K8S集群部署手册

一、 k8s部署环境规划

模块	安装的系统	主机名	IP	模块功能	内存大小	硬盘大小
1、ETCD集群安装	CentOS 7.4_64	yds-dev-svc01-etcd01	192.168.10.50		1G建议2G，下同	50G
		yds-dev-svc01-etcd02	192.168.10.51		1G	50G
		yds-dev-svc01-etcd03	192.168.10.52		1G	50G
2、apiserver高可用；3、master中的docker安装；4、master中flannel网络插件	CentOS 7.4_64	yds-dev-svc01-master01	192.168.10.53		1G	50G
		yds-dev-svc01-master02	192.168.10.54		1G	50G
5、node中docker安装及配置 6、node中flannel网络插件 7、安装kubelet和proxy	CentOS 7.4_64	yds-dev-svc02-node01	192.168.10.56		1G	50G
		yds-dev-svc02-node02	192.168.10.57		1G	50G
		yds-dev-svc02-node03	192.168.10.58		1G	50G
8、安装dashboard						

二、 Kubernetes1.9生产环境 ——ETCD高可用集群部署

2.1、 升级内核 (每台节点机都要操作)

```
[root@k8s-master01 ~]# uname -r 3.10.0-862.el7.x86_64

cat <<EOF >> /etc/resolv.conf
nameserver 61.139.2.69
nameserver 114.114.114.114
EOF
```

安装ELRepo到CentOS中（此处是从网络源获取资源）

```
[root@k8s-master01 ~]# rpm -Uvh http://www.elrepo.org/elrepo-release-7.0-3.el7.elrepo.noarch.rpm
```

安装完成后检查 /boot/grub2/grub.cfg 中对应内核 menuentry 中是否包含 initrd16 配置，如果没有，再安装一次！

```
[root@localhost ~]# cat /boot/grub2/grub.cfg |grep initrd16
```

安装kernel-lt（lt=long-term） （此处是从网络源获取资源）

```
[root@k8s-master01 ~]# yum --enablerepo=elrepo-kernel install -y kernel-lt
```

设置开机从新内核启动（编辑grub.conf文件，修改Grub引导顺序）

```
[root@k8s-master01 ~]# grub2-set-default 0
```

重启机器

```
[root@k8s-master01 ~]# init 6
```

安装内核源文件（在升级完内核并重启机器后执行，也可以不用执行这一步。可选）：

```
[root@k8s-master01 ~]# yum --enablerepo=elrepo-kernel install kernel-lt-
devel-$(uname -r) kernel-lt-headers-$(uname -r)
```

重新查看内核

```
[root@k8s-master01 ~]# uname -r4.4.180-2.el7.elrepo.x86_64
```

2.2、ETCD安装基础环境

2.2.1、 服务器初始配置

k8s-etcd01 IP地址配置

修改网络配置文件

```
[root@k8s-etcd01 ~]# cat /etc/sysconfig/network-scripts/ifcfg-ens33
TYPE=Ethernet
PROXY_METHOD=none
BROWSER_ONLY=no
BOOTPROTO=static
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
IPV6INIT=yes
IPV6_AUTOCONF=yes
IPV6_DEFROUTE=yes
IPV6_FAILURE_FATAL=no
IPV6_ADDR_GEN_MODE=stable-privacy
NAME=ens33
\#UUID=7d6fb2ed-364c-415f-9b02-0e54436ff1ec
DEVICE=ens33
ONBOOT=yes
IPADDR=192.168.10.50
NETMASK=255.255.255.0
GATEWAY=192.168.10.1
DNS1=192.168.10.10
DNS2=114.114.114.114
[root@localhost ~]#ip a
```

k8s-etcd01主机名配置

```
hostnamectl set-hostname k8s-etcd01
hostnamectl
```

配置完成后，重新登录一下

k8s-etcd02 IP*地址配置***
修改网络配置文件

```
[root@k8s-etcd02 ~]*#* *cat /etc/sysconfig/network-scripts/ifcfg-ens33*
```

查看网络配置信息。

```
[root@localhost ~]# ip a
```

k8s-etcd02 主机名配置

```
hostnamectl set-hostname k8s-etcd02

hostnamectl
```

配置完成后，重新登录一下

k8s-etcd03 IP地址配置

修改网络配置文件

```
[root@k8s-etcd03 ~]*#* *cat /etc/sysconfig/network-scripts/ifcfg-ens33*
```

查看网络配置信息。

```
[root@k8s-etcd03 ~]# ip a
```

k8s-etcd03 主机名配置

```
hostnamectl set-hostname k8s-etcd03
hostnamectl
```

配置完成后，重新登录一下

在k8s-etcd01上操作：

首先建立无密码ssh**信任关系：**

本篇部署文档有很有操作都是在*k8s-etcd01*节点上执行，然后远程分发文件到其他节点机器上并远程执行命令，所以需要添加该节点到其它节点的ssh信任关系。

```
[root@k8s-etcd01 ~]# ssh-keygen -t rsa
[root@k8s-etcd01 ~]# cp /root/.ssh/id_rsa.pub /root/.ssh/authorized_keys
[root@k8s-etcd01 ~]# ssh-copy-id -i /root/.ssh/id_rsa.pub -p22 root@
192.168.211.154
[root@k8s-etcd01 ~]# ssh-copy-id -i /root/.ssh/id_rsa.pub -p22 root@
192.168.211.153
[root@k8s-etcd01 ~]# ssh-copy-id -i /root/.ssh/id_rsa.pub -p22 root@
192.168.211.152
[root@k8s-etcd01 ~]# ssh-copy-id -i /root/.ssh/id_rsa.pub -p22 root@
192.168.211.151
[root@k8s-etcd01 ~]# ssh-copy-id -i /root/.ssh/id_rsa.pub -p22
root@192.168.211.150
[root@k8s-etcd01 ~]# ssh-copy-id -i /root/.ssh/id_rsa.pub -p22 root@
192.168.211.149
[root@k8s-etcd01 ~]# ssh-copy-id -i /root/.ssh/id_rsa.pub -p22 root@
192.168.211.156
```

以上信任关系设置后，最好手动验证下本节点登陆到其他节点的ssh无密码信任关系

```
[root@k8s-etcd01 ~]# ssh root@192.168.211.154
[root@k8s-etcd02 ~]# exit
```

在k8s-etcd01上操作:

然后，创建一个变量脚本文件

```
[root@k8s-etcd01 ~]# mkdir -p /etc/kubernetes/
[root@k8s-etcd01 ~]# vi /etc/kubernetes/environment.sh
\#!/usr/bin/bash
\# 生成 EncryptionConfig 所需的加密 key
export ENCRYPTION_KEY=$(head -c 32 /dev/urandom | base64)
\# 集群中所有节点机器IP数组（master,node,etcd节点）
export NODE_ALL_IPS=(192.168.211.155 192.168.211.154 192.168.211.153
192.168.211.152 192.168.211.151 192.168.211.150 192.168.211.149 192.168.211.156)
\# 集群中所有节点IP对应的主机名数组
export NODE_ALL_NAMES=(k8s-etcd01 k8s-etcd02 k8s-etcd03 k8s-master01 k8s-
master02 k8s-node01 k8s-node02 k8s-node03)
\# 集群中所有master节点集群IP数组
export NODE_MASTER_IPS=(192.168.211.152 192.168.211.151)
\# 集群中master节点IP对应的主机名数组
export NODE_MASTER_NAMES=(k8s-master01 k8s-master02)
\# 集群中所有node节点集群IP数组
export NODE_NODE_IPS=(192.168.211.150 192.168.211.149 192.168.211.156)
\# 集群中node节点IP对应的主机名数组
export NODE_NODE_NAMES=(k8s-node01 k8s-node02 k8s-node03)
\# 集群中所有etcd节点集群IP数组
export NODE_ETCD_IPS=(192.168.211.155 192.168.211.154 192.168.211.153)
\# 集群中etcd节点IP对应的主机名数组
export NODE_ETCD_NAMES=(k8s-etcd01 k8s-etcd02 k8s-etcd03)
\# etcd 集群服务地址列表
export
ETCD_ENDPOINTS="https://192.168.211.155:2379,https://192.168.211.154:2379,https:
//192.168.211.153:2379"
\# etcd 集群间通信的 IP 和端口
export ETCD_NODES="k8s-etcd01=https://192.168.211.155:2380,k8s-
etcd02=https://192.168.211.154:2380,k8s-etcd03=https://192.168.211.153:2380"
\# kube-apiserver 的反向代理(地址端口.这里也就是keepalived的VIP地址)
export KUBE_APISERVER="https://192.168.211.157:6443"
\# 节点间互联网络接口名称. 这里我所有的centos7节点机的网卡设备是ens33，而不是eth0
export IFACE="ens33"
\# etcd 数据目录
export ETCD_DATA_DIR="/var/lib/etcd/data/"
\# etcd WAL 目录，建议是 SSD 磁盘分区，或者和 ETCD_DATA_DIR 不同的磁盘分区
export ETCD_WAL_DIR="/var/lib/etcd/wal/"
\## 以下参数一般不需要修改
\# TLS Bootstrapping 使用的 Token，可以使用命令 head -c 16 /dev/urandom | od -An -t
x | tr -d ' ' 生成
BOOTSTRAP_TOKEN="5cc80ff9854de087a636deee421004e5"
\# 最好使用 当前未用的网段 来定义服务网段和 Pod 网段
\# 服务网段，部署前路由不可达，部署后集群内路由可达(kube-proxy 保证)
SERVICE_CIDR="10.254.0.0/16"
\# Pod 网段，建议 /16 段地址，部署前路由不可达，部署后集群内路由可达(flanneld 保证)
CLUSTER_CIDR="172.30.0.0/16"
\# 服务端口范围 (NodePort Range)
export NODE_PORT_RANGE="30000-32767"
```

拷贝到所有节点：

```
[root@k8s-etcd01 ~]# source /etc/kubernetes/environment.sh
[root@k8s-etcd01 ~]# for node_all_ip in ${NODE_ALL_IPS[@]}
 do
  echo ">>> ${node_all_ip}"
  ssh root@${node_all_ip} "mkdir -p /etc/kubernetes/"
  scp -r /etc/kubernetes/environment.sh root@${node_all_ip}:/etc/kubernetes/
 done
```

在k8s-etcd01操作：

所有节点关闭selinux

```
[root@k8s-etcd01 ~]# source /etc/kubernetes/environment.sh
[root@k8s-etcd01 ~]# for node_all_ip in ${NODE_ALL_IPS[@]}
 do
  echo ">>> ${node_all_ip}"
  ssh root@${node_all_ip} "setenforce 0"
  ssh root@${node_all_ip} "sed -i "s/^SELINUX=enforcing/SELINUX=disabled/g"
/etc/sysconfig/selinux"
  ssh root@${node_all_ip} "sed -i "s/^SELINUX=enforcing/SELINUX=disabled/g"
/etc/selinux/config"
  ssh root@${node_all_ip} "sed -i "s/^SELINUX=permissive/SELINUX=disabled/g"
/etc/sysconfig/selinux"
  ssh root@${node_all_ip} "sed -i "s/^SELINUX=permissive/SELINUX=disabled/g"
/etc/selinux/config"
  ssh root@${node_all_ip} "getenforce"
 done
```

在k8s-etcd01操作：

所有节点关闭交换分区swap

```
[root@k8s-etcd01 ~]# source /etc/kubernetes/environment.sh
[root@k8s-etcd01 ~]# for node_all_ip in ${NODE_ALL_IPS[@]}
 do
  echo ">>> ${node_all_ip}"
  ssh root@${node_all_ip} "swapoff -a"
  ssh root@${node_all_ip} "sed -i 's/.*swap.*/#&/' /etc/fstab"
  ssh root@${node_all_ip} "cat /etc/fstab"
 done
```

在k8s-etcd01操作：

所有节点设置内核

```
[root@k8s-etcd01 ~]# source /etc/kubernetes/environment.sh
[root@k8s-etcd01 ~]# for node_all_ip in ${NODE_ALL_IPS[@]}
 do
  echo ">>> ${node_all_ip}"
  ssh root@${node_all_ip} "cat > /etc/sysctl.d/k8s.conf <<EOF
net.bridge.bridge-nf-call-iptables=1
net.bridge.bridge-nf-call-ip6tables=1
```

```
net.ipv4.ip_forward=1
net.ipv4.tcp_tw_recycle=0
vm.swappiness=0
vm.overcommit_memory=1
vm.panic_on_oom=0
fs.inotify.max_user_instances=8192
fs.inotify.max_user_watches=1048576
fs.file-max=52706963
fs.nr_open=52706963
net.ipv6.conf.all.disable_ipv6=1
net.netfilter.nf_conntrack_max=2310720
EOF"
  ssh root@${node_all_ip} "sysctl -p /etc/sysctl.d/k8s.conf"
  ssh root@${node_all_ip} "cat /etc/sysctl.d/k8s.conf"
 done
```

这里需要注意：

必须关闭 tcp_tw_recycle，否则和 NAT 冲突，会导致服务不通；

关闭 IPV6，防止触发 docker BUG；

2.2.2、ETCD环境配置

在k8s-etcd01操作：

所有节点做地址映射

```
[root@k8s-etcd01 ~]# source /etc/kubernetes/environment.sh
[root@k8s-etcd01 ~]# for node_all_ip in ${NODE_ALL_IPS[@]
 do
  echo ">>> ${node_all_ip}"
  ssh root@${node_all_ip} "cat <<EOF >> /etc/hosts
192.168.211.155 k8s-etcd01
192.168.211.154 k8s-etcd02
192.168.211.153 k8s-etcd03
192.168.211.152 k8s-master01
192.168.211.151 k8s-master02
192.168.211.150 k8s-node01
192.168.211.149 k8s-node02
192.168.211.156 k8s-node03
EOF"
  ssh root@${node_all_ip} "source /etc/kubernetes/environment.sh"
  ssh root@${node_all_ip} "echo ${NODE_ALL_IPS[@]}
echo ${NODE_NODE_IPS[@]}
echo ${NODE_ETCD_IPS[@]}
echo ${NODE_MASTER_IPS[@]}"
  ssh root@${node_all_ip} "cat /etc/hosts"
 done
```

在k8s-etcd01操作：

设置三台ETCD服务器环境

```
[root@k8s-etcd01 ~]# source /etc/kubernetes/environment.sh
[root@k8s-etcd01 ~]# for node_etcd_ip in ${NODE_ETCD_IPS[@]}
 do
  echo ">>> ${node_etcd_ip}"
  ssh root@${node_etcd_ip} "cat <<EOF >> ~/.bash_profile
export NODE_IP=${node_etcd_ip}
EOF"
  ssh root@${node_etcd_ip} "cat ~/.bash_profile"
 done
[root@k8s-etcd01 ~]# for node_etcd_name in ${NODE_ETCD_NAMES[@]}
 do
  echo ">>> ${node_etcd_name}"
  ssh root@${node_etcd_name} "cat <<EOF >> ~/.bash_profile
export NODE_NAME=${node_etcd_name}
EOF"
  ssh root@${node_etcd_name} "cat ~/.bash_profile"
 done
```

这里把三台服务器reboot**重启一下，让修改的主机名和地址映射生效。**

2.2.3、ETCD安装

ETCD证书配置

*在**每台主机上**操作：*

为了方便操作，我们挂载本地镜像，制作本地yum源：

```
\# mkdir /media/cdrom        //新建一个挂载点，一般建在opt目录下
\# mount /dev/cdrom /media/cdrom          //挂载镜像到挂载点
\# cd /etc/yum.repos.d            //进入yum源配置目录
\# ls                                 //查看当前目录中的文件
\# mkdir bak
\# mv * bak           //改变网络源文件目录，即停止使用网络源
\# vi CentOS-Local.repo        //开启本地源，新增加如下内容：
[CentOS-base-yum]
name=CentOS base yum Repository
baseurl=file://**/**media/cdrom
gpgcheck=0
enabled=1
\# yum clean all               //清空yum缓存
\# yum makecache            //制作新的缓存
```

*此部分证书部分可以在自己的电脑上面执行，也可以只在**k8s-etcd01**中执行。在这里，我们在**k8s-etcd01**在执行。*

安装证书生成工具* （此处是从网络源获取资源）

```
yum install -y wget
mkdir /home/key
cd /home/key

wget https:*//pkg.cfssl.org/R1.2/cfssl_linux-amd64*
chmod +x cfssl_linux-amd64
mv cfssl_linux-amd64 /usr/local/bin/cfssl

wget https:*//pkg.cfssl.org/R1.2/cfssljson_linux-amd64*
```

```
chmod +x cfssljson_linux-amd64
mv cfssljson_linux-amd64 /usr/local/bin/cfssljson

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

创建根证书配置文件

CA 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。

创建CA文件：

CA 配置文件用于配置根证书的使用场景 (profile) 和具体参数 (usage，过期时间、服务端认证、客户端认证、加密等)，后续在签名其它证书时需要指定特定场景。
signing: 表示该证书可用于签名其它证书；生成的 ca.pem 证书中 CA=TRUE;
server auth: 表示 client 可以用该 CA 对 server 提供的证书进行验证;
client auth: 表示 server 可以用该 CA 对 client 提供的证书进行验证;

