manager
 3、kube-scheduler
 4、kubectl

worker节点组件：
 1、kubelet
 2、kube-proxy

etcd节点组件：
 1、etcd
 2、etcdctl
```

> 下载教程（需要梯子）

```
# 设定版本号
$ export VERSION=v1.20.2

# 下载master节点组件
$ wget https://storage.googleapis.com/kubernetes-
release/release/${VERSION}/bin/linux/amd64/kube-apiserver
$ wget https://storage.googleapis.com/kubernetes-
release/release/${VERSION}/bin/linux/amd64/kube-controller-manager
$ wget https://storage.googleapis.com/kubernetes-
release/release/${VERSION}/bin/linux/amd64/kube-scheduler
$ wget https://storage.googleapis.com/kubernetes-
release/release/${VERSION}/bin/linux/amd64/kubectl
解压：
 1、kube-apiserver
 2、kube-controller-manager
 3、kube-scheduler
 4、kubectl
```

```
# 下载worker节点组件。
$ wget https://storage.googleapis.com/kubernetes-release/release/${VERSION}/bin/linux/amd64/kube-proxy
$ wget https://storage.googleapis.com/kubernetes-release/release/${VERSION}/bin/linux/amd64/kubelet
解压：
 1、kubelet
 2、kube-proxy

# 下载etcd组件。解压后的(etcd、etcdctl)需要分发到所有etcd集群的服务器。
$ wget https://github.com/etcd-io/etcd/releases/download/v3.4.10/etcd-v3.4.10-linux-amd64.tar.gz
$ tar -xvf etcd-v3.4.10-linux-amd64.tar.gz
$ mv etcd-v3.4.10-linux-amd64/etcd* .
$ rm -fr etcd-v3.4.10-linux-amd64*
解压：
 1、etcd
 2、etcdctl

# 统一修改文件权限为可执行
$ chmod +x kube*
```

> 如果没有梯子下载，可以点击下载，这里面包含ETCD以及以上6个软件包
>
> 提取码：`kwvv`

1.4软件包分发

完成下载后，将这个 6个软件包分发到各个角色主机相关目录。不同的角色主机需要的相关软件包会有不同。

```
# 把master相关组件分发到master节点
# kube-apiserver kube-controller-manager kube-scheduler kubectl分发到所有的master节点
$ MASTERS=(node-1 node-2)
for instance in ${MASTERS[@]}; do
  scp kube-apiserver kube-controller-manager kube-scheduler kubectl root@${instance}:/usr/local/bin/
done

# 把worker先关组件分发到worker节点
# kubelet kube-proxy分发到所有的worker节点
$ WORKERS=(node-2 node-3)
for instance in ${WORKERS[@]}; do
  scp kubelet kube-proxy root@${instance}:/usr/local/bin/
done

# 把etcd组件分发到etcd节点
# etcd etcdctl分发到所有的etcd节点
$ ETCDS=(node-1 node-2 node-3)
for instance in ${ETCDS[@]}; do
  scp etcd etcdctl root@${instance}:/usr/local/bin/
done
```

2-证书制作

证书制作可以选择任意一台集群内的机器或者集群外的机器，只是用作证书签发的平台，所有证书签发好之后将各自的证书分发到不同的集群角色中。证书签发之前创建一个证书统一管理的目录，以便后续统一分发管理。

2.1 安装cfssl

cfssl是一个CA工具，下面将会用他来安装证书和秘钥文件，安装过程比较简单，安装方法如下

```
# 1、下载
$ 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
# 2、修改为可执行权限
$ chmod +x /usr/local/bin/cfssl /usr/local/bin/cfssljson
# 3、验证
$ cfssl version
```

2.2 根证书

根证书是集群所有节点共享的，只需要创建一个 CA 证书，后续创建的所有证书都由它签名。
在任意节点（可以免密登录到其他节点）创建一个单独的证书目录。

2.2.1 根证书配置文件

```
$ cat > ca-config.json <<EOF
{
  "signing": {
    "default": {
      "expiry": "876000h"
    },
    "profiles": {
      "kubernetes": {
        "usages": ["signing", "key encipherment", "server auth", "client auth"],
        "expiry": "876000h"
      }
    }
  }
}
EOF

$ cat > ca-csr.json <<EOF
{
  "CN": "Kubernetes",
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "US",
      "L": "Portland",
      "O": "Kubernetes",
      "OU": "CA",
      "ST": "Oregon"
    }
  ]
}
EOF
```

2.2.2 生成证书和私钥

```
# 生成证书和私钥
$ cfssl gencert -initca ca-csr.json | cfssljson -bare ca
# 生成完成后会有以下文件（我们最终想要的就是ca-key.pem和ca.pem，一个秘钥，一个证书）
$ ls
ca-config.json  ca.csr  ca-csr.json  ca-key.pem  ca.pem
```

2.3 admin客户端证书

2.3.1 admin客户端证书配置文件

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

2.3.2 生成admin客户端证书和私钥

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

2.4 kubelet客户端证书

`Kubernetes使用一种称为Node Authorizer的专用授权模式来授权Kubelets发出的API请求。 Kubelet使用将其标识为system:nodes组中的凭据，其用户名为system：node:nodeName，接下里就给每个工作节点生成证书。`

生成kubelet客户端证书和私钥

```
# 第一步：设置环境变量
$ WORKERS=(node-2 node-3) #所有worker节点的主机列表
$ WORKER_IPS=(10.155.19.64 10.155.19.147) #所有worker节点IP列表

# 第二步：生成所有worker节点的证书配置
$ for ((i=0;i<${#WORKERS[@]};i++)); do
```

```
cat > ${WORKERS[$i]}-csr.json <<EOF
{
  "CN": "system:node:${WORKERS[$i]}",
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "CN",
      "L": "Beijing",
      "O": "system:nodes",
      "OU": "seven",
      "ST": "Beijing"
    }
  ]
}
EOF

#第三步：生成证书
cfssl gencert \
  -ca=ca.pem \
  -ca-key=ca-key.pem \
  -config=ca-config.json \
  -hostname=${WORKERS[$i]},${WORKER_IPS[$i]} \
  -profile=kubernetes \
  ${WORKERS[$i]}-csr.json | cfssljson -bare ${WORKERS[$i]}
done
```

2.5 kube-controller-manager客户端证书

kube-controller-manager客户端证书配置文件

```
$ cat > kube-controller-manager-csr.json <<EOF
{
    "CN": "system:kube-controller-manager",
    "key": {
        "algo": "rsa",
        "size": 2048
    },
    "names": [
      {
        "C": "CN",
        "ST": "BeiJing",
        "L": "BeiJing",
        "O": "system:kube-controller-manager",
        "OU": "seven"
      }
    ]
}
EOF
```

生成kube-controller-manager客户端证书

```
$ 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
```

2.6 kube-proxy客户端证书

kube-proxy客户端证书配置文件

```
$ cat > kube-proxy-csr.json <<EOF
{
  "CN": "system:kube-proxy",
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "CN",
      "ST": "BeiJing",
      "L": "BeiJing",
      "O": "k8s",
      "OU": "seven"
    }
  ]
}
EOF
```

生成kube-proxy客户端证书

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

2.7 kube-scheduler客户端证书

kube-scheduler客户端证书配置文件

```
$ cat > kube-scheduler-csr.json <<EOF
{
    "CN": "system:kube-scheduler",
    "key": {
        "algo": "rsa",
        "size": 2048
    },
    "names": [
      {
        "C": "CN",
        "ST": "BeiJing",
        "L": "BeiJing",
        "O": "system:kube-scheduler",
        "OU": "seven"
      }
    ]
}
EOF
```

生成kube-scheduler客户端证书

```
$ cfssl gencert \
  -ca=ca.pem \
  -ca-key=ca-key.pem \
  -config=ca-config.json \
  -profile=kubernetes \
  kube-scheduler-csr.json | cfssljson -bare kube-scheduler
```

2.8 kube-apiserver服务端证书

kube-apiserver服务端证书配置文件

```
$ cat > kubernetes-csr.json <<EOF
{
  "CN": "kubernetes",
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "CN",
      "ST": "BeiJing",
      "L": "BeiJing",
      "O": "k8s",
      "OU": "seven"
    }
  ]
}
EOF
```

生成kube-apiserver服务端证书

服务端证书与客户端略有不同，客户端需要通过一个名字或者一个ip去访问服务端，所以证书必须要包含客户端所访问的名字或ip，用以客户端验证。

```
# apiserver的service ip地址（一般是svc网段的第一个ip）
$ KUBERNETES_SVC_IP=10.233.0.1
# 所有的master内网ip，逗号分隔（云环境可以加上master公网ip以便支持公网ip访问）
$ MASTER_IPS=192.168.10.121 192.168.10.122 192.168.10.123
# 生成证书
$ cfssl gencert \
  -ca=ca.pem \
  -ca-key=ca-key.pem \
  -config=ca-config.json \
  -hostname=${KUBERNETES_SVC_IP},${MASTER_IPS},127.0.0.1,kubernetes,kubernetes.default,kubernetes.default.svc,kubernetes.default.svc.cluster,kubernetes.svc.cluster.local \
  -profile=kubernetes \
  kubernetes-csr.json | cfssljson -bare kubernetes
```

2.9 Service Account证书

配置文件

```
$ cat > service-account-csr.json <<EOF
{
  "CN": "service-accounts",
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "CN",
      "ST": "BeiJing",
      "L": "BeiJing",
      "O": "k8s",
      "OU": "seven"
    }
  ]
}
EOF
```

生成证书

```
$ cfssl gencert \
  -ca=ca.pem \
  -ca-key=ca-key.pem \
  -config=ca-config.json \
  -profile=kubernetes \
  service-account-csr.json | cfssljson -bare service-account
```

2.10 proxy-client 证书

配置文件

```
$ cat > proxy-client-csr.json <<EOF
{
  "CN": "aggregator",
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "CN",
      "ST": "BeiJing",
      "L": "BeiJing",
      "O": "k8s",
      "OU": "seven"
    }
  ]
}
EOF
```

生成证书

```
$ cfssl gencert \
  -ca=ca.pem \
  -ca-key=ca-key.pem \
  -config=ca-config.json \
  -profile=kubernetes \
  proxy-client-csr.json | cfssljson -bare proxy-client
```

2.11 分发客户端、服务端证书

2.11.1 分发worker节点需要的证书和私钥

```
for instance in ${WORKERS[@]}; do
  scp ca.pem ${instance}-key.pem ${instance}.pem root@${instance}:~/
done
```

2.11.2 分发master节点需要的证书和私钥

> 注意：由于下面分发的证书即包含了etcd的证书也包含了k8s主节点的证书。 所以 MASTER_IPS 中必须包含所有 `master` 节点以及 `etcd` 节点。如果没有包含所有etcd节点的证书，需要重新定义，逗号分隔

```
OIFS=$IFS
IFS=','
for instance in ${MASTER_IPS}; do
  scp ca.pem ca-key.pem kubernetes-key.pem kubernetes.pem \
    service-account-key.pem service-account.pem proxy-client.pem proxy-client-key.pem root@${instance}:~/
done
IFS=$OIFS
```

3-kubernetes各组件的认证配置

> kubernetes的认证配置文件，也叫kubeconfigs，用于让kubernetes的客户端定位kube-apiserver并通过apiserver的安全认证。
>
> 接下来我们一起来生成各个组件的kubeconfigs，包括controller-manager，kubelet，kube-proxy，scheduler，以及admin用户。
>
> 以下命令需要与上一节“生成证书”在同一个目录下执行

3.1 kubelet

```
# 指定你的worker列表（hostname），空格分隔
$ WORKERS="node-2 node-3"
$ for instance in ${WORKERS}; do
  kubectl config set-cluster kubernetes \
    --certificate-authority=ca.pem \
    --embed-certs=true \
    --server=https://127.0.0.1:6443 \
    --kubeconfig=${instance}.kubeconfig

  kubectl config set-credentials system:node:${instance} \
    --client-certificate=${instance}.pem \
    --client-key=${instance}-key.pem \
    --embed-certs=true \
    --kubeconfig=${instance}.kubeconfig

  kubectl config set-context default \
    --cluster=kubernetes \
    --user=system:node:${instance} \
    --kubeconfig=${instance}.kubeconfig

  kubectl config use-context default --kubeconfig=${instance}.kubeconfig
done
```

3.2 kube-proxy