```
[root@localhost key]# pwd
/home/key
cat > ca-config.json <<EOF
{
"signing": {
"default": {
 "expiry": "87600h"
},
"profiles": {
 "kubernetes": {
  "usages": [
  "signing",
  "key encipherment",
  "server auth",
  "client auth"
  ],
  "expiry": "87600h"
 }
}
}
}
EOF
```

创建证书签名请求文件：

这里可以根据你的需要修改CN和O。
“CN”：Common Name，kube-apiserver 从证书中提取该字段作为请求的用户名 (User Name)；浏览器使用该字段验证网站是否合法；
“O”：Organization，kube-apiserver 从证书中提取该字段作为请求用户所属的组 (Group)；

```
cat > ca-csr.json <<EOF
{
"CN": "kubernetes",
"key": {
"algo": "rsa",
"size": 2048
},
"names": [
```

```
{
 "C": "CN",
 "ST": "chengdu",
 "L": "chengdu",
 "O": "k8s",
 "OU": "System"
}
]
}
EOF

ls
```

配置说明：

CN：Common Name，kube-apiserver 从证书中提取该字段作为请求的用户名 (User Name)，浏览器使用该字段验证网站是否合法；

O：Organization，kube-apiserver 从证书中提取该字段作为请求用户所属的组 (Group)；

kube-apiserver 将提取的 User、Group 作为 RBAC 授权的用户标识；

生成CA证书和私钥

```
[root@localhost key]# pwd
/home/key
cfssl gencert -initca ca-csr.json | cfssljson -bare ca
[root@localhost key]# pwd
/home/key
ls
```

*** *分发证书文件到所有节点* ***

将生成的 CA 证书、秘钥文件、配置文件拷贝到所有节点的 /etc/kubernetes/ssl 目录下：

[root@k8s-etcd01 key]# cd /home/key

[root@k8s-etcd01 key]# source /etc/kubernetes/environment.sh

[root@k8s-etcd01 key]# for node_all_ip in ${NODE_ALL_IPS[@]}

do

echo ">>> ${node_all_ip}"

ssh root@${node_all_ip} "mkdir -p /etc/kubernetes/ssl"

scp ca*.pem ca-config.json root@${node_all_ip}:/etc/kubernetes/ssl

done

创建etcd证书签名请求

hosts 字段指定授权使用该证书的 etcd 节点 IP；
每个节点IP 都要在里面或者每个机器申请一个对应IP的证书

```
cat > etcd-csr.json <<EOF
{
 "CN": "etcd",
```

```
 "hosts": [
  "127.0.0.1",
  "192.168.211.155",
  "192.168.211.154",
  "192.168.211.153"
 ],
 "key": {
  "algo": "rsa",
  "size": 2048
 },
 "names": [
  {
   "C": "CN"
   "ST": "chengdu",
   "L": "chengdu",
   "O": "k8s",
   "OU": "System"
  }
 ]
}
EOF
```

配置说明：

hosts 字段指定授权使用该证书的 etcd 节点 IP 或域名列表，需要将 etcd 集群的三个节点 IP 都列在其中；

生成 etcd 证书和私钥

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

ls etcd
ls
```

以上证书生产完成。为了安全起见，需要将生成的证书及配置文件进行备份。

在k8s-etcd01中，创建/etc/etcd/ssl目录并分发到k8s-etcd02，k8s-etcd03中：

```
[root@k8s-etcd01 key]# cd /home/key
[root@k8s-etcd01 key]# source /etc/kubernetes/environment.sh
[root@k8s-etcd01 key]# for node_etcd_ip in ${NODE_ETCD_IPS[@]}
 do
  echo ">>> ${node_etcd_ip}"
  ssh root@${node_etcd_ip} "mkdir -p /etc/etcd/ssl"
  scp /home/key/ca*.pem etcd* root@${node_etcd_ip}:/etc/etcd/ssl
 done
```

下载ETCD安装文件
在k8s-etcd01中：

我们在这里使用的ETCD版本为3.2.18，如果你在安装的时候，也可以使用这个版本，当然，也可以使用更高的版本或其他版本。 在k8s-etcd01中下载ETCD，下载完成后，复制安装文件到k8s-etcd02和k8s-etcd03中。（此处是从网络源获取资源）

```
cd /home
wget https:*//github.com/coreos/etcd/releases/download/v3.2.18/etcd-v3.2.18-
linux-amd64.tar.gz*
tar -xvzf etcd-v3.2.18-linux-amd64.tar.gz
cd etcd-v3.2.18-linux-amd64
cp etcd* /usr/local/bin/
scp etcd* root@k8s-etcd02:/usr/local/bin/
scp etcd* root@k8s-etcd03:/usr/local/bin/
```

在k8s-etcd01中操作，分发到k8s-etcd02、k8s-etcd03中：

创建 etcd 的 systemd unit 文件

先创建ETCD工作目录/var/lib/etcd/data/（我们用脚本创建）

如果没有配置这个目录，会现现Failed at step CHDIR spawning /usr/local/bin/etcd: No such file or directory的错误信息。
在各个服务器执行以下命令创建systemd unit文件。

```
echo ${NODE_ETCD_NAMES[@]}
echo ${NODE_ETCD_IPS[@]}
echo ${ETCD_NODES}
```

生成ETCD**配置文件**
这里生成的配置文件有: /etc/etcd/etcd-key.conf, /etc/etcd/etcd.conf
网上大部分是把这两个配置文件和systemd unit文件存放在一起, 也可以参考这样的方法，看个人习惯。
/etc/etcd/etcd-key.conf：存放我们证书的配置信息。
/etc/etcd/etcd.conf：存放ETCD集群的配置信息。

创建etcd-key.conf：

```
cat > /etc/etcd/etcd-key.conf <<EOF
ETCD_KEY='--cert-file=/etc/etcd/ssl/etcd.pem --key-file=/etc/etcd/ssl/etcd-
key.pem --peer-cert-file=/etc/etcd/ssl/etcd.pem --peer-key-
file=/etc/etcd/ssl/etcd-key.pem --trusted-ca-file=/etc/etcd/ssl/ca.pem --peer-
trusted-ca-file=/etc/etcd/ssl/ca.pem'
EOF
```

创建etcd.conf模板：

```
cat > /etc/etcd/etcd.conf.template <<EOF
ETCD_NAME='--name=`##NODE_ETCD_NAME##`'
DATA_DIR='--data-dir=/var/lib/etcd/data/'
INITIAL_CLUSTER_STATE='--initial-cluster-state=new'
INITIAL_CLUSTER_TOKEN='--initial-cluster-token=etcd-cluster-0'
INITIAL_ADVERTISE_PEER_URLS='--initial-advertise-peer-
urls=https://`##NODE_ETCD_IP##`:2380'
LISTEN_PEER_URLS='--listen-peer-urls=https://`##NODE_ETCD_IP##`:2380'
LISTEN_CLIENT_URLS='--listen-client-
urls=https://`##NODE_ETCD_IP##`:2379,http://127.0.0.1:2379'
ADVERTISE_CLIENT_URLS='--advertise-client-urls=https://`##NODE_ETCD_IP##`:2379'
INITIAL_CLUSTER='--initial-cluster=${ETCD_NODES}'
EOF
```

配置说明：

WorkingDirectory、--data-dir：指定工作目录和数据目录为 ${ETCD_DATA_DIR}，需在启动服务前创建这个目录；

--wal-dir：指定 wal 目录，为了提高性能，一般使用 SSD 或者和 --data-dir 不同的磁盘；

--name：指定节点名称，当 --initial-cluster-state 值为 new 时，--name 的参数值必须位于 --initial-cluster 列表中；

--cert-file、--key-file：etcd server 与 client 通信时使用的证书和私钥；

--trusted-ca-file：签名 client 证书的 CA 证书，用于验证 client 证书；

--peer-cert-file、--peer-key-file：etcd 与 peer 通信使用的证书和私钥；

--peer-trusted-ca-file：签名 peer 证书的 CA 证书，用于验证 peer 证书；

创建/etc/systemd/system/etcd.service：

（因为有变量，我们不用cat**直接**vi保存）

```
vi /etc/systemd/system/etcd.service
[Unit]
Description=Etcd Server
After=network.target
After=network-online.target
Wants=network-online.target
Documentation=https://github.com/coreos
[Service]
Type=notify
WorkingDirectory=/var/lib/etcd/data/
EnvironmentFile=-/etc/etcd/etcd.conf
EnvironmentFile=-/etc/etcd/etcd-key.conf
ExecStart=/usr/local/bin/etcd \
  $ETCD_NAME \
  $DATA_DIR \
  $INITIAL_CLUSTER_STATE \
  $INITIAL_CLUSTER_TOKEN \
  $INITIAL_ADVERTISE_PEER_URLS \
  $LISTEN_PEER_URLS \
  $LISTEN_CLIENT_URLS \
  $ADVERTISE_CLIENT_URLS \
  $INITIAL_CLUSTER \
  $ETCD_KEY
Restart=on-failure
RestartSec=5
LimitNOFILE=65536
[Install]
WantedBy=multi-user.target
```

参数说明:
WorkingDirectory: ETCD工作目录

配置说明：

必须先创建 etcd 数据目录和工作目录;

etcd 进程首次启动时会等待其它节点的 etcd 加入集群，命令 systemctl start etcd 会卡住一段时间，为正常现象；

在k8s-etcd01中，把创建的etcd-key.conf、etcd.conf、etcd.service文件分发到k8s-etcd02，k8s-etcd03中：

分发 etcd-key.conf、etcd.conf、etcd.service配置文件到各etcd节点：

```
[root@k8s-etcd01 ~]# cd /etc/etcd/
[root@k8s-etcd01 kubernetes]# source /etc/kubernetes/environment.sh
[root@k8s-etcd01 kubernetes]# for node_etcd_ip in ${NODE_ETCD_IPS[@]}
 do
  echo` `">>> ${node_etcd_ip}"
ssh root@${node_etcd_ip} "mkdir -p /var/lib/etcd/data/"``
`  scp` /etc/etcd/etcd-key.conf` root@${node_etcd_ip}:`/etc/etcd/``
`  scp` /etc/etcd/etcd.conf.template`
root@${node_etcd_ip}:`/etc/etcd/etcd.conf.template``
`  ssh` `root@${node_etcd_ip} "sed` `-e "s/##NODE_ETCD_IP##/${node_etcd_ip}/"
`/etc/etcd/etcd.conf.template` > `/etc/etcd/etcd.conf`"```
`  scp` /etc/systemd/system/etcd.service`
root@${node_etcd_ip}:`/etc/systemd/system/``
` done```
[root@k8s-etcd01 kubernetes]# for node_etcd_name in ${NODE_ETCD_NAMES[@]}
 do
  echo` `">>> ${node_etcd_name}"
  ssh` `root@${node_etcd_name} "sed` `-i
"s/##NODE_ETCD_NAME##/${node_etcd_name}/"` /etc/etcd/etcd.conf`"
 done
```

在k8s-etcd01、k8s-etcd02、k8s-etcd03中：

测试环境中，我们直接关闭防火墙即可。

在每台机器上关闭防火墙，清理防火墙规则，设置默认转发策略：

```
systemctl stop firewalld
systemctl disable firewalld
iptables -F && iptables -X && iptables -F -t nat && iptables -X -t nat
iptables -P FORWARD ACCEPT
firewall-cmd --state
```

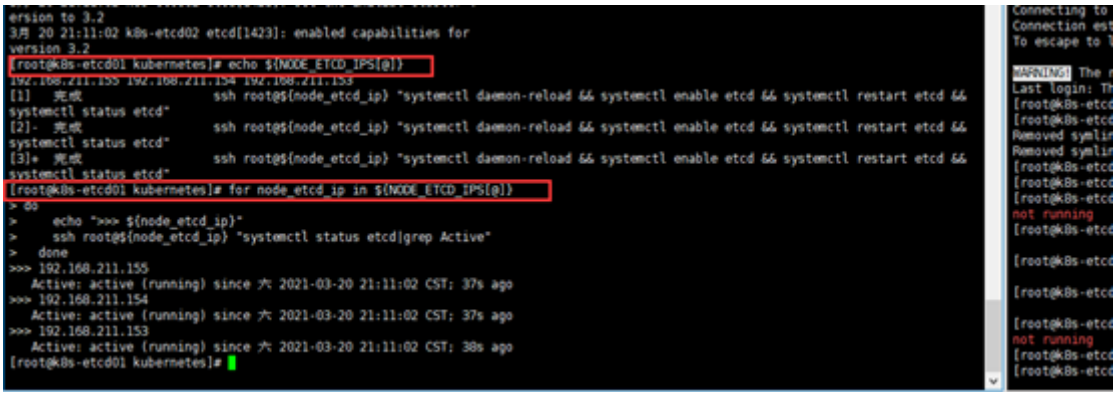
在k8s-etcd01中：

启动 etcd 服务

```
[root@k8s-etcd01 ~]# source /etc/kubernetes/environment.sh

[root@k8s-etcd01 ~]# for node_etcd_ip in ${NODE_ETCD_IPS[@]}

 do

  echo ">>> ${node_etcd_ip}"

  ssh root@${node_etcd_ip} "systemctl daemon-reload && systemctl enable etcd &&
systemctl restart etcd && systemctl status etcd" &

 done
```

验证：

```
echo ${NODE_ETCD_IPS[@]}
for node_etcd_ip in ${NODE_ETCD_IPS[@]}
 do
  echo ">>> ${node_etcd_ip}"
  ssh root@${node_etcd_ip} "systemctl status etcd|grep Active"
 done
```



验证ETCD服务

$source /etc/kubernetes/environment.sh ETCDCTL_API=2 etcdctl --endpoints=$ETCD_ENDPOINTS --ca-file=/etc/etcd/ssl/ca.pem --cert-file=/etc/etcd/ssl/etcd.pem --key-file=/etc/etcd/ssl/etcd-key.pem cluster-health$

查看当前etcd集群中的leader

在三台etcd节点中的任意一个节点机器上执行下面命令：

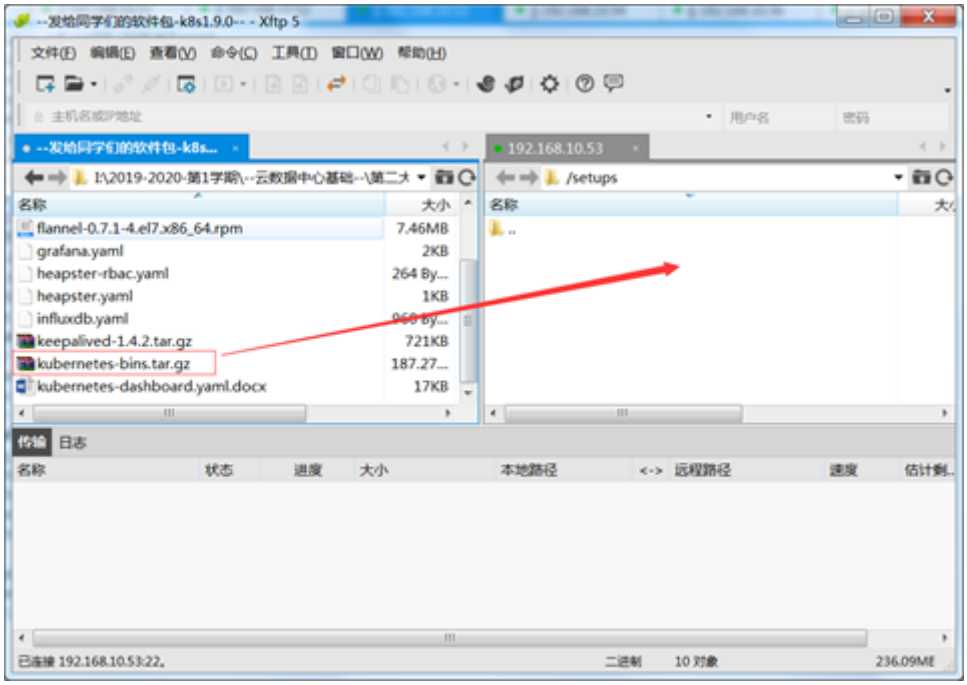
```
\# source /etc/kubernetes/environment.sh
\# ETCDCTL_API=3 /usr/local/bin/etcdctl \
 -w table --cacert=/etc/kubernetes/ssl/ca.pem \
 --cert=/etc/etcd/ssl/etcd.pem \
 --key=/etc/etcd/ssl/etcd-key.pem \
 --endpoints=${ETCD_ENDPOINTS} endpoint status
```