```
kubectl config set-cluster kubernetes \
    --certificate-authority=ca.pem \
    --embed-certs=true \
    --server=https://127.0.0.1:6443 \
    --kubeconfig=kube-proxy.kubeconfig

kubectl config set-credentials system:kube-proxy \
   --client-certificate=kube-proxy.pem \
   --client-key=kube-proxy-key.pem \
   --embed-certs=true \
   --kubeconfig=kube-proxy.kubeconfig

kubectl config set-context default \
   --cluster=kubernetes \
   --user=system:kube-proxy \
   --kubeconfig=kube-proxy.kubeconfig

kubectl config use-context default --kubeconfig=kube-proxy.kubeconfig
```

3.3 kube-controller-manager

```
kubectl config set-cluster kubernetes \
  --certificate-authority=ca.pem \
  --embed-certs=true \
  --server=https://127.0.0.1:6443 \
  --kubeconfig=kube-controller-manager.kubeconfig

kubectl config set-credentials system:kube-controller-manager \
  --client-certificate=kube-controller-manager.pem \
  --client-key=kube-controller-manager-key.pem \
  --embed-certs=true \
  --kubeconfig=kube-controller-manager.kubeconfig

kubectl config set-context default \
  --cluster=kubernetes \
  --user=system:kube-controller-manager \
  --kubeconfig=kube-controller-manager.kubeconfig

kubectl config use-context default --kubeconfig=kube-controller-manager.kubeconfig
```

3.4 kube-scheduler

```
kubectl config set-cluster kubernetes \
  --certificate-authority=ca.pem \
  --embed-certs=true \
  --server=https://127.0.0.1:6443 \
  --kubeconfig=kube-scheduler.kubeconfig

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

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

kubectl config use-context default --kubeconfig=kube-scheduler.kubeconfig
```

3.5 admin用户配置

> 为admin用户生成kubeconfig配置

```
kubectl config set-cluster kubernetes \
  --certificate-authority=ca.pem \
  --embed-certs=true \
  --server=https://127.0.0.1:6443 \
  --kubeconfig=admin.kubeconfig

kubectl config set-credentials admin \
  --client-certificate=admin.pem \
  --client-key=admin-key.pem \
  --embed-certs=true \
  --kubeconfig=admin.kubeconfig

kubectl config set-context default \
  --cluster=kubernetes \
  --user=admin \
  --kubeconfig=admin.kubeconfig

kubectl config use-context default --kubeconfig=admin.kubeconfig
```

3.6 分发配置文件

3.6.1 把kubelet和kube-proxy需要的kubeconfig配置分发到每个worker节点

```
$ WORKERS="node-2 node-3"
$ for instance in ${WORKERS}; do
    scp ${instance}.kubeconfig kube-proxy.kubeconfig ${instance}:~/
done
```

3.6.2 把kube-controller-manager和kube-scheduler需要的kubeconfig配置分发到master节点

```
$ MASTERS="node-1 node-2"
$ for instance in ${MASTERS}; do
    scp admin.kubeconfig kube-controller-manager.kubeconfig kube-scheduler.kubeconfig ${instance}:~/
done
```

4-部署ETCD集群

> Kubernetes组件是无状态的，并在etcd中存储集群状态。 在本小节中，我们将部署三个节点的etcd群集，并对其进行配置以实现高可用性和安全的远程访问。
>
> 注意：以下操作需要在所有的ETCD服务器上操作，注意各自的hostname以及IP。

4.1 配置ETCD

> copy必要的证书文件

```
$ mkdir -p /etc/etcd /var/lib/etcd
$ chmod 700 /var/lib/etcd
$ cp ca.pem kubernetes-key.pem kubernetes.pem /etc/etcd/
```

> 配置etcd.service文件

```
#各个etcd主机名
$ ETCD_NAME=$(hostname -s)
#各个etcd主机IP
$ ETCD_IP=10.155.19.223
# etcd所有节点的ip地址
$ ETCD_NAMES=(node-1 node-2 node-3)
$ ETCD_IPS=(192.168.10.121 192.168.10.122 192.168.10.123)
$ cat <<EOF > /etc/systemd/system/etcd.service
[Unit]
Description=etcd
Documentation=https://github.com/coreos

[Service]
Type=notify
ExecStart=/usr/local/bin/etcd \\
  --name ${ETCD_NAME} \\
  --cert-file=/etc/etcd/kubernetes.pem \\
  --key-file=/etc/etcd/kubernetes-key.pem \\
  --peer-cert-file=/etc/etcd/kubernetes.pem \\
  --peer-key-file=/etc/etcd/kubernetes-key.pem \\
  --trusted-ca-file=/etc/etcd/ca.pem \\
  --peer-trusted-ca-file=/etc/etcd/ca.pem \\
  --peer-client-cert-auth \\
  --client-cert-auth \\
  --initial-advertise-peer-urls https://${ETCD_IP}:2380 \\
  --listen-peer-urls https://${ETCD_IP}:2380 \\
  --listen-client-urls https://${ETCD_IP}:2379,https://127.0.0.1:2379 \\
  --advertise-client-urls https://${ETCD_IP}:2379 \\
  --initial-cluster-token etcd-cluster-0 \\
  --initial-cluster ${ETCD_NAMES[0]}=https://${ETCD_IPS[0]}:2380,${ETCD_NAMES[1]}=https://${ETCD_IPS[1]}:2380,${ETCD_NAMES[2]}=https://${ETCD_IPS[2]}:2380 \\
  --initial-cluster-state new \\
  --data-dir=/var/lib/etcd
Restart=on-failure
RestartSec=5