三、 K8s1.9生产环境高可用集群部署手册-apiserver

3.1、安装 kubernetes

我们这里先直接下载已编译好的包，然后上传到master进行安装：

```
mkdir -p /setups
cd /setups/
```



解压：

```
tar -xzvf kubernetes-bins.tar.gz
```

将二进制文件拷贝到所有 master 节点：

```
[root@k8s-master01 ~]# cd /setups/kubernetes-bins/
[root@k8s-master01 kubernetes-bins]# source /etc/kubernetes/environment.sh
[root@k8s-master01 kubernetes-bins]# for node_master_ip in ${NODE_MASTER_IPS[@]}
 do
  echo ">>> ${node_master_ip}"
  scp /setups/kubernetes-bins/{kube-apiserver,kube-controller-manager,kube-
proxy,kube-scheduler,kubectl,kubelet} root@${node_master_ip}:/usr/bin/
  ssh root@${node_master_ip} "chmod +x /usr/bin/*"
 done
[root@k8s-master01 ~]# ls /usr/bin/kube*

/usr/bin/kube-apiserver /usr/bin/kube-controller-manager /usr/bin/kubectl
/usr/bin/kube-scheduler
```

安装文件配置

查看kube-apiserver版本

```
[root@k8s-master01 ~]# /usr/bin/kube-apiserver --version Kubernetes v1.9.0
```

证书配置

记得我们在ETCD**创建证书的服务器吗，服务名为：**k8s-etcd01。为了方便，我们继续使用k8s-etcd01这一台服务器来创建证书

创建 kubernetes 证书

在k8s-etcd01服务器中，我们进入到证书创建目录：

```
[root@k8s-etcd01 key]# cd /home/key
[root@k8s-etcd01 key]# ls
ca-config.json ca-csr.json ca.pem  etcd-csr.json etcd.pem
ca.csr     ca-key.pem  etcd.csr etcd-key.pem
```

创建kubernetes**证书配置文件**

在k8s-etcd01服务器中

```
[root@k8s-etcd01 key]# cd /home/key
[root@k8s-etcd01 key]*#* cat > *kubernetes-csr.json* <<EOF
{
  "CN": "kubernetes",
  "hosts": [
   "127.0.0.1",
   "192.168.10.53",
   "192.168.10.54",
   "192.168.10.55",
   "10.254.0.1",
   "kubernetes",
   "kubernetes.default",
   "kubernetes.default.svc",
   "kubernetes.default.svc.cluster",
   "kubernetes.default.svc.cluster.local"
  ],
  "key": {
  "algo": "rsa",
  "size": 2048
  },
  "names": [
  {
   "C": "CN",
   "ST": "Chengdu",
    "L": "Chengdu",
    "O": "k8s",
    "OU": "System"
  }
  ]
}
EOF
```

解释说明：

• hosts 字段指定授权使用该证书的 IP 或域名列表，这里列出了 VIP 、apiserver 节点 IP、kubernetes 服务 IP 和域名；

• 域名最后字符不能是 .(如不能为 kubernetes.default.svc.cluster.local.)，否则解析时失败，提示：

x509: cannot parse dnsName "kubernetes.default.svc.cluster.local.";

• 如果使用非 cluster.local 域名，如 opsnull.com，则需要修改域名列表中的最后两个域名为：kubernetes.default.svc.opsnull、kubernetes.default.svc.opsnull.com

• kubernetes 服务 IP 是 apiserver 自动创建的，一般是 --service-cluster-ip-range 参数指定的网段的第一个IP，后续可以通过如下命令获取：

```
ls
```

生成 kubernetes 证书和私钥

在k8s-etcd01服务器中

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

检查生成证书

在k8s-etcd01服务器中

```
[root@k8s-etcd01 key]# ls kubernetes
kubernetes.csr kubernetes-csr.json kubernetes-key.pem kubernetes.pem
```

证书校验

在k8s-etcd01服务器中

```
[root@k8s-etcd01 key]# openssl x509 -noout -text -in kubernetes.pem
```

确认 Issuer 字段的内容和 ca-csr.json 一致；
确认 Subject 字段的内容和 kubernetes-csr.json 一致；
确认 X509v3 Subject Alternative Name 字段的内容和 kubernetes-csr.json 一致；
确认 X509v3 Key Usage、Extended Key Usage 字段的内容和 ca-config.json 中 kubernetes profile 一致；

证书分发

将生成的证书复制到Kubernetes的配置目录/etc/kubernetes/ssl/

在k8s-etcd01服务器中：

从k8s-etcd01**复制证书到**k8s-master01**的**ssl**目录下：**

需要先在服务器中创建目录mkdir -p /etc/kubernetes/ssl/，我们用for语句批量创建

```
[root@k8s-etcd01 key]# cd /home/key
[root@k8s-etcd01 key]# source /etc/kubernetes/environment.sh
[root@k8s-etcd01 key]# for node_master_ip in ${NODE_MASTER_IPS[@]}
 do
  echo ">>> ${node_master_ip}"
  ssh root@${node_master_ip} "mkdir -p /etc/kubernetes/ssl"
 done
[root@k8s-etcd01 key]# cd /home/key
[root@k8s-etcd01 key]# scp etcd*.pem ca*.pem kubernetes*.pem
root@192.168.10.53:/etc/kubernetes/ssl
```

至于k8s-master02**，我们后面一起**SCP即可。

创建加密配置文件

在k8s-master01执行

```
[root@k8s-master01 kubernetes]# echo
${ENCRYPTION_KEY}lKeucWYeur+mvZOQaTljiJxHoR4YqEztldppLwKsChs=
# cd /etc/kubernetes/
# source /etc/kubernetes/environment.sh
```

```
# cat > encryption-config.yaml <<EOF
kind: EncryptionConfig
apiVersion: v1
resources:
   - resources:
   - secrets
  providers:
   - aescbc:
  keys:
   - name: key1
   secret: ${ENCRYPTION_KEY}
   - identity: {}
EOF
```

配置文件

审核配置文件

在k8s-master01执行

审计日志可以记录所有对 apiserver 接口的调用，让我们能够非常清晰的知道集群到底发生了什么事情，通过记录的日志可以查到所发生的事件、操作的用户和时间。kubernetes 在 v1.7 中支持了日志审计功能（Alpha），在 v1.8 中为 Beta 版本，v1.12 为 GA 版本。

创建审核配置文件。

```
# cd /etc/kubernetes
# source /etc/kubernetes/environment.sh
```

```
cat >> audit-policy.yaml <<EOF
apiVersion: audit.k8s.io/v1beta1
kind: Policy
rules:
 # The following requests were manually identified as high-volume and low-risk,
so drop them.
  - level: None
  resources:
    - group: ""
  resources:
    - endpoints
    - services
    - services/status
  users:
    - system:kube-proxy
  verbs:
   - watch
   - level: None
  resources:
   - group: ""
  resources:
   - nodes
   - nodes/status
  userGroups:
   - 'system:nodes'
  verbs:
   - get
   - level: None
```

```
  namespaces:
   - kube-system
  resources:
   - group: ""
  resources:
   - endpoints
  users:
   - 'system:kube-controller-manager'
   - 'system:kube-scheduler'
   - 'system:serviceaccount:kube-system:endpoint-controller'
  verbs:
   - get
   - update
   - level: None
  resources:
   - group: ""
  resources:
   - namespaces
   - namespaces/status
   - namespaces/finalize
  users:
   - 'system:apiserver'
  verbs:
   - get
  # Don't log HPA fetching metrics.
  - level: None
  resources:
    - group: metrics.k8s.io
  users:
    - 'system:kube-controller-manager'
  verbs:
    - get
    - list
  # Don't log these read-only URLs.
    - level: None
 nonResourceURLs:
    - '/healthz'
    - /version
    - '/swagger'
# Don't log events requests.
    - level: None
 resources:
    - group: ""
 resources:
    - events
# node and pod status calls from nodes are high-volume and can be large, don't
log responses for expected updates from nodes
    - level: Request
omitStages:
    - RequestReceived
resources:
    - group: ""
    resources:
     - nodes/status
     - pods/status
      users:
      kubelet
     - 'system:node-problem-detector'
```

```
      - 'system:serviceaccount:kube-system:node-problem-detector'
       verbs:
      - update
      - patch
      - level: Request
      omitStages:
      - RequestReceived
      resources:
      - group: ""
      resources:
      - nodes/status
      - pods/status
      userGroups:
      - 'system:nodes'
      verbs:
      - update
      - patch
 # deletecollection calls can be large, don't log responses for expected
namespace deletions
      - level: Request
      omitStages:
      - RequestReceived
      users:
      - 'system:serviceaccount:kube-system:namespace-controller'
      verbs:
      - deletecollection
 # Secrets, ConfigMaps, and TokenReviews can contain sensitive & binary data,
 # so only log at the Metadata level.
      - level: Metadata
      omitStages:
      - RequestReceived
      resources:
      - group: ""
      resources:
      - secrets
      - configmaps
      - group: authentication.k8s.io
      resources:
      - tokenreviews
    # Get repsonses can be large; skip them.
    - level: Request
    omitStages:
    - RequestReceived
    resources:
    - group: ""
    - group: admissionregistration.k8s.io
    - group: apiextensions.k8s.io
    - group: apiregistration.k8s.io
    - group: apps
    - group: authentication.k8s.io
    - group: authorization.k8s.io
    - group: autoscaling
    - group: batch
    - group: certificates.k8s.io
    - group: extensions
    - group: metrics.k8s.io
    - group: networking.k8s.io
    - group: policy
```

```
    - group: rbac.authorization.k8s.io
    - group: scheduling.k8s.io
    - group: settings.k8s.io
    - group: storage.k8s.io
    verbs:
    - get
    - list
    - watch
    # Default level for known APIs
    - level: RequestResponse
    omitStages:
    - RequestReceived
    resources:
    - group: ""
    - group: admissionregistration.k8s.io
    - group: apiextensions.k8s.io
    - group: apiregistration.k8s.io
    - group: apps
    - group: authentication.k8s.io
    - group: authorization.k8s.io
    - group: autoscaling
    - group: batch
    - group: certificates.k8s.io
    - group: extensions
    - group: metrics.k8s.io
    - group: networking.k8s.io
    - group: policy
    - group: rbac.authorization.k8s.io
    - group: scheduling.k8s.io
    - group: settings.k8s.io
    - group: storage.k8s.io
     # Default level for all other requests.
     - level: Metadata
      omitStages:
    - RequestReceived
    EOF
```

将创建的配置文件 audit-policy.yaml放到/etc/kubernetes目录中。

[root@k8s-master01 kubernetes]# pwd

/etc/kubernetes

[root@k8s-master01 kubernetes]# ls

audit-policy.yaml encryption-config.yaml environment.sh ssl

3.2、配置与启动kube-apiserver

创建kube-apiserver apiserver**配置文件**

```
[root@k8s-master01 ~]#
cat > /etc/kubernetes/apiserver <<EOF
###
## kubernetes system config
##
## The following values are used to configure the kube-apiserver
##
```

```
#
## The address on the local server to listen to.
KUBE_API_ADDRESS="--advertise-address=192.168.10.53 --bind-address=0.0.0.0 --
insecure-bind-address=0.0.0.0"
#
## The port on the local server to listen on.
KUBE_API_PORT="--port=8080 --secure-port=6443"
#
## Port minions listen on
#KUBELET_PORT="--kubelet-port=10250"
#
## Comma separated list of nodes in the etcd cluster
KUBE_ETCD_SERVERS="--etcd-
servers=https://192.168.10.50:2379,192.168.10.51:2379,192.168.10.52:2379"
#
## Address range to use for services
KUBE_SERVICE_ADDRESSES="--service-cluster-ip-range=10.254.0.0/16"
#
## default admission control policies
KUBE_ADMISSION_CONTROL="--admission-
control=NamespaceLifecycle,LimitRanger,ServiceAccount,DefaultStorageClass,Resour
ceQuota,NodeRestriction"
#
## Add your own!
KUBE_API_ARGS="--enable-aggregator-routing=true --apiserver-count=2 --
authorization-mode=RBAC,Node --experimental-encryption-provider-
config=/etc/kubernetes/encryption-config.yaml --runtime-
config=rbac.authorization.k8s.io/v1beta1 --kubelet-https=true --token-auth-
file=/etc/kubernetes/token.csv --service-node-port-range=30000-32767 --tls-cert-
file=/etc/kubernetes/ssl/kubernetes.pem --tls-private-key-
file=/etc/kubernetes/ssl/kubernetes-key.pem --client-ca-
file=/etc/kubernetes/ssl/ca.pem --service-account-key-
file=/etc/kubernetes/ssl/ca-key.pem --etcd-cafile=/etc/kubernetes/ssl/ca.pem --
etcd-certfile=/etc/kubernetes/ssl/etcd.pem --etcd-
keyfile=/etc/kubernetes/ssl/etcd-key.pem --enable-swagger-ui=true --audit-log-
maxage=30 --audit-log-maxbackup=3 --audit-log-maxsize=100 --audit-log-
path=/var/lib/audit.log --event-ttl=1h"
KUBE_AUDIT="--audit-policy-file=/etc/kubernetes/audit-policy.yaml --audit-log-
path=/var/log/kubernetes/audit.log --audit-log-maxage=30 --audit-log-maxbackup=3
--audit-log-maxsize=100"
EOF
```

创建apiserver systemd unit**文件**

创建kube-apiserver systemd unit文件kube-apiserver.service

```
[root@k8s-master01 ~]# （内容有变量不用cat创建）
vi /usr/lib/systemd/system/kube-apiserver.service
[Unit]
Description=Kubernetes API Service
Documentation=https:*//github.com/GoogleCloudPlatform/kubernetes*
After=network.target
After=etcd.service
[Service]
WorkingDirectory=/etc/kubernetes/kube-apiserver
EnvironmentFile=-/etc/kubernetes/config
EnvironmentFile=-/etc/kubernetes/apiserver
```

```
ExecStart=/usr/bin/kube-apiserver \
    $KUBE_LOGTOSTDERR \
    $KUBE_AUDIT \
    $KUBE_LOG_LEVEL \
    $KUBE_ETCD_SERVERS \
    $KUBE_API_ADDRESS \
    $KUBE_API_PORT \
    $KUBELET_PORT \
    $KUBE_ALLOW_PRIV \
    $KUBE_SERVICE_ADDRESSES \
    $KUBE_ADMISSION_CONTROL \
    $KUBE_API_ARGS
Restart=on-failure
RestartSec=5
Type=notify
LimitNOFILE=65536
[Install]
WantedBy=multi-user.target
```

创建kube-apiserver config配置文件

```
[root@k8s-master01 ~]#
cat > /etc/kubernetes/config <<EOF
###
# kubernetes system config
#
# The following values are used to configure various aspects of all
# kubernetes services, including
#
#  kube-apiserver.service
#  kube-controller-manager.service
#  kube-scheduler.service
#  kubelet.service
#  kube-proxy.service
# logging to stderr means we get it in the systemd journal
KUBE_LOGTOSTDERR="--logtostderr=true"
# journal message level, 0 is debug
KUBE_LOG_LEVEL="--v=0"
# Should this cluster be allowed to run privileged docker containers
KUBE_ALLOW_PRIV="--allow-privileged=true"
# How the controller-manager, scheduler, and proxy find the apiserver
KUBE_MASTER="--master=http://127.0.0.1:8080"
EOF
```

config 文件中的配置会在kube-apiserver.service，kube-controller-manager.service，kube-scheduler.service，kubelet.service，kube-proxy.service文件件调用。
KUBE_MASTER 这里配置的是http类型访问。因为这里主要是本服务器中controller-manager,scheduler和proxy使用。

注意：

KUBE_MASTER="--master=http://127.0.0.1:8080"不能配置成192.168.10.55

因为：

打开防火墙端口

我们测试环境确保防火墙关闭即可。

在每台机器上关闭防火墙，清理防火墙规则，设置默认转发策略：

```
systemctl stop firewalld
systemctl disable firewalld
iptables -F && iptables -X && iptables -F -t nat && iptables -X -t nat
iptables -P FORWARD ACCEPT
firewall-cmd --state
```

启动apiserver服务

注意：启动服务前必须先创建工作目录；

```
[root@k8s-master01 ~]# source /etc/kubernetes/environment.sh
[root@k8s-master01 ~]# for node_master_ip in ${NODE_MASTER_IPS[@]}
 do
  echo ">>> ${node_master_ip}"
  ssh root@${node_master_ip} "mkdir -p /etc/kubernetes/kube-apiserver"
 done
systemctl daemon-reload && systemctl enable kube-apiserver && systemctl restart
kube-apiserver && systemctl status kube-apiserver
```

启动apiserver服务

在k8s-etcd01执行：

1）打印*kube-apiserver* **写入 etcd 的数据：**

```
# source /etc/kubernetes/environment.sh
# ETCDCTL_API=3 etcdctl \
  --endpoints=${ETCD_ENDPOINTS} \
  --cacert=/etc/etcd/ssl/ca.pem \
  --cert=/etc/etcd/ssl/etcd.pem \
  --key=/etc/etcd/ssl/etcd-key.pem \
  get /registry/` `--prefix --keys-only
```