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

4.2 启动ETCD集群

> 所有etcd节点都配置好etcd.service后，启动etcd集群

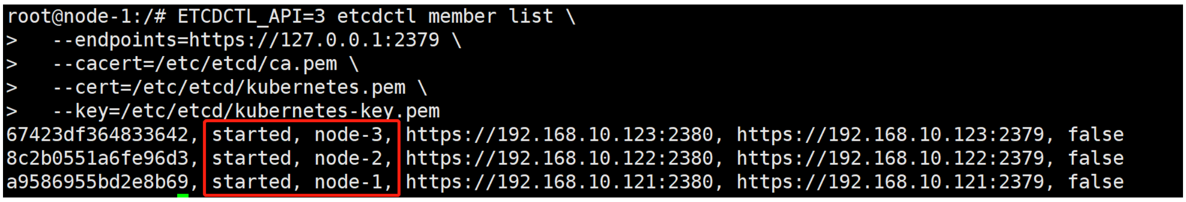
```
$ systemctl daemon-reload && systemctl enable etcd && systemctl restart etcd
```

4.3 验证ETCD集群

> 验证etcd集群状态

```
ETCDCTL_API=3 etcdctl member list \
  --endpoints=https://127.0.0.1:2379 \
  --cacert=/etc/etcd/ca.pem \
  --cert=/etc/etcd/kubernetes.pem \
  --key=/etc/etcd/kubernetes-key.pem
```

> 如下所示，表示启动成功



```
root@node-1:/# ETCDCTL_API=3 etcdctl member list \
>   --endpoints=https://127.0.0.1:2379 \
>   --cacert=/etc/etcd/ca.pem \
>   --cert=/etc/etcd/kubernetes.pem \
>   --key=/etc/etcd/kubernetes-key.pem
67423df364833642, started, node-3, https://192.168.10.123:2380, https://192.168.10.123:2379, false
8c2b0551a6fe96d3, started, node-2, https://192.168.10.122:2380, https://192.168.10.122:2379, false
a9586955bd2e8b69, started, node-1, https://192.168.10.121:2380, https://192.168.10.121:2379, false
```

5-部署kubernetes控制平面

> 这部分我们部署kubernetes的控制平面，每个组件有多个点保证高可用。实例中我们在两个节点上部署 API Server、Scheduler 和 Controller Manager。当然你也可以按照教程部署三个节点的高可用，操作都是一致的。

5.1 配置 API Server

> 下面的所有命令都是运行在每个master节点的，我们的实例中是 node-1 和 node-2.

```
# 创建kubernetes必要目录
$ mkdir -p /etc/kubernetes/ssl
# 准备证书文件
$ mv ca.pem ca-key.pem kubernetes-key.pem kubernetes.pem \
    service-account-key.pem service-account.pem \
    proxy-client.pem proxy-client-key.pem \
    /etc/kubernetes/ssl

# 配置kube-apiserver.service
# 本机内网ip
$ IP=10.155.19.223
# apiserver实例数
$ APISERVER_COUNT=2
# etcd节点
$ ETCD_ENDPOINTS=(10.155.19.223 10.155.19.64 10.155.19.147)
# 创建 apiserver service
$ cat <<EOF > /etc/systemd/system/kube-apiserver.service
[Unit]
Description=Kubernetes API Server
Documentation=https://github.com/kubernetes/kubernetes

[Service]
ExecStart=/usr/local/bin/kube-apiserver \\
  --advertise-address=${IP} \\
  --allow-privileged=true \\
  --apiserver-count=${APISERVER_COUNT} \\
  --audit-log-maxage=30 \\
  --audit-log-maxbackup=3 \\
  --audit-log-maxsize=100 \\
  --audit-log-path=/var/log/audit.log \\
  --authorization-mode=Node,RBAC \\
  --bind-address=0.0.0.0 \\
  --client-ca-file=/etc/kubernetes/ssl/ca.pem \\
  --enable-admission-plugins=NamespaceLifecycle,NodeRestriction,LimitRanger,ServiceAccount,DefaultStorageClass,ResourceQuota \\
  --etcd-cafile=/etc/kubernetes/ssl/ca.pem \\
  --etcd-certfile=/etc/kubernetes/ssl/kubernetes.pem \\
  --etcd-keyfile=/etc/kubernetes/ssl/kubernetes-key.pem \\
  --etcd-servers=https://${ETCD_ENDPOINTS[0]}:2379,https://${ETCD_ENDPOINTS[1]}:2379,https://${ETCD_ENDPOINTS[2]}:2379 \\
  --event-ttl=1h \\
  --kubelet-certificate-authority=/etc/kubernetes/ssl/ca.pem \\
  --kubelet-client-certificate=/etc/kubernetes/ssl/kubernetes.pem \\
  --kubelet-client-key=/etc/kubernetes/ssl/kubernetes-key.pem \\
  --service-account-issuer=api \\
  --service-account-key-file=/etc/kubernetes/ssl/service-account.pem \\
  --service-account-signing-key-file=/etc/kubernetes/ssl/service-account-key.pem \\
  --api-audiences=api,vault,factors \\
  --service-cluster-ip-range=10.233.0.0/16 \\
  --service-node-port-range=30000-32767 \\
  --proxy-client-cert-file=/etc/kubernetes/ssl/proxy-client.pem \\
  --proxy-client-key-file=/etc/kubernetes/ssl/proxy-client-key.pem \\
  --runtime-config=api/all=true \\
  --requestheader-client-ca-file=/etc/kubernetes/ssl/ca.pem \\
  --requestheader-allowed-names=aggregator \\
  --requestheader-extra-headers-prefix=X-Remote-Extra- \\
  --requestheader-group-headers=X-Remote-Group \\
  --requestheader-username-headers=X-Remote-User \\
  --tls-cert-file=/etc/kubernetes/ssl/kubernetes.pem \\
  --tls-private-key-file=/etc/kubernetes/ssl/kubernetes-key.pem \\
  --v=1
Restart=on-failure
RestartSec=5

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

5.2 配置 kube-controller-manager

> 下面的所有命令都是运行在每个master节点的，我们的实例中是 node-1 和 node-2.

```
# 准备kubeconfig配置文件
$ mv kube-controller-manager.kubeconfig /etc/kubernetes/

# 创建 kube-controller-manager.service
$ cat <<EOF > /etc/systemd/system/kube-controller-manager.service
[Unit]
Description=Kubernetes Controller Manager
Documentation=https://github.com/kubernetes/kubernetes

[Service]
ExecStart=/usr/local/bin/kube-controller-manager \\
  --bind-address=0.0.0.0 \\
  --cluster-cidr=10.200.0.0/16 \\
  --cluster-name=kubernetes \\
  --cluster-signing-cert-file=/etc/kubernetes/ssl/ca.pem \\
  --cluster-signing-key-file=/etc/kubernetes/ssl/ca-key.pem \\
  --cluster-signing-duration=876000h0m0s \\
  --kubeconfig=/etc/kubernetes/kube-controller-manager.kubeconfig \\
  --leader-elect=true \\
  --root-ca-file=/etc/kubernetes/ssl/ca.pem \\
  --service-account-private-key-file=/etc/kubernetes/ssl/service-account-key.pem \\
  --service-cluster-ip-range=10.233.0.0/16 \\
  --use-service-account-credentials=true \\
  --v=1
Restart=on-failure
RestartSec=5

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

5.3 配置 配置kube-scheduler

> 下面的所有命令都是运行在每个master节点的，我们的实例中是 node-1 和 node-2.

```
# 准备kubeconfig配置文件
$ mv kube-scheduler.kubeconfig /etc/kubernetes

# 创建 scheduler service 文件
$ cat <<EOF > /etc/systemd/system/kube-scheduler.service
[Unit]
Description=Kubernetes Scheduler
Documentation=https://github.com/kubernetes/kubernetes
```

```
[Service]
ExecStart=/usr/local/bin/kube-scheduler \\
  --authentication-kubeconfig=/etc/kubernetes/kube-scheduler.kubeconfig \\
  --authorization-kubeconfig=/etc/kubernetes/kube-scheduler.kubeconfig \\
  --kubeconfig=/etc/kubernetes/kube-scheduler.kubeconfig \\
  --leader-elect=true \\
  --bind-address=0.0.0.0 \\
  --port=0 \\
  --v=1
Restart=on-failure
RestartSec=5

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

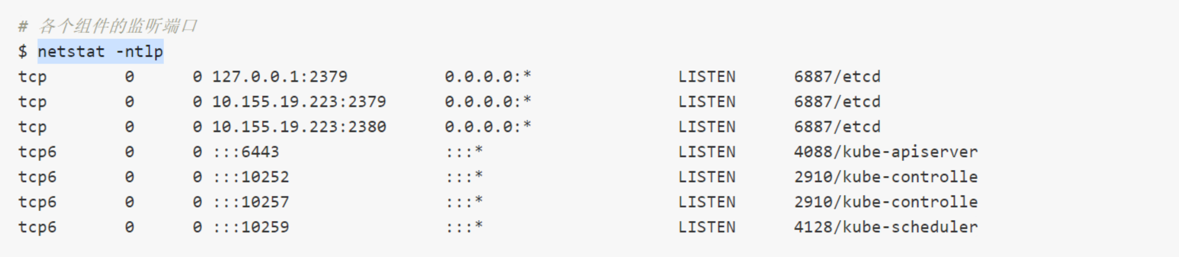
5.4 启动服务

> 下面的所有命令都是运行在每个master节点的，我们的实例中是 node-1 和 node-2.

```
systemctl daemon-reload \
& systemctl enable kube-apiserver
& systemctl enable kube-controller-manager \
& systemctl enable kube-scheduler \
& systemctl restart kube-apiserver \
& systemctl restart kube-controller-manager \
& systemctl restart kube-scheduler
```

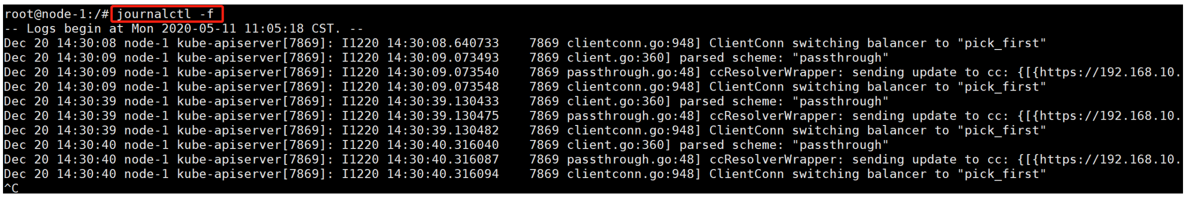
5.5 服务验证

> 在任意master服务器上都可以操作。
>
> netstat -ntlp查看一下组件都表示启动成功
>
> etcd、apiserver、controller、scheduler四大组件。



```
# 各个组件的监听端口
$ netstat -ntlp
tcp        0      0 127.0.0.1:2379          0.0.0.0:*               LISTEN      6887/etcd
tcp        0      0 10.155.19.223:2379      0.0.0.0:*               LISTEN      6887/etcd
tcp        0      0 10.155.19.223:2380      0.0.0.0:*               LISTEN      6887/etcd
tcp6       0      0 :::6443                 :::*                    LISTEN      4088/kube-apiserver
tcp6       0      0 :::10252                :::*                    LISTEN      2910/kube-controlle
tcp6       0      0 :::10257                :::*                    LISTEN      2910/kube-controlle
tcp6       0      0 :::10259                :::*                    LISTEN      4128/kube-scheduler
```

> 查看系统日志 journalctl -f