`预期会打印出很多写入到etcd中的数据信息`。（如果没有打印信息，只要不报错，忽略即可）

2）授予 **kube-apiserver 访问 kubelet API 的权限**

在k8s-master01执行：

在执行 kubectl exec、run、logs 等命令时，apiserver 会将请求转发到 kubelet 的 https 端口。

这里定义 RBAC 规则，授权 apiserver 使用的证书（kubernetes.pem）用户名（CN：kuberntes）访问 kubelet API 的权限：

```
[root@k8s-master01 ~]*#* kubectl create clusterrolebinding kube-
apiserver:kubelet-apis --clusterrole=system:kubelet-api-admin --user kubernetes
```

apiserver的访问地址可以通过"kubectl cluster-info"获取。

通过curl访问API接口。

```
[root@k8s-master01 ~]*#* *curl -L --cacert /etc/kubernetes/ssl/ca.pem
https://192.168.10.53:6443/api*
{
 "kind": "APIVersions",
 "versions": [
  "v1"
 ],
 "serverAddressByClientCIDRs": [
  {
   "clientCIDR": "0.0.0.0/0",
   "serverAddress": "192.168.10.53:6443"
  }
 ]
}
```

如果这里不带证书访问，可能会提示匿名用户被禁止访问。

You can't use 'macro parameter character #' in math mode

3）检查 kube-apiserver 监听的端口

在k8s-master01执行：

```
[root@k8s-master01 kubernetes]*#* netstat -lnpt|grep kube
```

```
[root@k8s-master01 kubernetes]# netstat -lnpt|grep kube
tcp        0      0 127.0.0.1:8080          0.0.0.0:*               LISTEN      4932/kube-apiserver
tcp6       0      0 :::6443                 :::*                    LISTEN      4932/kube-apiserver
```

需要注意：

6443: 接收 https 请求的安全端口，对所有请求做认证和授权；

如果关闭了非安全端口，就不会监听 8080；

3.3、配置kube-controller-manager

创建 kube-controller-manager 证书和私钥

在k8s-etcd01操作：

创建证书签名请求：

```
[root@k8s-etcd01 ~]# cd /home/key/
```

```
[root@k8s-etcd01 key]# ls
ca-config.json  ca-csr.json  ca.pem    etcd-csr.json  etcd.pem        kubernetes-csr.json  kubernetes.pem
ca.csr          ca-key.pem   etcd.csr  etcd-key.pem   kubernetes.csr  kubernetes-key.pem
```

```
[root@k8s-etcd01 key]#
cat > kube-controller-manager-csr.json <<EOF
{
  "CN": "system:kube-controller-manager",
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "hosts": [
   "127.0.0.1",
```

```
    "192.1668.10.53",
    "192.1668.10.54"
  ],
  "names": [
    {
      "C": "CN",
      "ST": "chengdu",
      "L": "chengdu",
      "O": "system:kube-controller-manager",
      "OU": "System"
    }
  ]
}
EOF
```

```
[root@k8s-etcd01 key]# ls
ca-config.json  ca-key.pem  etcd-csr.json  kube-controller-manager-csr.json  kubernetes-key.pem
ca.csr          ca.pem      etcd-key.pem   kubernetes.csr                    kubernetes.pem
ca-csr.json     etcd.csr    etcd.pem       kubernetes-csr.json
[root@k8s-etcd01 key]#
```

• hosts 列表包含所有 kube-controller-manager 节点 IP；

• CN 为 system:kube-controller-manager、O 为 system:kube-controller-manager，kubernetes 内置的 ClusterRoleBindings system:kube-controller-manager

赋予 kube-controller-manager 工作所需的权限。

在k8s-etcd01操作：

生成证书和私钥：

```
[root@k8s-etcd01 key]# cd /home/key/
[root@k8s-etcd01 key]# cfssl gencert -ca=/home/key/ca.pem \
 -ca-key=/home/key/ca-key.pem \
 -config=/home/key/ca-config.json \
 -profile=kubernetes kube-controller-manager-csr.json | cfssljson -bare kube-controller-manager
[root@k8s-etcd01 key]# ll kube-controller-manager*pem
-rw------- 1 root root 1679 Jun 18 11:43 kube-controller-manager-key.pem
-rw-r--r-- 1 root root 1517 Jun 18 11:43 kube-controller-manager.pem
```

在k8s-etcd01操作：

分发证书

将生成的证书kube-controller-manager.pem**和私钥**kube-controller-manager-key.pem**分发到master01**的ssl目录下：

$$cd/home/key/scpkube-controller-manager.pemkube-controller-manager-key.pemroot@192.168.10.53:/etc/kubernetes/ssl/$$

去k8s-master01查看下：

```
[root@k8s-master01 kubernetes]# pwd
/etc/kubernetes
[root@k8s-master01 kubernetes]# ls
apiserver          encryption-config.yaml  kube-controller-manager.kubeconfig
audit-policy.yaml  environment.sh          ssl
config             kube-apiserver          token.csv
```

创建和分发 **kubeconfig 文件**

在k8s-master01操作：

kube-controller-manager 使用 kubeconfig 文件访问 apiserver，该文件提供了 apiserver 地址、嵌入的 CA 证书和 kube-controller-manager 证书：

```
[root@k8s-master01 ~]# cd /etc/kubernetes/
[root@k8s-master01 kubernetes]# source /etc/kubernetes/environment.sh
[root@k8s-master01 kubernetes]# echo ${KUBE_APISERVER}
[root@k8s-master01 kubernetes]# kubectl config set-cluster kubernetes
 --certificate-authority=/etc/kubernetes/ssl/ca.pem
 --embed-certs=true
 --server=${KUBE_APISERVER}
 --kubeconfig=kube-controller-manager.kubeconfig
[root@k8s-master01 kubernetes]# kubectl config set-credentials system:kube-
controller-manager
 --client-certificate=/etc/kubernetes/ssl/kube-controller-manager.pem
 --client-key=/etc/kubernetes/ssl/kube-controller-manager-key.pem
 --embed-certs=true
 --kubeconfig=kube-controller-manager.kubeconfig
[root@k8s-master01 kubernetes]# kubectl config set-context system:kube-
controller-manager
 --cluster=kubernetes
 --user=system:kube-controller-manager
 --kubeconfig=kube-controller-manager.kubeconfig
```

设置默认上下文关联：

[root@k8s-master01 kubernetes]# kubectl config use-context system:kube-controller-manager --kubeconfig=kube-controller-manager.kubeconfig

查看生成的kube-controller-manager.kubeconfig文件：

```
[root@k8s-master01 kubernetes]# pwd
/etc/kubernetes
[root@k8s-master01 kubernetes]# ls
apiserver          encryption-config.yaml  kube-controller-manager.kubeconfig
audit-policy.yaml  environment.sh          ssl
config             kube-apiserver          token.csv
```

创建配置文件/etc/kubernetes/controller-manager

在k8s-master01操作：

```
[root@k8s-master01 ~]#
cat > /etc/kubernetes/controller-manager <<EOF
###
# The following values are used to configure the kubernetes controller-manager
# defaults from config and apiserver should be adequate
# Add your own!
KUBE_CONTROLLER_MANAGER_ARGS="--address=127.0.0.1 --service-cluster-ip-
range=10.254.0.0/16 --cluster-name=kubernetes **--use-service-account-
credentials=true** --cluster-signing-cert-file=/etc/kubernetes/ssl/ca.pem --
cluster-signing-key-file=/etc/kubernetes/ssl/ca-key.pem --service-account-
private-key-file=/etc/kubernetes/ssl/ca-key.pem --root-ca-
file=/etc/kubernetes/ssl/ca.pem --leader-elect=true"
EOF
```

需要在 kube-controller-manager 的启动参数中添加 --use-service-account-credentials=true 参数，这样 main controller 会为各 controller 创建对应的 ServiceAccount XXX-controller。
内置的 ClusterRoleBinding system:controller:XXX 将赋予各 XXX-controller ServiceAccount 对应的 ClusterRole system:controller:XXX 权限。

07.04. 创建 kube-controller-manager的serivce配置文件

在k8s-master01操作：

创建配置文件/usr/lib/systemd/system/kube-controller-manager.service

```
[root@k8s-master01 ~]*#* *（有变量不用**cat**）*

vi /usr/lib/systemd/system/kube-controller-manager.service
Description=Kubernetes Controller Manager
Documentation=https://github.com/GoogleCloudPlatform/kubernetes
[Service]
WorkingDirectory=/etc/kubernetes/kube-controller-manager
EnvironmentFile=-/etc/kubernetes/config
EnvironmentFile=-/etc/kubernetes/controller-manager
ExecStart=/usr/bin/kube-controller-manager
•    $KUBE_LOGTOSTDERR
•    $KUBE_LOG_LEVEL
•    $KUBE_MASTER
•    $KUBE_CONTROLLER_MANAGER_ARGS
Restart=on-failure
LimitNOFILE=65536
[Install]
WantedBy=multi-user.target
```

kube-controller-manager

在k8s-master01操作：

```
注意：启动服务前必须先创建工作目录；
source /etc/kubernetes/environment.sh
for node_master_ip in ${NODE_MASTER_IPS[@]}
 do
  echo ">>> ${node_master_ip}"
  ssh root@${node_master_ip} "mkdir -p /etc/kubernetes/kube-controller-manager"
```

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

检查服务运行状态

```
[root@k8s-master01 ~]# source /etc/kubernetes/environment.sh
[root@k8s-master01 ~]# for node_master_ip in ${NODE_MASTER_IPS[@]}
 do
  echo ">>> ${node_master_ip}"
  ssh root@${node_master_ip} "systemctl status kube-controller-manager|grep
Active"
 done
```

确保状态为 active (running)，否则查看日志，确认原因（journalctl -u kube-controller-manager）

因为我们master02还没做，所以状态not be found。

```
[root@k8s-master01 kubernetes]# source /etc/kubernetes/environment.sh
[root@k8s-master01 kubernetes]# for node_master_ip in ${NODE_MASTER_IPS[@]}
>   do
>     echo ">>> ${node_master_ip}"
>     ssh root@${node_master_ip} "systemctl status kube-controller-manager|grep Active"
>   done
>>> 192.168.10.53
   Active: active (running) since Thu 2019-12-19 03:32:10 EST; 22s ago
>>> 192.168.10.54
Unit kube-controller-manager.service could not be found.
```

kube-controller-manager 监听 10252 端口，接收 https 请求：

```
[root@k8s-master01 ~]# netstat -lnpt|grep kube-controll
[root@k8s-master01 ~]# kubectl get cs
```

```
[root@k8s-master01 kubernetes]# kubectl get cs
NAME                 STATUS      MESSAGE
                                ERROR
scheduler            Unhealthy   Get http://127.0.0.1:10251/healthz: dial tcp 127.0.0.1:10251: ge
tsockopt: connection refused
controller-manager   Healthy     ok

etcd-0               Healthy     {"health": "true"}
```

```
 [root@k8s-master01 ~]# kubectl describe clusterrole system:kube-controller-
manager
```

```
[root@k8s-master01 kubernetes]# kubectl describe clusterrole system:kube-controller-manager
Name:         system:kube-controller-manager
Labels:       kubernetes.io/bootstrapping=rbac-defaults
Annotations:  rbac.authorization.kubernetes.io/autoupdate=true
PolicyRule:
  Resources                          Non-Resource URLs  Resource Names  Verbs
  ---------                          -----------------  --------------  -----
  *.*                                []                 []              [list watch]
  endpoints                          []                 []              [create get update]
  events                             []                 []              [create patch update]
  namespaces                         []                 []              [get]
  secrets                            []                 []              [create delete get updat
e]
  serviceaccounts                    []                 []              [create get update]
  tokenreviews.authentication.k8s.io  []                 []              [create]
[root@k8s-master01 kubernetes]#
```

查看kube-controller-manager集群中当前的leader：

```
[root@k8s-master01 ~]# kubectl get endpoints kube-controller-manager --
namespace=kube-system -o yaml
```

```
[root@k8s-master01 kubernetes]# kubectl get endpoints kube-controller-manager --namespace=kube-sy
stem  -o yaml
apiVersion: v1
kind: Endpoints
metadata:
  annotations:
    control-plane.alpha.kubernetes.io/leader: '{"holderIdentity":"k8s-master01","leaseDurationSec
onds":15,"acquireTime":"2019-12-19T08:32:10Z","renewTime":"2019-12-19T08:56:35Z","leaderTransitio
ns":0}'
  creationTimestamp: 2019-12-19T08:32:10Z
  name: kube-controller-manager
  namespace: kube-system
  resourceVersion: "887"
  selfLink: /api/v1/namespaces/kube-system/endpoints/kube-controller-manager
  uid: 0c92e909-223a-11ea-9cf3-000c29aae3f0
subsets: null
```

3.4、配置kube-scheduler

创建 kube-scheduler 证书和私钥

在k8s-etcd01节点：

创建证书签名请求：

```
[root@k8s-etcd01 ~]# cd /home/key/


[root@k8s-etcd01 key]# cat > kube-scheduler-csr.json <<EOF

{

  "CN": "system:kube-scheduler",

  "hosts": [

   "127.0.0.1",

   "192.168.10.53",

   "192.168.10.54"

  ],

  "key": {

    "algo": "rsa",

    "size": 2048

  },

  "names": [

   {

    "C": "CN",

    "ST": "chengdu",

    "L": "chengdu",

    "O": "system:kube-scheduler",

    "OU": "System"

   }

  ]

}

EOF
```

在k8s-etcd01节点：

生成证书和私钥：

```
[root@k8s-etcd01 ~]# cd /home/key/

[root@k8s-etcd01 key]# cfssl gencert -ca=/home/key/ca.pem \

 -ca-key=/home/key/ca-key.pem \

 -config=/home/key/ca-config.json \

 -profile=kubernetes kube-scheduler-csr.json | cfssljson -bare kube-scheduler

[root@k8s-etcd01 key]# ls kube-scheduler*pem

kube-scheduler-key.pem kube-scheduler.pem
```

在k8s-etcd01节点：

分发证书

将生成的证书kube-scheduler.pem**和私钥**kube-scheduler-key.pem **分发到 master01**的ssl目录下：

```
cd /home/key/

scp kube-scheduler.pem kube-scheduler-key.pem
root@192.168.10.53:/etc/kubernetes/ssl/
```

创建和分发 kubeconfig 文件

kube-scheduler 使用 kubeconfig 文件访问 apiserver，该文件提供了 apiserver 地址、嵌入的 CA 证书和 kube-scheduler 证书：

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

[root@k8s-master01 kubernetes]# source /etc/kubernetes/environment.sh



[root@k8s-master01 kubernetes]# echo ${KUBE_APISERVER}

[root@k8s-master01 kubernetes]# kubectl config set-cluster kubernetes \

 --certificate-authority=/etc/kubernetes/ssl/ca.pem \

 --embed-certs=true \

 --server=${KUBE_APISERVER} \

 --kubeconfig=kube-scheduler.kubeconfig



[root@k8s-master01 kubernetes]# kubectl config set-credentials system:kube-
scheduler \

 --client-certificate=/etc/kubernetes/ssl/kube-scheduler.pem \
```

```
 --client-key=/etc/kubernetes/ssl/kube-scheduler-key.pem \

 --embed-certs=true \

 --kubeconfig=kube-scheduler.kubeconfig



[root@k8s-master01 kubernetes]# kubectl config set-context system:kube-scheduler
\

 --cluster=kubernetes \

 --user=system:kube-scheduler \

 --kubeconfig=kube-scheduler.kubeconfig
```

关联 上下文:

```
[root@k8s-master01 kubernetes]# kubectl config use-context system:kube-scheduler
--kubeconfig=kube-scheduler.kubeconfig
```

创建scheduler**文件**

创建配置文件/etc/kubernetes/scheduler

```
[root@k8s-master01 ~]#
cat > /etc/kubernetes/scheduler <<EOF
\###
\# kubernetes scheduler config
\# default config should be adequate
\# Add your own!
KUBE_SCHEDULER_ARGS="--leader-elect=true --address=127.0.0.1 --leader-
elect=true"
EOF
```

创建kube-scheduler systemd文件

创建配置文件/usr/lib/systemd/system/kube-scheduler.service

```
[root@k8s-master01 ~]# vi /usr/lib/systemd/system/kube-scheduler.service*

[Unit]
Description=Kubernetes Scheduler Plugin
Documentation=https://github.com/GoogleCloudPlatform/kubernetes

[Service]
WorkingDirectory=/etc/kubernetes/kube-scheduler
EnvironmentFile=-/etc/kubernetes/config
EnvironmentFile=-/etc/kubernetes/scheduler
ExecStart=/usr/bin/kube-scheduler \

•       $KUBE_LOGTOSTDERR \

•       $KUBE_LOG_LEVEL \
```

```
•       $KUBE_MASTER \
•       $KUBE_SCHEDULER_ARGS
Restart=on-failure
LimitNOFILE=65536
[Install]
WantedBy=multi-user.target
```

启动scheduler

在k8s-master01节点：``

```
注意：启动服务前必须先创建工作目录；
```

```
[root@k8s-master01 ~]# source /etc/kubernetes/environment.sh
[root@k8s-master01 ~]# for node_master_ip in ${NODE_MASTER_IPS[@]}
 do
  echo ">>> ${node_master_ip}"
  ssh root@${node_master_ip} "mkdir -p /etc/kubernetes/kube-scheduler"
 done
systemctl daemon-reload
systemctl enable kube-scheduler
systemctl start kube-scheduler
systemctl status kube-scheduler
```

检查服务运行状态：