```
root@node-1:/# journalctl -f
-- Logs begin at Mon 2020-05-11 11:05:18 CST. --
Dec 20 14:30:08 node-1 kube-apiserver[7869]: I1220 14:30:08.640733    7869 clientconn.go:948] ClientConn switching balancer to "pick_first"
Dec 20 14:30:09 node-1 kube-apiserver[7869]: I1220 14:30:09.073493    7869 client.go:360] parsed scheme: "passthrough"
Dec 20 14:30:09 node-1 kube-apiserver[7869]: I1220 14:30:09.073540    7869 passthrough.go:48] ccResolverWrapper: sending update to cc: {[{https://192.168.10.
Dec 20 14:30:09 node-1 kube-apiserver[7869]: I1220 14:30:09.073548    7869 clientconn.go:948] ClientConn switching balancer to "pick_first"
Dec 20 14:30:39 node-1 kube-apiserver[7869]: I1220 14:30:39.130433    7869 client.go:360] parsed scheme: "passthrough"
Dec 20 14:30:39 node-1 kube-apiserver[7869]: I1220 14:30:39.130475    7869 passthrough.go:48] ccResolverWrapper: sending update to cc: {[{https://192.168.10.
Dec 20 14:30:39 node-1 kube-apiserver[7869]: I1220 14:30:39.130482    7869 clientconn.go:948] ClientConn switching balancer to "pick_first"
Dec 20 14:30:40 node-1 kube-apiserver[7869]: I1220 14:30:40.316040    7869 client.go:360] parsed scheme: "passthrough"
Dec 20 14:30:40 node-1 kube-apiserver[7869]: I1220 14:30:40.316087    7869 passthrough.go:48] ccResolverWrapper: sending update to cc: {[{https://192.168.10.
Dec 20 14:30:40 node-1 kube-apiserver[7869]: I1220 14:30:40.316094    7869 clientconn.go:948] ClientConn switching balancer to "pick_first"
^C
```

5.6 配置kubectl

> kubectl是用来管理kubernetes集群的客户端工具，前面我们已经下载到了所有的master节点。下面我们来配置这个工具，让它可以使用。

```
# 创建kubectl的配置目录
$ mkdir ~/.kube/
# 把管理员的配置文件移动到kubectl的默认目录
$ mv ~/admin.kubeconfig ~/.kube/config
# 测试
$ kubectl get nodes
```

> 在执行 kubectl exec、run、logs 等命令时，apiserver 会转发到 kubelet。这里定义 RBAC 规则，授权 apiserver 调用 kubelet API。

```
$ kubectl create clusterrolebinding kube-apiserver:kubelet-apis --
clusterrole=system:kubelet-api-admin --user kubernetes
```

6 部署kubernetes worker节点

> 这部分我们部署kubernetes的工作节点。实例中我们有两个工作节点，一个是独立的工作节点，一个是跟master在一起的节点。
> 在每个节点上我们会部署kubelet、kube-proxy、container runtime、cni、nginx-proxy。
>
> 注意：下面的操作需要在每一个worker节点上操作。

6.1 安装Containerd

> 软件包下载

```
# 设定containerd的版本号
$ VERSION=1.4.3
# 下载压缩包
$ wget
https://github.com/containerd/containerd/releases/download/v${VERSION}/cri-
containerd-cni-${VERSION}-linux-amd64.tar.gz
```

> 整理压缩文件
>
> 下载后的文件是一个tar.gz，是一个allinone的包，包括了runc、circtl、ctr、containerd等容器运行时以及cni相关的文件，解压缩到一个独立的目录中

```
# 解压缩
$ tar -xvf cri-containerd-cni-${VERSION}-linux-amd64.tar.gz
# 复制需要的文件
$ cp etc/crictl.yaml /etc/
$ cp etc/systemd/system/containerd.service /etc/systemd/system/
$ cp -r usr /
```

> containerd配置文件

```
$ mkdir -p /etc/containerd
# 默认配置生成配置文件
$ containerd config default > /etc/containerd/config.toml
# 定制化配置（可选）
$ vi /etc/containerd/config.toml
```

> 启动containerd

```
$ systemctl enable containerd \
& systemctl restart containerd \
& systemctl status containerd
```

6.2 配置kubelete

> 准备kubelet相关配置

```
$ mkdir -p /etc/kubernetes/ssl/
$ mv ${HOSTNAME}-key.pem ${HOSTNAME}.pem ca.pem ca-key.pem /etc/kubernetes/ssl/
$ mv ${HOSTNAME}.kubeconfig /etc/kubernetes/kubeconfig
$ IP=10.155.19.64
# 写入kubelet配置文件
$ cat <<EOF > /etc/kubernetes/kubelet-config.yaml
kind: KubeletConfiguration
apiVersion: kubelet.config.k8s.io/v1beta1
authentication:
  anonymous:
    enabled: false
  webhook:
    enabled: true
  x509:
    clientCAFile: "/etc/kubernetes/ssl/ca.pem"
authorization:
  mode: Webhook
clusterDomain: "cluster.local"
clusterDNS:
  - "169.254.25.10"
podCIDR: "10.200.0.0/16"
address: ${IP}
readOnlyPort: 0
staticPodPath: /etc/kubernetes/manifests
healthzPort: 10248
healthzBindAddress: 127.0.0.1
kubeletCgroups: /systemd/system.slice
resolvConf: "/etc/resolv.conf"
runtimeRequestTimeout: "15m"
kubeReserved:
  cpu: 200m
  memory: 512M
tlsCertFile: "/etc/kubernetes/ssl/${HOSTNAME}.pem"
tlsPrivateKeyFile: "/etc/kubernetes/ssl/${HOSTNAME}-key.pem"
EOF
```

> 配置kubelet服务

```
$ cat <<EOF > /etc/systemd/system/kubelet.service
[Unit]
Description=Kubernetes Kubelet
Documentation=https://github.com/kubernetes/kubernetes
After=containerd.service
Requires=containerd.service

[Service]
ExecStart=/usr/local/bin/kubelet \\
  --config=/etc/kubernetes/kubelet-config.yaml \\
  --container-runtime=remote \\
  --container-runtime-endpoint=unix:///var/run/containerd/containerd.sock \\
  --image-pull-progress-deadline=2m \\
  --kubeconfig=/etc/kubernetes/kubeconfig \\
  --network-plugin=cni \\
  --node-ip=${IP} \\
  --register-node=true \\
  --v=2
Restart=on-failure
RestartSec=5

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

6.3 配置nginx-proxy

`nginx-proxy是一个用于worker节点访问apiserver的一个代理，是apiserver一个优雅的高可用方案，它使用kubelet的staticpod方式启动，让每个节点都可以均衡的访问到每个apiserver服务，优雅的替代了通过虚拟ip访问apiserver的方式。`

> nginx-proxy只需要在没有apiserver的worker节点上部署，既只有worker角色的服务器上，拿我们的机器就是只有node-3上才会部署。

> nginx配置文件

```
$ mkdir -p /etc/nginx
# master ip列表
$ MASTER_IPS=(10.155.19.223 10.155.19.64)
# 执行前请先copy一份，并修改好upstream的 'server' 部分配置
$ cat <<EOF > /etc/nginx/nginx.conf
error_log stderr notice;

worker_processes 2;
worker_rlimit_nofile 130048;
worker_shutdown_timeout 10s;

events {
  multi_accept on;
  use epoll;
  worker_connections 16384;
}

stream {
  upstream kube_apiserver {
    least_conn;
    #这个地方注意，配置成所有的master节点的代理。
    server ${MASTER_IPS[0]}:6443;
    server ${MASTER_IPS[1]}:6443;
    ...
    server ${MASTER_IPS[N]}:6443;
  }

  server {
    listen        127.0.0.1:6443;
    proxy_pass    kube_apiserver;
    proxy_timeout 10m;
    proxy_connect_timeout 1s;
  }
}

http {
  aio threads;
  aio_write on;
  tcp_nopush on;
  tcp_nodelay on;

  keepalive_timeout 5m;
  keepalive_requests 100;
  reset_timedout_connection on;
  server_tokens off;
  autoindex off;

  server {
    listen 8081;
    location /healthz {
      access_log off;
      return 200;
    }
    location /stub_status {
      stub_status on;
      access_log off;
    }
  }
}
EOF
```

> nginx manifest

```
$ mkdir -p /etc/kubernetes/manifests/
$ cat <<EOF > /etc/kubernetes/manifests/nginx-proxy.yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-proxy
  namespace: kube-system
  labels:
    addonmanager.kubernetes.io/mode: Reconcile
    k8s-app: kube-nginx
spec:
  hostNetwork: true
  dnsPolicy: ClusterFirstWithHostNet
  nodeSelector:
    kubernetes.io/os: linux
  priorityClassName: system-node-critical
  containers:
  - name: nginx-proxy
    image: docker.io/library/nginx:1.19
    imagePullPolicy: IfNotPresent
    resources:
      requests:
        cpu: 25m
        memory: 32M
    securityContext:
      privileged: true
    livenessProbe:
      httpGet:
        path: /healthz
        port: 8081
    readinessProbe:
      httpGet:
        path: /healthz
        port: 8081
    volumeMounts:
    - mountPath: /etc/nginx
      name: etc-nginx
      readOnly: true
  volumes:
  - name: etc-nginx
    hostPath:
      path: /etc/nginx
EOF
```

6.4 配置kube-proxy

`这个需要在所有的worker角色服务器上做部署`

> 配置文件

```
$ mv kube-proxy.kubeconfig /etc/kubernetes/
# 创建 kube-proxy-config.yaml
$ cat <<EOF > /etc/kubernetes/kube-proxy-config.yaml
apiVersion: kubeproxy.config.k8s.io/v1alpha1
kind: KubeProxyConfiguration
bindAddress: 0.0.0.0
clientConnection:
  kubeconfig: "/etc/kubernetes/kube-proxy.kubeconfig"
clusterCIDR: "10.200.0.0/16"
mode: ipvs
EOF
```

> kube-proxy服务文件

```
$ cat <<EOF > /etc/systemd/system/kube-proxy.service
[Unit]
Description=Kubernetes Kube Proxy
Documentation=https://github.com/kubernetes/kubernetes

[Service]
ExecStart=/usr/local/bin/kube-proxy \\
  --config=/etc/kubernetes/kube-proxy-config.yaml
Restart=on-failure
RestartSec=5

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

> 启动服务器

```
$ systemctl daemon-reload \
& systemctl enable kubelet kube-proxy \
systemctl restart kubelet kube-proxy
```

> 查看相关日志

```
$ journalctl -f -u kubelet
$ journalctl -f -u kube-proxy
```

> 启动前可以先手动下载镜像（服务器如果不能访问外网的情况下）

在每个worker角色下载pause镜像

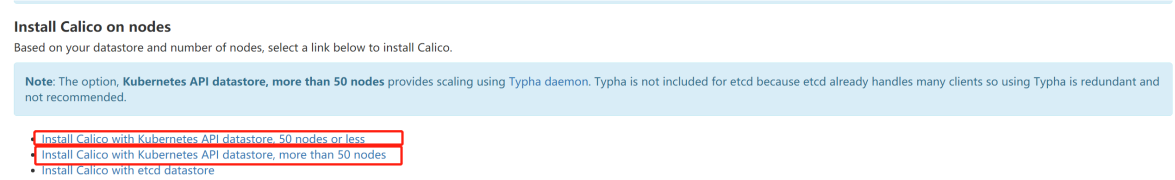
```
#拉取pause镜像
$ crictl pull registry.cn-hangzhou.aliyuncs.com/kubernetes-kubespray/pause:3.2
#将非官网pause容器转成官网标准容器名称
$ ctr -n k8s.io i tag  registry.cn-hangzhou.aliyuncs.com/kubernetes-kubespray/pause:3.2 k8s.gcr.io/pause:3.2
```

7 网络插件-Calico

官网CNI参考

> 1、下载文件说明
>
> ```
> curl https://docs.projectcalico.org/manifests/calico.yaml -O
> ```

文档中有两个配置，50以下节点和50以上节点，它们的主要区别在于这个：typha。
当节点数比较多的情况下，Calico 的 Felix组件可通过 Typha 直接和 Etcd 进行数据交互，不通过 kube-apiserver，降低kube-apiserver的压力。大家根据自己的实际情况选择下载。
下载后的文件是一个all-in-one的yaml文件，我们只需要在此基础上做少许修改即可。这里演示下载第一个就好了。



Install Calico on nodes

Based on your datastore and number of nodes, select a link below to install Calico.

Note: The option, **Kubernetes API datastore, more than 50 nodes** provides scaling using Typha daemon. Typha is not included for etcd because etcd already handles many clients so using Typha is redundant and not recommended.

- Install Calico with Kubernetes API datastore, 50 nodes or less
- Install Calico with Kubernetes API datastore, more than 50 nodes
- Install Calico with etcd datastore

> 2、修改IP自动发现

当kubelet的启动参数中存在--node-ip的时候，以host-network模式启动的pod的status.hostIP字段就会自动填入kubelet中指定的ip地址。

- 修改前