```
[root@k8s-master01 ~]# source /etc/kubernetes/environment.sh
[root@k8s-master01 ~]# for node_master_ip in ${NODE_MASTER_IPS[@]}
 do
  echo ">>> ${node_master_ip}"
  ssh root@${node_master_ip} "systemctl status kube-scheduler|grep Active"
 done
[root@k8s-master01 ~]# kubectl get cs
```

查看当前的 leader：

```
[root@k8s-master01 ~]# kubectl get endpoints kube-scheduler --namespace=kube-system -o yaml
```

3.5、配置kubectl管理工具

创建 admin 证书

在k8s-etcd01上：

kubectl与apiserver https安全端口通信，apiserver 对提供的证书进行认证和授权。

kubectl作为集群的管理工具，需要被授予最高权限，这里创建具有最高权限的 admin 证书。

创建证书签名请求：

```
cd /home/key/

cat > admin-csr.json <<EOF

{

 "CN": "admin",

 "hosts": [],

 "key": {

  "algo": "rsa",

  "size": 2048

 },

 "names": [

  {

   "C": "CN",

   "ST": "BeiJing",

   "L": "BeiJing",

   "O": "system:masters",

   "OU": "System"

  }

 ]

}

EOF
```

生面admin证书

在k8s-etcd01上：

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

分发证书

在k8s-etcd01上：

```
scp admin-key.pem admin.pem root@192.168.10.53:/etc/kubernetes/ssl/
```

配置工具

回到服务器k8s-master01中进行操作。

```
cd /etc/kubernetes/

echo ${KUBE_APISERVER}
```

设置**KUBE_APISERVER变量****

```
source /etc/kubernetes/environment.sh
```

设置集群参数

```
kubectl config set-cluster kubernetes \

 --certificate-authority=/etc/kubernetes/ssl/ca.pem \

 --embed-certs=true \

 --server=${KUBE_APISERVER} \

 --kubeconfig=kubectl.kubeconfig
```

设置客户端认证参数

```
kubectl config set-credentials admin \

 --client-certificate=/etc/kubernetes/ssl/admin.pem \

 --embed-certs=true \

 --client-key=/etc/kubernetes/ssl/admin-key.pem \

 --kubeconfig=kubectl.kubeconfig
```

设置上下文参数

```
kubectl config set-context kubernetes \

 --cluster=kubernetes \

 --user=admin \

 --kubeconfig=kubectl.kubeconfig
```

设置默认上下文

```
kubectl config use-context kubernetes --kubeconfig=kubectl.kubeconfig
```

```
cd /etc/kubernetes/
source /etc/kubernetes/environment.sh
for node_master_ip in ${NODE_MASTER_IPS[@]}
do
 echo ">>> ${node_master_ip}"
 ssh root@${node_master_ip} "mkdir -p /etc/kubernetes/"
 scp kubectl.kubeconfig root@${node_master_ip}:/etc/kubernetes/
done
```

检查集群信息

在服务器k8s-master01中：

```
[root@k8s-master01 ~]*#* *kubectl get componentstatuses*
```

3.6、配置k8s-master02

下面的复制操作都在服务器k8s-master01中，有注明，请注意看：

```
[root@k8s-master01 kubernetes]# pwd

/etc/kubernetes
```

进到/kubernetes目录下。

复制证书到/etc/kubernetes/ssl**中**

k8s-master01上操作：

```
[root@k8s-master01 ~]# scp -r /etc/kubernetes/ssl
root@192.168.10.54:/etc/kubernetes/

root@192.168.10.54*'s password:*

ca.pem                        100% 1359   2.9MB/s   00:00

ca-key.pem                      100% 1679   2.7MB/s   00:00

kubernetes.pem                    100% 1627   2.9MB/s   00:00

kubernetes-key.pem                  100% 1679   3.5MB/s   00:00
```

```
etcd.pem                      100% 1436   3.1MB/s   00:00
etcd-key.pem                  100% 1679   3.9MB/s   00:00
admin-key.pem                 100% 1675  83.4KB/s   00:00
admin.pem                     100% 1399  185.6KB/s   00:00
```

复制加密配置文件encryption-config.yaml到** **master02 节点的 /etc/kubernetes 目录下**:

```
k8s-master01上操作：
cd /etc/kubernetes/
source /etc/kubernetes/environment.sh
for node_master_ip in ${NODE_MASTER_IPS[@]}
 do
  echo ">>> ${node_master_ip}"
  scp encryption-config.yaml root@${node_master_ip}:/etc/kubernetes/
 done
```

复制分发 kube-controller-manager.kubeconfig到所有** **master 节点**:

k8s-master01上操作:

```
[root@k8s-master01 kubernetes]# cd /etc/kubernetes/
[root@k8s-master01 kubernetes]# source /etc/kubernetes/environment.sh
[root@k8s-master01 kubernetes]# for node_master_ip in ${NODE_MASTER_IPS[@]}
 do
  echo ">>> ${node_master_ip}"
  scp kube-controller-manager.kubeconfig root@${node_master_ip}:/etc/kubernetes/
 done
```

复制分发kube-scheduler.kubeconfig** **到所有 master 节点**:

k8s-master01上操作:

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

[root@k8s-master01 kubernetes]# source /etc/kubernetes/environment.sh

[root@k8s-master01 kubernetes]# for node_master_ip in ${NODE_MASTER_IPS[@]}

 do

  echo ">>> ${node_master_ip}"

  scp kube-scheduler.kubeconfig root@${node_master_ip}:/etc/kubernetes/

 done
```

复制/etc/kubernetes/apiserver**

k8s-master01上操作：

```
[root@k8s-master01 kubernetes]*#* *scp -r* /etc/kubernetes/*apiserver
root@192.168.10.54:/etc/kubernetes/*

root@192.168.10.54's password:

apiserver                     100% 1645  122.6KB/s  00:00
```

复制完成，在**master02**主机**上需要把以下行的192.168.10.53改成192.168.10.54

```
*vi /etc/kubernetes/apiserver*

*KUBE_API_ADDRESS="*--*advertise*-*address=**192.168.10.53* --*bind*-
*address=192.168.10.53* --*insecure*-*bind*-*address=0*.*0*.*0*.*0"
```

改为

```
KUBE_API_ADDRESS="*--*advertise*-*address=192.168.10.54* --*bind*-
*address=**192.168.10.54* --*insecure*-*bind*-*address=0*.*0*.*0*.*0"*
```

注：bind地址可以不管。默认0.0.0.0。

复制/etc/kubernetes/config**

k8s-master01上操作：

```
[root@k8s-master01 kubernetes]*#* *scp /etc/kubernetes/config
root@192.168.10.54:/etc/kubernetes/*
root@192.168.10.54's password:
config                        100% 657  60.4KB/s  00:00
```

复制/etc/kubernetes/controller-manager**

k8s-master01上操作：

```
[root@k8s-master01 kubernetes]*#* *scp /etc/kubernetes/controller-manager
root@192.168.10.54:/etc/kubernetes/*
root@192.168.10.54's password:
controller-manager                100% 517   49.9KB/s   00:00
```

复制/etc/kubernetes/scheduler**

k8s-master01上操作：

```
[root@k8s-master01 kubernetes]*#* *scp /etc/kubernetes/scheduler
root@192.168.10.54:/etc/kubernetes/*

root@192.168.10.54's password:

scheduler                     100% 150   25.8KB/s   00:00
```

复制/etc/kubernetes/token.csv**

k8s-master01上操作：

```
[root@k8s-master01 kubernetes]*#* *scp /etc/kubernetes/token.csv
root@192.168.10.54:/etc/kubernetes/*

root@192.168.10.54's password:

token.csv                     100%   84    6.4KB/s   00:00
```

复制apiserver,controller-manager,scheduler systemd配置文件**

k8s-master01上操作：

```
[root@k8s-master01 system]*#* *cd /usr/lib/systemd/system/*
[root@k8s-master01 system]*#* *scp kube-\*
root@192.168.10.54:/usr/lib/systemd/system/*
root@192.168.10.54's password:
kube-apiserver.service              100% 611   57.5KB/s   00:00
kube-controller-manager.service         100% 432   53.9KB/s   00:00
kube-scheduler.service              100% 438   53.5KB/s   00:00
```

复制二进制文件

k8s-master01上操作：

```
[root@k8s-master01 ~]*#* *scp /usr/bin/kube\* root@192.168.10.54:/usr/bin/*
root@192.168.10.54's password:
kube-apiserver                  100% 200MB 12.5MB/s   00:16
kube-controller-manager             100% 130MB 10.9MB/s   00:12
kubectl                       100%   64MB 10.7MB/s   00:06
kube-scheduler                  100%   59MB 58.7MB/s   00:01
```

复制审计文件

k8s-master01上操作：

```
[root@k8s-master01 ~]*#* *scp /etc/kubernetes/audit-policy.yaml
root@192.168.10.54:/etc/kubernetes/*

root@192.168.10.54's password:

kube-apiserver                  100% 200MB 12.5MB/s   00:16

kube-controller-manager             100% 130MB 10.9MB/s   00:12

kubectl                      100%  64MB 10.7MB/s   00:06

kube-scheduler                  100%  59MB 58.7MB/s   00:01
```

k8s-master02服务器配置**

将所有的服务复制完成后，我们在k8s-master02中进行一些配置。并启动服务。

开放端口：

```
firewall-cmd --add-port=6443/tcp --permanent

firewall-cmd --reload
```

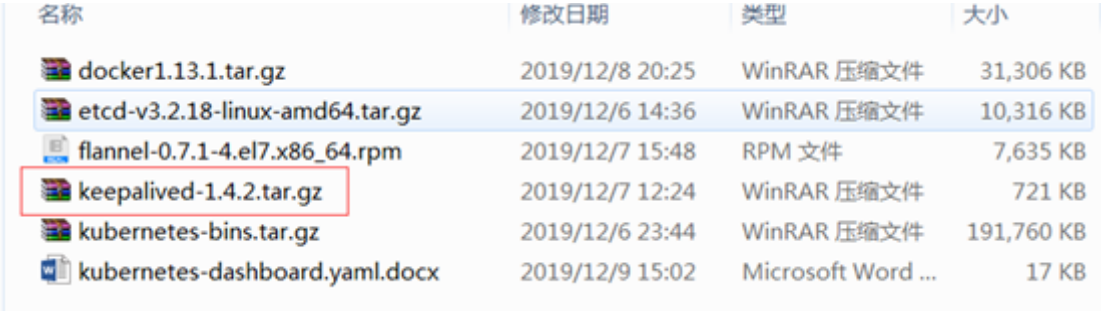
启动：

```
chmod +x /usr/bin/kube*

systemctl daemon-reload

systemctl enable kube-apiserver kube-controller-manager kube-scheduler

systemctl start kube-apiserver kube-controller-manager kube-scheduler

systemctl status kube-apiserver

systemctl status kube-controller-manager

systemctl status kube-scheduler
```

3.7、配置apiserver高可用

```
在k8s-master01和k8s-master02中都要操作：
```

我们这里安装的版本为：keepalived-1.4.2



cd keepalived-1.4.2

./configure --prefix=/usr/local/keepalived

make && make install

ln -s /usr/local/keepalived/sbin/keepalived /usr/sbin/keepalived

编辑systemd unit文件

```
[root@k8s-master01 ~]*#* *cat /usr/lib/systemd/system/keepalived.service*

[Unit]

Description=LVS and VRRP High Availability Monitor

After= network-online.target syslog.target

Wants=network-online.target

[Service]

Type=forking

PIDFile=/var/run/keepalived.pid

KillMode=process

EnvironmentFile=-/usr/local/keepalived/etc/sysconfig/keepalived

ExecStart=/usr/local/keepalived/sbin/keepalived $KEEPALIVED_OPTIONS

ExecReload=/bin/kill -HUP $MAINPID

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

k8s-master01***配置keepalived**

在k8s-master01中操作：

```
mkdir -p /etc/keepalived/

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

global_defs {

  router_id k8s-master01

  script_user root

  enable_script_security

}

vrrp_script CheckK8sMaster {

  script "/usr/bin/curl -o /dev/null -s -w %{http_code} -k
https://192.168.10.53:6443"
```

```
  interval 3
  timeout 3
  fall 2
  rise 2
}
vrrp_instance VI_1 {
  state BACKUP
  interface **ens33**
  virtual_router_id 110
  priority 120
  advert_int 1
  mcast_src_ip 192.168.10.53
  nopreempt
  authentication {
    auth_type PASS
    auth_pass ydstest
  }
  unicast_peer {
    192.168.10.54
  }
  virtual_ipaddress {
    192.168.10.55/24
  }
  track_script {
    CheckK8sMaster
  }
}
EOF
```

k8s-master02**配置keepalived****

在k8s-master02中操作：

```
mkdir -p /etc/keepalived/



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

global_defs {

  router_id k8s-master02

  script_user root

  enable_script_security

}



vrrp_script CheckK8sMaster {

  script "/usr/bin/curl -o /dev/null -s -w %{http_code} -k
https://192.168.10.54:6443"

  interval 3

  timeout 3

  fall 2

  rise 2

}



vrrp_instance VI_1 {

  state BACKUP

  interface **ens33**

  virtual_router_id 110

  priority 100

  advert_int 1

  mcast_src_ip 192.168.10.54

  nopreempt

  authentication {

•    auth_type PASS
```

```
    auth_pass ydstest
  }
  unicast_peer {
    192.168.10.53
  }
  virtual_ipaddress {
    192.168.10.55/24
  }
  track_script {
    CheckK8sMaster
  }

}
EOF
```

keepalived启动

在k8s-master01和k8s-master02中启动keepalived.

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

高可用测试

1.**查看IP信息****
k8s-master01

```
[root@k8s-master01 ~]*#*  *systemctl status kube-apiserver*
[root@k8s-master01 ~]*#*  *ip a*
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1
  link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
  inet 127.0.0.1/8 scope host lo
    valid_lft forever preferred_lft forever
  inet6 ::1/128 scope host
    valid_lft forever preferred_lft forever
```

```
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
qlen 1000

  link/ether 00:0c:29:48:d8:a8 brd ff:ff:ff:ff:ff:ff

  inet 192.168.10.53/24 brd 192.168.10.255 scope global ens33

•    valid_lft forever preferred_lft forever

  inet 192.168.10.55/24 scope global secondary ens33

•    valid_lft forever preferred_lft forever

  inet6 fe80::9cd:60a3:99e2:48ff/64 scope link tentative dadfailed

•    valid_lft forever preferred_lft forever

  inet6 fe80::fbd2:5239:fe68:ea3d/64 scope link tentative dadfailed

•    valid_lft forever preferred_lft forever

  inet6 fe80::2a36:8b76:9a1d:7d50/64 scope link tentative dadfailed

•    valid_lft forever preferred_lft forever

k8s-master02

[root@k8s-master02 ~]*#*  *systemctl status kube-apiserver*

[root@k8s-master02 ~]*#* *ip a*

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1

  link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

  inet 127.0.0.1/8 scope host lo

•    valid_lft forever preferred_lft forever

  inet6 ::1/128 scope host

•    valid_lft forever preferred_lft forever

2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
qlen 1000

  link/ether 00:0c:29:fc:62:1d brd ff:ff:ff:ff:ff:ff

  inet 192.168.10.54/24 brd 192.168.10.255 scope global ens33

•    valid_lft forever preferred_lft forever

  inet6 fe80::9cd:60a3:99e2:48ff/64 scope link tentative dadfailed

•    valid_lft forever preferred_lft forever

  inet6 fe80::fbd2:5239:fe68:ea3d/64 scope link tentative dadfailed
```

```
    valid_lft forever preferred_lft forever
  inet6 fe80::2a36:8b76:9a1d:7d50/64 scope link tentative dadfailed
    valid_lft forever preferred_lft forever
```

我们查看到，现在192.168.10.55在k8s-master01中。
我们访问一下192.168.10.55

```
[root@k8s-master02 ~]# curl -k https://192.168.10.55:6443
{
 "kind": "Status",
 "apiVersion": "v1",
 "metadata": {

 },
 "status": "Failure",
 "message": "forbidden: User \"system:anonymous\" cannot get path \"/\"",
 "reason": "Forbidden",
 "details": {

 },
 "code": 403
```

服务访问正常。

现在，我们停用k8s-master01中的kube-apiserver.

在k8s-master01中

```
[root@k8s-master01 ~]*#* *systemctl stop kube-apiserver*
[root@k8s-master01 ~]*#* *ip a*
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1
  link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
  inet 127.0.0.1/8 scope host lo
    valid_lft forever preferred_lft forever
  inet6 ::1/128 scope host
```

```
    valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
qlen 1000
  link/ether 00:0c:29:48:d8:a8 brd ff:ff:ff:ff:ff:ff
  inet 192.168.10.53/24 brd 192.168.10.255 scope global ens33
    valid_lft forever preferred_lft forever
  inet6 fe80::9cd:60a3:99e2:48ff/64 scope link tentative dadfailed
    valid_lft forever preferred_lft forever
  inet6 fe80::fbd2:5239:fe68:ea3d/64 scope link tentative dadfailed
    valid_lft forever preferred_lft forever
  inet6 fe80::2a36:8b76:9a1d:7d50/64 scope link tentative dadfailed
    valid_lft forever preferred_lft forever
```

在k8s-master02