  ```
  - name: IP
    value: "autodetect"
  ```

- 修改后

  ```
  - name: IP
    valueFrom:
      fieldRef:
        fieldPath: status.hostIP
  ```

> 3、修改CIDR

- 修改前（源文件是杯注释了的）

  ```
  # - name: CALICO_IPV4POOL_CIDR
  #   value: "192.168.0.0/16"
  ```

- 修改后（修改成你自己的value哦，我这里是10.200.0.0/16。也就是前面证书以及组件认证的时候配置的IP）

  ```
  - name: CALICO_IPV4POOL_CIDR
    value: "10.200.0.0/16"
  ```

> 4、启动calico网络插件（在一个master上操作就好了）

```
kubectl apply -f calico.yaml
```

8 DNS插件-CoreDNS

> 这部分我们部署kubernetes的DNS插件 - CoreDNS。
>
> 在早期的版本中dns组件以pod形式独立运行，为集群提供dns服务，所有的pod都会请求同一个dns服务。
> 从kubernetes 1.18版本开始NodeLocal DnsCache功能进入stable状态。
> NodeLocal DNSCache通过daemon-set的形式运行在每个工作节点，作为节点上pod的dns缓存代理，从而避免了iptables的DNAT规则和connection tracking。极大提升了dns的性能。

以下两文件下载地址

1、部署CoreDNS

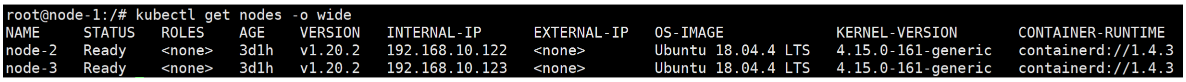
```
# 设置 coredns 的 cluster-ip
$ COREDNS_CLUSTER_IP=10.233.0.10
# 下载coredns配置all-in-one（addons/coredns.yaml）
# 替换cluster-ip
$ sed -i "s/\${COREDNS_CLUSTER_IP}/${COREDNS_CLUSTER_IP}/g" coredns.yaml
# 创建 coredns
$ kubectl apply -f coredns.yaml
```

2、部署NodeLocal DNSCache

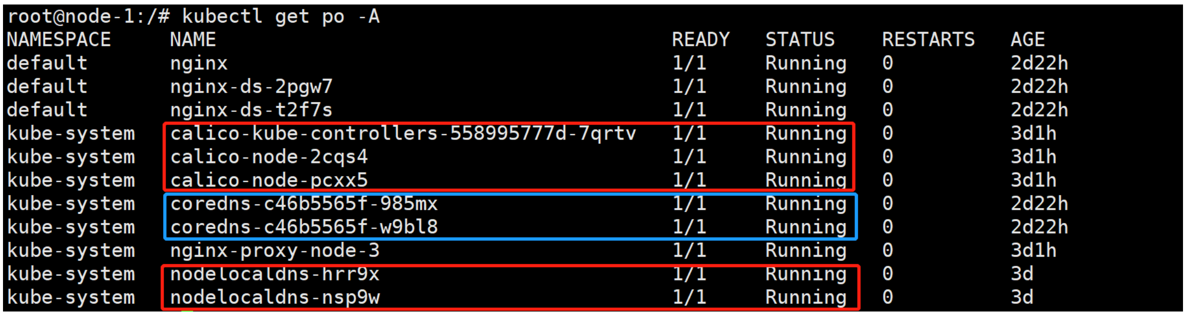
```
# 设置 coredns 的 cluster-ip
$ COREDNS_CLUSTER_IP=10.233.0.10
# 下载nodelocaldns配置all-in-one(addons/nodelocaldns.yaml)
# 替换cluster-ip
$ sed -i "s/\${COREDNS_CLUSTER_IP}/${COREDNS_CLUSTER_IP}/g" nodelocaldns.yaml
# 创建 nodelocaldns
$ kubectl apply -f nodelocaldns.yaml
```

3、部署查看

```
kubectl get nodes -o wide #查看集群状态
kubectl get po -A #查看所有的POD
```



```
root@node-1:/# kubectl get nodes -o wide
NAME     STATUS   ROLES    AGE    VERSION   INTERNAL-IP      EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION       CONTAINER-RUNTIME
node-2   Ready    <none>   3d1h   v1.20.2   192.168.10.122   <none>        Ubuntu 18.04.4 LTS   4.15.0-161-generic   containerd://1.4.3
node-3   Ready    <none>   3d1h   v1.20.2   192.168.10.123   <none>        Ubuntu 18.04.4 LTS   4.15.0-161-generic   containerd://1.4.3
```



```
root@node-1:/# kubectl get po -A
NAMESPACE     NAME                                       READY   STATUS    RESTARTS   AGE
default       nginx                                      1/1     Running   0          2d22h
default       nginx-ds-2pgw7                             1/1     Running   0          2d22h
default       nginx-ds-t2f7s                             1/1     Running   0          2d22h
kube-system   calico-kube-controllers-558995777d-7qrtv   1/1     Running   0          3d1h
kube-system   calico-node-2cqs4                          1/1     Running   0          3d1h
kube-system   calico-node-pcxx5                          1/1     Running   0          3d1h
kube-system   coredns-c46b5565f-985mx                    1/1     Running   0          2d22h
kube-system   coredns-c46b5565f-w9bl8                    1/1     Running   0          2d22h
kube-system   nginx-proxy-node-3                         1/1     Running   0          3d1h
kube-system   nodelocaldns-hrr9x                         1/1     Running   0          3d
kube-system   nodelocaldns-nsp9w                         1/1     Running   0          3d
```