```
[root@k8s-master02 ~]*#* *ip a*
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1
  link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
  inet 127.0.0.1/8 scope host lo
    valid_lft forever preferred_lft forever
  inet6 ::1/128 scope host
    valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
qlen 1000
  link/ether 00:0c:29:fc:62:1d brd ff:ff:ff:ff:ff:ff
  inet 192.168.10.54/24 brd 192.168.10.255 scope global ens33
    valid_lft forever preferred_lft forever
  inet 192.168.10.55/24 scope global secondary ens33
    valid_lft forever preferred_lft forever
  inet6 fe80::9cd:60a3:99e2:48ff/64 scope link tentative dadfailed
    valid_lft forever preferred_lft forever
  inet6 fe80::fbd2:5239:fe68:ea3d/64 scope link tentative dadfailed
```

```
    valid_lft forever preferred_lft forever
  inet6 fe80::2a36:8b76:9a1d:7d50/64 scope link tentative dadfailed
    valid_lft forever preferred_lft forever
```

看到IP 192.168.10.55已经切换到k8s-master02中。测试访问：

```
[root@k8s-master02 ~]# curl -k https://192.168.10.55:6443
{
 "kind": "Status",
 "apiVersion": "v1",
 "metadata": {

 },
 "status": "Failure",
 "message": "forbidden: User \"system:anonymous\" cannot get path \"/\"",
 "reason": "Forbidden",
 "details": {

 },
 "code": 403
```

服务访问正常。

3.8、检查 kube-apiserver 监听的端口

netstat -lnpt|grep kube

```
[root@k8s-master02 keepalived]# netstat -lnpt|grep kube
tcp        0      0 127.0.0.1:10251         0.0.0.0:*               LISTEN      1098/kube-scheduler
tcp        0      0 127.0.0.1:10252         0.0.0.0:*               LISTEN      1093/kube-controlle
tcp        0      0 127.0.0.1:8080          0.0.0.0:*               LISTEN      1103/kube-apiserver
tcp6       0      0 :::6443                 :::*                    LISTEN      1103/kube-apiserver
```

或

```
[root@k8s-master01 ~]# netstat -lnpt|grep kube
tcp        0      0 127.0.0.1:10251         0.0.0.0:*               LISTEN      8395/kube-scheduler
tcp        0      0 127.0.0.1:10252         0.0.0.0:*               LISTEN      8396/kube-controlle
tcp6       0      0 :::6443                 :::*                    LISTEN      8415/kube-apiserver
tcp6       0      0 :::8080                 :::*                    LISTEN      8415/kube-apiserver
```

以上，我们的apiserver高可用配置完成。

四、K8s1.9生产环境高可用集群部署手册-docker

4.1、服务器配置

k8s-node01上操作:

01.01 配置服务器名和IP

```
[root@localhost ~]*#* *hostnamectl set-hostname k8s-node01*

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

nameserver 61.139.2.69

nameserver 114.114.114.114

EOF

[root@localhost ~]*#* *cat /etc/sysconfig/network-scripts/ifcfg-ens33*

TYPE=Ethernet

PROXY_METHOD=none

BROWSER_ONLY=no

BOOTPROTO=static

DEFROUTE=yes

IPV4_FAILURE_FATAL=no

IPV6INIT=yes

IPV6_AUTOCONF=yes

IPV6_DEFROUTE=yes

IPV6_FAILURE_FATAL=no

IPV6_ADDR_GEN_MODE=stable-privacy

NAME=ens32

UUID=7d6fb2ed-364c-415f-9b02-0e54436ff1ec

DEVICE=ens32

ONBOOT=yes

IPADDR=192.168.10.56

NETMASK=255.255.255.0

GATEWAY=192.168.10.1

DNS1=192.168.10.1
```

```
DNS2=61.139.2.69
```

配置完成后，退出重新登录。

查看地址映射：

```
cat /etc/hosts
```

k8s-node02上操作：

```
[root@localhost ~]*#* *hostnamectl set-hostname k8s-node02*

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

nameserver 61.139.2.69

nameserver 114.114.114.114

EOF
[root@localhost ~]*#* *cat /etc/sysconfig/network-scripts/ifcfg-ens33*
```

配置完成后，退出重新登录。

查看地址映射：

```
cat /etc/hosts
```

k8s-node03上操作：

```
[root@localhost ~]*#* *hostnamectl set-hostname k8s-node03*

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

nameserver 61.139.2.69

nameserver 114.114.114.114

EOF

[root@localhost ~]*#* *cat /etc/sysconfig/network-scripts/ifcfg-ens33*
```

配置完成后，退出重新登录。

查看地址映射：

```
cat /etc/hosts
```

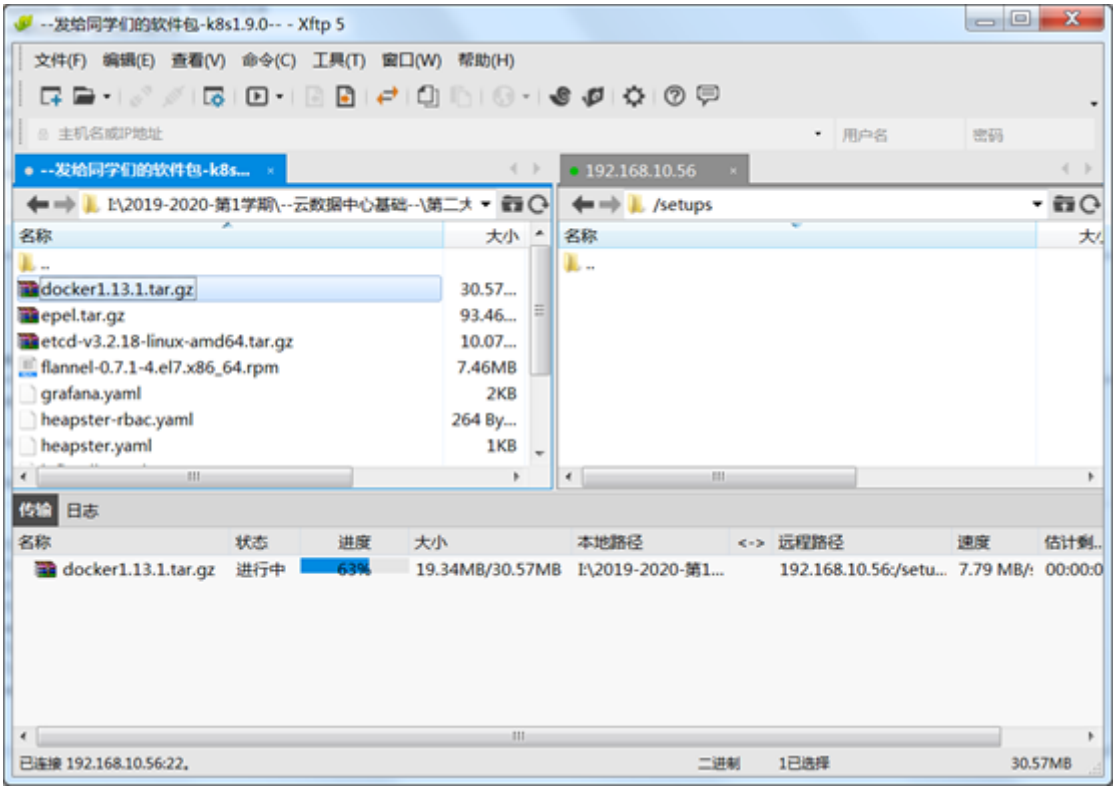
4.2、Docker安装部署

4.2.1、安装Docker

在k8s-node01上：

```
mkdir -p /setups/
```

把docker1.13.1版本离线包docker上传到/setups/packets/目录下，进入docker目录执行rpm --Uvh * 安装。




```
cd /setups/

tar -xzvf docker1.13.1.tar.gz

cd /setups/docker/

rpm -Uvh * --nodeps --force
```

从node01分发docker包到node02和node03上面，并进行安装：

```
[root@k8s-node01 setups]#

source /etc/kubernetes/environment.sh



for (( i=1; i < 3; i++ ));

  do

  echo ">>> ${NODE_NODE_IPS[i]}";

  ssh root@${NODE_NODE_IPS[i]} "mkdir -p /setups/"

  scp -r /setups/docker root@${NODE_NODE_IPS[i]}:/setups
```

```
  ssh root@${NODE_NODE_IPS[i]} "cd /setups/docker/ && rpm -Uvh * --nodeps --
force"

 done
```

分发docker包到master01上面，并进行安装：（**如果报错缺少依赖包，则使用网络源安装，参见后面的步骤。命令中我们忽略了依赖检查，因为有已安装的包时，安装就会暂停，不过如果后面有问题，再针对性的解决**）

```
for (( i=0; i < 1; i++ ));

  do

  echo ">>> ${NODE_MASTER_IPS[i]}";

  ssh root@${NODE_MASTER_IPS[i]} "yum install -y libselinux-devel"

  ssh root@${NODE_MASTER_IPS[i]} "yum install -y libsepol-devel wget lsof"

  ssh root@${NODE_MASTER_IPS[i]} "mkdir -p /setups/"

  scp -r /setups/docker root@${NODE_MASTER_IPS[i]}:/setups

  ssh root@${NODE_MASTER_IPS[i]} "cd /setups/docker/ && rpm -Uvh * --nodeps --
force"

 done

[root@k8s-node01 ~]# docker --version

Docker version 1.13.1, build 94f4240/1.13.1
```

4.2.2、启动Docker

在k8s-node01上：

```
[root@k8s-node01 ~]# source /etc/kubernetes/environment.sh

[root@k8s-node01 ~]# for node_node_ip in ${NODE_NODE_IPS[@]}

 do

  echo ">>> ${node_node_ip}"

  ssh root@${node_node_ip} "systemctl daemon-reload && systemctl enable docker
&& systemctl restart docker && systemctl status docker"

 done



for (( i=0; i < 1; i++ ));

  do
```

```
  echo ">>> ${NODE_MASTER_IPS[i]}";

  ssh root@${NODE_MASTER_IPS[i]} "systemctl daemon-reload && systemctl enable
docker && systemctl restart docker && systemctl status docker"

 done
```

4.2.3、检查Docker

在k8s-node01上:

```
[root@k8s-node01 ~]# docker info

Containers: 0

 Running: 0

 Paused: 0

 Stopped: 0

Images: 0

Server Version: 1.13.1

Storage Driver: overlay2

 Backing Filesystem: xfs

 Supports d_type: true

 Native Overlay Diff: true

Logging Driver: journald

Cgroup Driver: systemd

Plugins:

 Volume: local

 Network: bridge host macvlan null overlay

Swarm: inactive

Runtimes: docker-runc runc

Default Runtime: docker-runc

Init Binary: docker-init

containerd version: (expected: aa8187dbd3b7ad67d8e5e3a15115d3eef43a7ed1)

runc version: N/A (expected: 9df8b306d01f59d3a8029be411de015b7304dd8f)

init version: N/A (expected: 949e6facb77383876aeff8a6944dde66b3089574)
```

```
Security Options:
 seccomp
 WARNING: You're not using the default seccomp profile
 Profile: /etc/docker/seccomp.json
 selinux
Kernel Version: 3.10.0-693.21.1.el7.x86_64
Operating System: CentOS Linux 7 (Core)
OSType: linux
Architecture: x86_64
Number of Docker Hooks: 3
CPUs: 2
Total Memory: 1.78 GiB
Name: k8s-node01
ID: YKWT:7Y6M:O3FB:C7BC:KU3Q:ZI5I:KM7E:QGTW:7TZV:2WF4:S5LD:ROKB
Docker Root Dir: /var/lib/docker
Debug Mode (client): false
Debug Mode (server): false
Registry: https:*//index.docker.io/v1/*
Experimental: false
Insecure Registries:
 127.0.0.0/8
Live Restore Enabled: false
Registries: docker.io (secure)
```

现在Docker默认的Storage Driver存储驱动为overlay2，只适用于测试环境。但我们在生产环境，需要把这一项改为devicemapper的direct-lvm模式，不要在生产中使用loop-lvm模式。

接下来，我们把docker的存储改为direct-lvm。

4.3、生产环境

4.3.1、 配置Docker direct-lvm模式

硬盘检查

我们在设置里面添加一块20G的硬盘。

```
reboot
```

我们先查看一下硬盘信息。

```
[root@k8s-node01 ~]# fdisk -l
```

停止Docker

```
systemctl stop docker
```

安装软件

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

创建物理卷

```
[root@k8s-node01 ~]*#* *pvcreate /dev/sdb*

 Physical volume "/dev/sdb" successfully created.
```

创建Docker卷组

```
[root@k8s-node01 ~]# vgcreate docker /dev/sdb

 Volume group "docker" successfully created
```

创建逻辑卷

在这里需要创建二个逻辑卷，名称为thinpool和thinpoolmeta。

```
[root@k8s-node01 ~]*#* *lvcreate --wipesignatures y -n thinpool docker -l 95%VG*
 Logical volume "thinpool" created.
[root@k8s-node01 ~]*#* *lvcreate --wipesignatures y -n thinpoolmeta docker -l
1%VG*
 Logical volume "thinpoolmeta" created.
```

卷转换

Convert the volumes to a thin pool and a storage location for metadata for the thin pool, using the lvconvert command.
将刚创建的卷转换为thin pool，并使用刚创建的thinpoolmeta卷。

```
[root@k8s-node01 ~]# lvconvert -y \

*--zero n \*

-c 512K \

*--thinpool docker/thinpool \*
```

```
*--poolmetadata docker/thinpoolmeta*

 Thin pool volume with chunk size 512.00 KiB can address at most 126.50 TiB of
data.

 WARNING: Converting logical volume docker/thinpool and docker/thinpoolmeta to
thin pool's data and metadata volumes with metadata wiping.

 THIS WILL DESTROY CONTENT OF LOGICAL VOLUME (filesystem etc.)

 Converted docker/thinpool_tdata to thin pool.
```

自动扩展配置

需要配置的选项有thin_pool_autoextend_threshold和thin_pool_autoextend_percent。

thin_pool_autoextend_threshold: 当使用量达到百分之多少是会尝试进行自动扩容，使用已经存在的空间。配置为100表示为不扩容(Disable)。
thin_pool_autoextend_percent: 在扩容是增加百分之多少空间。
The example below adds 20% more capacity when the disk usage reaches 80%.
下面配置是当卷的使用量达到80%的时候增加20%的容量。

activation {

thin_pool_autoextend_threshold=80

thin_pool_autoextend_percent=20

}

现在我们把这个配置写到配置文件/etc/lvm/profile/docker-thinpool.profile中。

```
[root@k8s-node01 ~]# cat /etc/lvm/profile/docker-thinpool.profile

activation {

 thin_pool_autoextend_threshold=80

 thin_pool_autoextend_percent=20

}
```

追加以下内容到docker-thinpool.profile文件末尾：

```
cat <<EOF >> /etc/lvm/profile/docker-thinpool.profile

activation {

 thin_pool_autoextend_threshold=80

 thin_pool_autoextend_percent=20

}

EOF
```

应用**LVM配置文件****

```
[root@k8s-node01 ~]# lvchange --metadataprofile docker-thinpool docker/thinpool

 Logical volume docker/thinpool changed.
```

启用**LVM监控****

如果不启用lvm监控，刚才我们配置的自动扩容是不生效的。

```
[root@k8s-node01 ~]*#* *lvs -o+seg_monitor*
```

消除**Docker数据****

如果存在/var/lib/docker，将里面的文件备份或清空。

mkdir /var/lib/docker.bk

mv /var/lib/docker/* /var/lib/docker.bk

当配置完成后，如果不出错，就可以删除掉目录/var/lib/docker.bk

** **配置****Docker存储驱动****

如果在配置前/etc/docker/daemon.json为空。现在我们把这个文件修改成以下内容.

```
[root@k8s-node01 docker]*#* *vi /etc/docker/daemon.json*
{
  "storage-driver": "devicemapper",

  "storage-opts": [

  "dm.thinpooldev=/dev/mapper/docker-thinpool",

  "dm.use_deferred_removal=true",

  "dm.use_deferred_deletion=true"
  ]
}
```

如果在/etc/sysconfig/docker-storage在有下面配置，注释掉.

vi /etc/sysconfig/docker-storage
 #DOCKER_STORAGE_OPTIONS="--storage-driver overlay2

[root@k8s-node01 docker]# cat /etc/sysconfig/docker-storage

查看/etc/sysconfig/docker-storage-setup中信息.

```
[root@k8s-node01 docker]# sed -i
"s/^STORAGE_DRIVER=overlay2/STORAGE_DRIVER=devicemapper/g"
/etc/sysconfig/docker-storage-setup

[root@k8s-node01 docker]# cat /etc/sysconfig/docker-storage-setup

STORAGE_DRIVER=devicemapper
```

启动Docker****

```
systemctl start docker
systemctl status docker
```

验证配置

```
[root@k8s-node01 docker]# docker info
```

如果配置正确，Data file和Metadata file俩是空的，pool name 是docker-thinpool。

清理

当验证完我们以将前面我们备份的目录删除掉.

```
rm -rf /var/lib/docker.bk
```

4.3.2、Docker配置

由于默认的Base Device Size为10G，而经常Docker的大小会超过10G，需要修改这个值的大小。
我们这里把Base Device Size修改为30G。
只需要在/etc/docker/daemon.json中增加参数：dm.basesize=20G

```
[root@k8s-node01 docker]*#* *vi /etc/docker/daemon.json*

{

  "storage-driver": "devicemapper",

  "storage-opts": [

  "dm.thinpooldev=/dev/mapper/docker-thinpool",

  "dm.use_deferred_removal=true",

  "dm.use_deferred_deletion=true",

  "dm.basesize=20G"

  ]

}
```

这里注意添加"dm.basesize=20G"这行后，上面一行需要加个,号。

重启Docker然后验证如下:

```
systemctl restart docker

[root@k8s-node01 docker]# docker info
```

/etc/sysconfig/docker配置文件

```
[root@k8s-node01 ~]# cat /etc/sysconfig/docker

\# /etc/sysconfig/docker



\# Modify these options if you want to change the way the docker daemon runs

OPTIONS=''

if [ -z "${DOCKER_CERT_PATH}" ]; then

  DOCKER_CERT_PATH=/etc/docker

fi
\# Do not add registries in this file anymore. Use
/etc/containers/registries.conf

\# from the atomic-registries package.

\#



\# On an SELinux system, if you remove the --selinux-enabled option, you

\# also need to turn on the docker_transition_unconfined boolean.

\# setsebool -P docker_transition_unconfined 1



\# Location used for temporary files, such as those created by

\# docker load and build operations. Default is /var/lib/docker/home

\# Can be overriden by setting the following environment variable.

\# DOCKER_HOMEDIR=/var/home



\# Controls the /etc/cron.daily/docker-logrotate cron job status.

\# To disable, uncomment the line below.

\# LOGROTATE=false
```

```
\# docker-latest daemon can be used by starting the docker-latest unitfile.

\# To use docker-latest client, uncomment below lines

\#DOCKERBINARY=/usr/bin/docker-latest

\#DOCKERDBINARY=/usr/bin/dockerd-latest

\#DOCKER_CONTAINERD_BINARY=/usr/bin/docker-containerd-latest

\#DOCKER_CONTAINERD_SHIM_BINARY=/usr/bin/docker-containerd-shim-latest
```

docker-storage-setup 配置文件

```
[root@k8s-node01 ~]# cat /etc/sysconfig/docker-storage-setup

STORAGE_DRIVER=devicemapper
```

/etc/docker/daemon.json 配置文件

```
[root@k8s-node01 ~]*#* *cat /etc/docker/daemon.json*
{

  "storage-driver": "devicemapper",

  "storage-opts": [

•    "dm.thinpooldev=/dev/mapper/docker-thinpool",

•    "dm.use_deferred_removal=true",

•    "dm.use_deferred_deletion=true",

•    "dm.basesize=20G"

  ],

  "log-driver": "json-file",

  "log-opts": {

•    "max-size": "200m",

•    "max-file": "5",

•    "labels": "prod"

  },

  "insecure-registries": [

•    "192.168.0.0/16"
```

```
  ],
  "dns": [
    "10.254.0.2",
    "61.139.2.69"
  ],
    "selinux-enabled": false,
    "dns-search": [
        "default.svc.cluster.local",
        "svc.cluster.local"
     ],
    "dns-opt": [
        "ndots:2",
        "timeout:2",
        "attempts:2"
     ]
}
```

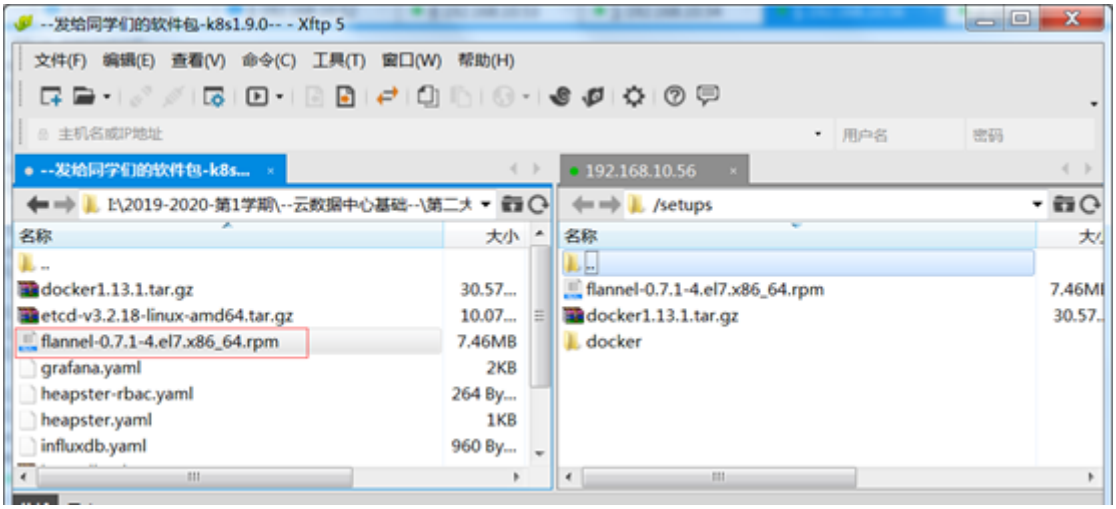
五、K8s1.9生产环境高可用集群部署手册-flannel网络插件

5.1、安装flannel

在k8s-node01上：

cd /setups/

把flannel-0.7.1-4.el7.x86_64.rpm上传到k8s-node01的/setups/目录下。然后分别拷贝到其他两个节点：



分发二进制文件到集群所有节点：

```
[root@k8s-node01~]# cd /setups/

[root@ k8s-node01 setups]# source /etc/kubernetes/environment.sh

[root@ k8s-node01 setups]# for node_all_ip in ${NODE_ALL_IPS[@]}

 do

  echo ">>> ${node_all_ip}"

  ssh root@${node_all_ip} "mkdir -p /setups/"

  scp -r /setups/flannel-0.7.1-4.el7.x86_64.rpm root@${node_all_ip}:/setups/

  ssh root@${node_all_ip} "yum localinstall -y /setups/flannel-0.7.1-
4.el7.x86_64.rpm"

 done
```

查看版本

```
[root@k8s-node01 ~]# flanneld -version
0.7.1
```

5.2、准备证书

创建 flannel 证书和私钥

flanneld 从 etcd 集群存取网段分配信息，而 etcd 集群启用了双向 x509 证书认证，所以需要为 flanneld 生成证书和私钥。

创建证书签名请求：

```
[root@k8s-etcd01~]# cd /home/key

[root@ k8s-etcd01 key]# cat > flanneld-csr.json <<EOF

{

 "CN": "flanneld",
```

```
 "hosts": [],
 "key": {
  "algo": "rsa",
  "size": 2048
 },
 "names": [
  {
   "C": "CN",
   "ST": "chengdu",
   "L": "chengdu",
   "O": "k8s",
   "OU": "System"
  }
 ]
}
EOF
```

该证书只会被 kubectl 当做 client 证书使用，所以 hosts 字段为空；

生成证书和私钥：

```
[root@ k8s-etcd01 key]# cfssl gencert -ca=/home/key/ca.pem \
 -ca-key=/home/key/ca-key.pem \
 -config=/home/key/ca-config.json \
 -profile=kubernetes flanneld-csr.json | cfssljson -bare flanneld
ls
```

```
[root@k8s-etcd01 key]# ls
admin.csr       ca.pem             flanneld.pem                      kubernetes.pem
admin-csr.json  etcd.csr           kube-controller-manager.csr       kube-scheduler.csr
admin-key.pem   etcd-csr.json      kube-controller-manager-csr.json  kube-scheduler-csr.json
admin.pem       etcd-key.pem       kube-controller-manager-key.pem   kube-scheduler-key.pem
ca-config.json  etcd.pem           kube-controller-manager.pem       kube-scheduler.pem
ca.csr          flanneld.csr       kubernetes.csr
ca-csr.json     flanneld-csr.json  kubernetes-csr.json
ca-key.pem      flanneld-key.pem   kubernetes-key.pem
```

从k8s-etcd01将生成的证书和私钥分发到所有节点（**所有节点**）：

```
[root@ k8s-etcd01 key]# cd /home/key/
source /etc/kubernetes/environment.sh
for node_all_ip in ${NODE_ALL_IPS[@]}
 do
  echo ">>> ${node_all_ip}"
  ssh root@${node_all_ip} "mkdir -p /etc/kubernetes/ssl/"
  scp flanneld*.pem root@${node_all_ip}:/etc/kubernetes/ssl/
 done
```

5.3、配置 flanneld

查看flanneld启动文件**

在所有节点中：

```
[root@k8s-node01 ~]*#* *cat /usr/lib/systemd/system/flanneld.service*

[Unit]

Description=Flanneld overlay address etcd agent

After=network.target

After=network-online.target

Wants=network-online.target

After=etcd.service

Before=docker.service


[Service]

Type=notify

EnvironmentFile=/etc/sysconfig/flanneld

EnvironmentFile=-/etc/sysconfig/docker-network

ExecStart=/usr/bin/flanneld-start $FLANNEL_OPTIONS

ExecStartPost=/usr/libexec/flannel/mk-docker-opts.sh -k DOCKER_NETWORK_OPTIONS -
d /run/flannel/docker

Restart=on-failure


[Install]

WantedBy=multi-user.target

RequiredBy=docker.service
```

解释说明：

mk-docker-opts.sh 脚本将分配给 flanneld 的 Pod 子网段信息写入 /run/flannel/docker 文件，后续 docker 启动时使用这个文件中的环境变量配置 docker0 网桥；

flanneld 使用系统缺省路由所在的接口与其它节点通信，对于有多个网络接口（如内网和公网）的节点，可以用 -iface 参数指定通信接口；

flanneld 运行时需要 root 权限；

-ip-masq: flanneld 为访问 Pod 网络外的流量设置 SNAT 规则，同时将传递给 Docker 的变量 --ip-masq（/run/flannel/docker 文件中）设置为 false，这样 Docker 将不再创建 SNAT 规则； Docker 的 --ip-masq 为 true 时，创建的 SNAT 规则比较“暴力”：将所有本节点 Pod 发起的、访问非 docker0 接口的请求做 SNAT，这样访问其他节点 Pod 的请求来源 IP 会被设置为 flannel.1 接口的 IP，导致目的 Pod 看不到真实的来源 Pod IP。 flanneld 创建的 SNAT 规则比较温和，只对访问非 Pod 网段的请求做 SNAT。

修改配置文件

在k8s-node01，修改flanneld的配置文件/etc/sysconfig/flanneld。

```
[root@k8s-node01 ~]# vi /etc/sysconfig/flanneld
```

把原来的清空，添加如下内容：

```
\# Flanneld configuration options

\# etcd url location. Point this to the server where etcd runs

FLANNEL_ETCD_ENDPOINTS="https://192.168.10.50:2379,https://192.168.10.51:2379,https://192.168.10.52:2379"



\# etcd config key. This is the configuration key that flannel queries

\# For address range assignment

FLANNEL_ETCD_PREFIX="kube-centos/network"


\# Any additional options that you want to pass

FLANNEL_OPTIONS="-etcd-cafile=/etc/kubernetes/ssl/ca.pem -etcd-certfile=/etc/kubernetes/ssl/etcd.pem -etcd-keyfile=/etc/kubernetes/ssl/etcd-key.pem"

分别flanneld的配置文件/etc/sysconfig/flanneld到所有节点：
source /etc/kubernetes/environment.sh

for node_all_ip in ${NODE_ALL_IPS[@]}
 do
  echo ">>> ${node_all_ip}"
  scp /etc/sysconfig/flanneld root@${node_all_ip}:/etc/sysconfig/
 done
```

在etcd中创建网络配置

docker分配IP地址段。
我们在k8s-etcd01中执行下面命令.

分别etcd.pem证书分发到所有节点：

```
source /etc/kubernetes/environment.sh

for node_all_ip in ${NODE_ALL_IPS[@]}

 do

  echo ">>> ${node_all_ip}"

  ssh root@${node_all_ip} "mkdir -p /etc/kubernetes/ssl/"

  scp -r /etc/etcd/ssl/{etcd.pem,etcd-key.pem}
root@${node_all_ip}:/etc/kubernetes/ssl/

 done

[root@k8s-etcd01 key]#

[root@k8s-etcd01 key]# cd /home/key

[root@k8s-etcd01 key]# ls

etcdctl --
endpoints=https://192.168.211.155:2379,https://192.168.211.154:2379,https://192.
168.211.153:2379 \
 --ca-file=/etc/kubernetes/ssl/ca.pem \
 --cert-file=/etc/kubernetes/ssl/etcd.pem \
 --key-file=/etc/kubernetes/ssl/etcd-key.pem \
 mkdir /kube-centos/network

etcdctl --
endpoints=https://192.168.211.155:2379,https://192.168.211.154:2379,https://192.
168.211.153:2379 \
 --ca-file=/etc/kubernetes/ssl/ca.pem \
 --cert-file=/etc/kubernetes/ssl/etcd.pem \
 --key-file=/etc/kubernetes/ssl/etcd-key.pem \
 mk /kube-centos/network/config
'{"Network":"172.30.0.0/16","SubnetLen":24,"Backend":{"Type":"host-gw"}}'
```

解释说明：

flanneld 当前版本 (v0.11.0) 不支持 etcd v3，故使用 etcd v2 API 写入配置 key 和网段数据；

写入的 Pod 网段 ${CLUSTER_CIDR} 地址段（如 /16）必须小于 SubnetLen，必须与 kube-controller-manager 的 --cluster-cidr 参数值一致；

执行结果如下：

```
[root@k8s-etcd01 key]*# etcdctl --
endpoints=https://192.168.10.50:2379,https://192.168.10.51:2379,https://192.168.
10.52:2379 \*

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

\>  --cert-file=/etc/kubernetes/ssl/etcd.pem \

\>  --key-file=/etc/kubernetes/ssl/etcd-key.pem \

\>  mkdir /kube-centos/network

[root@k8s-etcd01 key]*# etcdctl --
endpoints=https://192.168.10.50:2379,https://192.168.10.51:2379,https://192.168.
10.52:2379 \*

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

\>  --cert-file=/etc/kubernetes/ssl/etcd.pem \

\>  --key-file=/etc/kubernetes/ssl/etcd-key.pem \

\>  mk /kube-centos/network/config
'{"Network":"172.30.0.0/16","SubnetLen":24,"Backend":{"Type":"host-gw"}}'

{"Network":"172.30.0.0/16","SubnetLen":24,"Backend":{"Type":"host-gw"}}
```

启动flanneld

在k8s-node01中:

启动 flanneld 服务:

```
\# source /etc/kubernetes/environment.sh

\# for node_all_ip in ${NODE_ALL_IPS[@]}

 do

  echo ">>> ${node_all_ip}"

  ssh root@${node_all_ip} "systemctl daemon-reload && systemctl enable flanneld
&& systemctl restart flanneld && systemctl status flanneld"

 done
```

检查启动结果:

\

```
# source /etc/kubernetes/environment.sh

\# for node_all_ip in ${NODE_ALL_IPS[@]}

 do

  echo ">>> ${node_all_ip}"

  ssh root@${node_all_ip} "systemctl status flanneld|grep Active"

 done
```

确保状态为 active (running)，否则查看日志，确认原因"journalctl -u flanneld"

```
[root@k8s-node01 ssl]# for node_all_ip in ${NODE_ALL_IPS[@]}
>   do
>     echo ">>> ${node_all_ip}"
>     ssh root@${node_all_ip} "systemctl status flanneld|grep Active"
>   done
>>> 192.168.10.50
   Active: active (running) since Thu 2019-12-19 08:58:05 EST; 24s ago
>>> 192.168.10.51
   Active: active (running) since Thu 2019-12-19 08:58:06 EST; 24s ago
>>> 192.168.10.52
   Active: active (running) since Thu 2019-12-19 08:58:06 EST; 24s ago
>>> 192.168.10.53
   Active: active (running) since Thu 2019-12-19 08:58:06 EST; 24s ago
>>> 192.168.10.54
   Active: active (running) since Thu 2019-12-19 08:58:07 EST; 23s ago
>>> 192.168.10.56
   Active: active (running) since Thu 2019-12-19 08:58:07 EST; 23s ago
>>> 192.168.10.57
   Active: active (running) since Thu 2019-12-19 08:58:07 EST; 23s ago
>>> 192.168.10.58
   Active: active (running) since Thu 2019-12-19 08:58:08 EST; 23s ago
```

查看分配网段

在k8s-etcd01中:

```
[*root@k8s-etcd01* *key*]*#* *etcdctl* --
*endpoints=https://192.168.211.155:2379*,*https://192.168.211.154:2379*,*https:/
/192.168.211.153:2379*  --*ca*-*file=/etc/kubernetes/ssl/ca*.*pem*  --*cert*-
*file=/etc/kubernetes/ssl/etcd*.*pem*  --*key*-*file=/etc/kubernetes/ssl/etcd*-
*key*.*pem*  *ls* */kube*-*centos/network/subnets*

*/kube-centos/network/subnets/172.30.74.0-24*

*/kube-centos/network/subnets/172.30.40.0-24*

*/kube-centos/network/subnets/172.30.47.0-24*

*/kube-centos/network/subnets/172.30.55.0-24*

*/kube-centos/network/subnets/172.30.96.0-24*

*/kube-centos/network/subnets/172.30.26.0-24*

*/kube-centos/network/subnets/172.30.23.0-24*

*/kube-centos/network/subnets/172.30.22.0-24*
```

查看某一 Pod 网段对应的节点 IP 和 flannel 接口地址:

```
[root@k8s-etcd01 ~]# source /etc/kubernetes/environment.sh

[root@k8s-etcd01 ~]# etcdctl \

 --endpoints=${ETCD_ENDPOINTS} \

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

 --cert-file=/etc/kubernetes/ssl/flanneld.pem \

 --key-file=/etc/kubernetes/ssl/flanneld-key.pem \

 get */kube-centos/network/subnets/172.30.22.0-24*
```

预期输出：{"PublicIP":"192.168.10.50","BackendType":"host-gw"}

解决说明：

172.30.22.0-24 被分配给节点k8s-etcd01（192.168.10.50）；

```
[root@k8s-etcd01 ~]# etcdctl \

 --endpoints=${ETCD_ENDPOINTS} \

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

 --cert-file=/etc/kubernetes/ssl/flanneld.pem \

 --key-file=/etc/kubernetes/ssl/flanneld-key.pem \

 *get /kube-centos/network/subnets/172.30.23.0-24
```

检查节点 flannel 网络信息 （比如**k8s-master01节点）**

```
[root@k8s-master01 ~]# ip addr show

[root@k8s-master01 ~]# ip route show |grep flannel
```

验证各节点能通过 Pod 网段互通

在各节点上部署 flannel 后，检查是否创建了 flannel 接口(名称可能为 flannel0、flannel.0、flannel.1 等):

```
[root@k8s-master01 ~]# source /etc/kubernetes/environment.sh
[root@k8s-master01 ~]# for node_all_ip in ${NODE_ALL_IPS[@]}
 do
  echo ">>> ${node_all_ip}"
  ssh ${node_all_ip} "/usr/sbin/ip addr show flannel.1|grep -w inet"
 done
```

预期输出：

```
\>>> 172.16.60.241
  inet 172.30.232.0/32 scope global flannel.1
\>>> 172.16.60.242
  inet 172.30.152.0/32 scope global flannel.1
\>>> 172.16.60.243
  inet 172.30.40.0/32 scope global flannel.1
\>>> 172.16.60.244
  inet 172.30.88.0/32 scope global flannel.1
\>>> 172.16.60.245
  inet 172.30.56.0/32 scope global flannel.1
\>>> 172.16.60.246
  inet 172.30.72.0/32 scope global flannel.1
```

在各节点上 ping 所有 flannel 接口 IP，确保能通：

```
[root@k8s-master01 ~]# source /etc/kubernetes/environment.sh
[root@k8s-master01 ~]# for node_all_ip in ${NODE_ALL_IPS[@]}
 do
  echo ">>> ${node_all_ip}"
  ssh ${node_all_ip} "ping -c 1 172.30.232.0"
  ssh ${node_all_ip} "ping -c 1 172.30.152.0"
  ssh ${node_all_ip} "ping -c 1 172.30.40.0"
  ssh ${node_all_ip} "ping -c 1 172.30.88.0"
```

```
  ssh ${node_all_ip} "ping -c 1 172.30.56.0"

  ssh ${node_all_ip} "ping -c 1 172.30.72.0"

 done*
\*
```

六、K8s1.9生产环境高可用集群部署手册-node中kubelet和proxy

6.1、配置kubelet

6.1.1、准备文件

下载需要使用的文件

我们在 Kubernetes1.9生产环境高可用实践–002 中，已经上传了集群安装的所有二进制文件。

k8s-master01中查看：

```
[root@k8s-master01 bin]# cd /setups/

[root@k8s-master01 setups]# ls

kubernetes-bins kubernetes-bins.tar.gz

[root@k8s-master01 setups]# cd kubernetes-bins

[root@k8s-master01 kubernetes-bins]# ls

calico calicoctl calico-ipam etcd etcdctl kube-apiserver kube-controller-manager
kubectl kubelet kube-proxy kube-scheduler loopback VERSION.md
```

```
[root@k8s-master01 kubernetes-bins]# ll
total 793672
-rwxr-xr-x. 1 root 1000  29062464 Jan  6  2018 calico
-rwxr-xr-x. 1 root 1000  32285568 Jan  6  2018 calicoctl
-rwxr-xr-x. 1 root 1000  28424000 Jan  6  2018 calico-ipam
-rwxr-xr-x. 1 root 1000  17809440 Jan  6  2018 etcd
-rwxr-xr-x. 1 root 1000  15230304 Jan  6  2018 etcdctl
-rwxr-xr-x. 1 root 1000 209244331 Jan  6  2018 kube-apiserver
-rwxr-xr-x. 1 root 1000 136621177 Jan  6  2018 kube-controller-manager
-rwxr-xr-x. 1 root 1000  67390552 Jan  6  2018 kubectl
-rwxr-xr-x. 1 root 1000 147765224 Jan  6  2018 kubelet
-rwxr-xr-x. 1 root 1000  63378954 Jan  6  2018 kube-proxy
-rwxr-xr-x. 1 root 1000  61566971 Jan  6  2018 kube-scheduler
-rwxr-xr-x. 1 root 1000   3909976 Jan  6  2018 loopback
-rw-r--r--. 1 root 1000        78 Jan  6  2018 VERSION.md
[root@k8s-master01 kubernetes-bins]# pwd
/setups/kubernetes-bins
```

在这节中，我们使用到的文件有： kubelet和kube-proxy

下面这几步在k8s-node01、k8s-node02、k8s-node03三台服务器上分别操作：

1、关闭交换分区swap

```
cat /etc/fstab
```

2、**设置内核，修改iptables相关参数**

```
cat /etc/sysctl.d/k8s.conf

sysctl -p /etc/sysctl.d/k8s.conf   #使配置生效
```

3、**kube-proxy开启ipvs的前置条件加载ipvs相关模块**

```
modprobe br_netfilter

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

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

bash /etc/sysconfig/modules/ipvs.modules
```

报错　　忽略，下一步查看加载正确即可。

```
lsmod | grep -e ip_vs -e nf_conntrack_ipv4
```

脚本创建了的/etc/sysconfig/modules/ipvs.modules文件，保证在节点重启后能自动加载所需模块。使用lsmod | grep -e ip_vs -e nf_conntrack_ipv4命令查看是否已经正确加载所需的内核模块。

```
[root@k8s-node02 ~]# lsmod | grep -e ip_vs -e nf_conntrack_ipv4
ip_vs_sh               16384  0
ip_vs_wrr              16384  0
ip_vs_rr               16384  0
ip_vs                 155648  6 ip_vs_rr,ip_vs_sh,ip_vs_wrr
nf_conntrack          147456  4 xt_conntrack,nf_nat,xt_MASQUERADE,ip_vs
nf_defrag_ipv6         24576  2 nf_conntrack,ip_vs
libcrc32c              16384  4 nf_conntrack,nf_nat,xfs,ip_vs
```

为了便于查看ipvs的代理规则，最好安装一下管理工具ipvsadm。

```
\# yum install ipset ipvsadm -y
```

6.1.2、配置kubelet

目录结构：

```
[root@yds-dev-svc02-node01 kubernetes]# tree
.
├── bootstrap.kubeconfig
├── config
├── kubelet
├── kubelet.kubeconfig
├── kube-proxy.kubeconfig
├── proxy
└── ssl
    ├── ca.pem
    ├── etcd-key.pem
    ├── etcd.pem
    ├── kubelet-client.crt
    ├── kubelet-client.key
    ├── kubelet.crt
    └── kubelet.key

1 directory, 13 files
```

第一部分：可执行文件，为前面下载的kubelet-bin二进制包中内容。我们统一放在了/usr/bin/目录下。

第二部分：证书文件为前面部署master时生成的。

第三部分：cfg配置文件：

1）基本配置文件

2）连接api配置文件

3）还有主要配置文件

准备**kubelet**

在k8s-master01操作：

把二进制文件分发到node节点：

```
source /etc/kubernetes/environment.sh echo $NODE_NODE_IPS

[root@k8s-master01 ~]# cd /setups/kubernetes-bins/

[root@k8s-master01 kubernetes-bins]# source /etc/kubernetes/environment.sh

[root@k8s-master01 kubernetes-bins]# for node_node_ip in ${NODE_NODE_IPS[@]}

 do

  echo ">>> ${node_node_ip}"

  scp -r /setups/kubernetes-bins/kubelet root@${node_node_ip}:/usr/bin/

  ssh root@${node_node_ip} "chmod +x /usr/bin/*"

 done
```

下载**pod-infrastructure**镜像（此处是从网络源安装组件包，docker也是从docker镜像源网络源获取资源，这里拉取需要等待一段时间才完成）

在k8s-node01操作：

```
source /etc/kubernetes/environment.sh
```

```
echo $NODE_NODE_IPS

[root@k8s-master01 ~]# for node_node_ip in ${NODE_NODE_IPS[@]}

 do

  echo ">>> ${node_node_ip}"

  ssh root@${node_node_ip} "yum install -y *rhsm*"

  ssh root@${node_node_ip} "wget
http://mirror.centos.org/centos/7/os/x86_64/Packages/python-rhsm-certificates-
1.19.10-1.el7_4.x86_64.rpm
 rpm2cpio python-rhsm-certificates-1.19.10-1.el7_4.x86_64.rpm | cpio -iv --to-
stdout ./etc/rhsm/ca/redhat-uep.pem | tee /etc/rhsm/ca/redhat-uep.pem"

  ssh root@${node_node_ip} " **mkdir -p /etc/sysconfig**

cat > /etc/sysconfig/docker <<EOF

*OPTIONS='--selinux-enabled --log-driver=journald --registry-
mirror=https://docker.mirrors.ustc.edu.cn'*

*EOF*

cat > daemon.json<<eof

*{*

 *"registry-mirrors": ["https://registry.docker-cn.com","http://hub-
mirror.c.163.com"]*

*}*
*eof*

service docker restart"

  ssh root@${node_node_ip} "docker pull registry.access.redhat.com/rhel7/pod-
infrastructure:latest"

 done
```

6.1.3、 准备证书文件

在k8s-etcd01中进行查看。

查看token：

[root@k8s-etcd01 key]# *cd /home/key*

[root@k8s-etcd01 key]# ls

```
[root@k8s-etcd01 key]# ls
admin.csr       ca.pem             flanneld.pem                      kubernetes.pem
admin-csr.json  etcd.csr           kube-controller-manager.csr       kube-scheduler.csr
admin-key.pem   etcd-csr.json      kube-controller-manager-csr.json  kube-scheduler-csr.json
admin.pem       etcd-key.pem       kube-controller-manager-key.pem   kube-scheduler-key.pem
ca-config.json  etcd.pem           kube-controller-manager.pem       kube-scheduler.pem
ca.csr          flanneld.csr       kubernetes.csr
ca-csr.json     flanneld-csr.json  kubernetes-csr.json
ca-key.pem      flanneld-key.pem   kubernetes-key.pem
```

创建**kubelet bootstrap kubeconfig配置文件****

这一步，我们会在安装**kubectl的k8s-master01上面执行。****

*在**k8s-master01**中:*

```
cd /etc/kubernetes
```

*##**首先指定**kube-api**访问入口，即**master ip*

```
source /etc/kubernetes/environment.sh

echo $KUBE_APISERVER

cd /etc/kubernetes

cat token.csv

echo $BOOTSTRAP_TOKEN
```

确保$BOOTSTRAP_TOKEN与token.csv里面的token一致，同时跟环境变量里面的一致，不一致就export一下。

```
export BOOTSTRAP_TOKEN=5cc80ff9854de087a636deee421004e5

source /etc/kubernetes/environment.sh

cd /etc/kubernetes
```

#设置集群参数

```
kubectl config set-cluster kubernetes \

 --certificate-authority=/etc/kubernetes/ssl/ca.pem \

 --embed-certs=true \

 --server=${KUBE_APISERVER} \

 --kubeconfig=bootstrap.kubeconfig
```

#设置客户端认证参数

```
kubectl config set-credentials kubelet-bootstrap \

 --token=${BOOTSTRAP_TOKEN} \

 --kubeconfig=bootstrap.kubeconfig
```

#设置上下文参数

```
kubectl config set-context default \

 --cluster=kubernetes \

 --user=kubelet-bootstrap \

 --kubeconfig=bootstrap.kubeconfig
```

#设置默认上下文

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

ls
```

```
解释说明:
->``向 kubeconfig 写入的是 token，bootstrap 结束后 kube-controller-manager 为
kubelet 创建 client 和 server 证书；
-> ``如果用kubeadm` 生成的`token ``有效期为 1 天，超期后将不能再被用来 boostrap
kubelet，且会被 kube-controller-manager 的 tokencleaner 清理；但是我们这里用的永久的
token。
-> kube-apiserver ``接收 kubelet 的 bootstrap token 后，将请求的 user 设置为
system:bootstrap:，group 设置为 system:bootstrappers，
后续将为这个 group 设置 ClusterRoleBinding；
```

#**执行完后从`k8s-master01`将生成bootstrap.kubeconfig文件分发bootstrap.kubeconfig文件到所有node节点；**

```
[root@k8s-master01 ~]# cd /etc/kubernetes
[root@k8s-master01 kubernetes]# source /etc/kubernetes/environment.sh
[root@k8s-master01 kubernetes]# for node_node_ip in ${NODE_NODE_IPS[@]}
 do
  echo` `">>> ${node_node_ip}"
  scp -r /etc/kubernetes/bootstrap.kubeconfig
root@${node_node_ip}:/etc/kubernetes/
 done
```

6.1.4、 创建bootstrap配置文件

这一步，我们会在安装*kubectl的k8s-master01上面执行。

kubelet 启动时向 kube-apiserver 发送 TLS bootstrapping 请求，需要先将 bootstrap token 文件中的 kubelet-bootstrap 用户赋予 system:node-bootstrapper cluster 角色(role)， 然后 kubelet 才能有权限创建认证请求(certificate signing requests)：

k8s-master01上

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

[root@k8s-master01 kubernetes]# ls

apiserver config controller-manager scheduler ssl token.csv

[root@k8s-master01 kubernetes]# kubectl create clusterrolebinding kubelet-
bootstrap --clusterrole=system:node-bootstrapper --user=kubelet-bootstrap

*clusterrolebinding "kubelet-bootstrap" created
```

查看创建结果:

```
[root@k8s-master01 kubernetes]# kubectl get clusterrolebinding

NAME                           AGE

cluster-admin                      8d

kubelet-bootstrap                    3m

system:aws-cloud-provider               8d

system:basic-user                    8d

system:controller:attachdetach-controller       8d

system:controller:certificate-controller        8d

system:controller:clusterrole-aggregation-controller  8d

system:controller:cronjob-controller          8d

system:controller:daemon-set-controller        8d

system:controller:deployment-controller        8d

system:controller:disruption-controller        8d

system:controller:endpoint-controller         8d

system:controller:generic-garbage-collector      8d

system:controller:horizontal-pod-autoscaler      8d

system:controller:job-controller            8d

system:controller:namespace-controller         8d

system:controller:node-controller            8d

system:controller:persistent-volume-binder       8d

system:controller:pod-garbage-collector        8d

system:controller:replicaset-controller        8d
```

```
system:controller:replication-controller        8d

system:controller:resourcequota-controller       8d

system:controller:route-controller           8d

system:controller:service-account-controller      8d

system:controller:service-controller          8d

system:controller:statefulset-controller        8d

system:controller:ttl-controller            8d

system:discovery                    8d

system:kube-controller-manager             8d

system:kube-dns                     8d

system:kube-scheduler                  8d

system:node                      8d

system:node-proxier                   8d
```

查看描述：

```
[root@k8s-master01 kubernetes]# kubectl describe clusterrolebinding kubelet-
bootstrap

Name:     kubelet-bootstrap

Labels:    <none>

Annotations: <none>

Role:

 Kind: ClusterRole

 Name: system:node-bootstrapper

Subjects:

 Kind Name        Namespace

 ---- ----        ---------

 User kubelet-bootstrap
```

查看内容：

```
[root@k8s-master01 kubernetes]# kubectl edit clusterrolebinding kubelet-
bootstrap
```

```
\# Please edit the object below. Lines beginning with a '#' will be ignored,

\# and an empty file will abort the edit. If an error occurs while saving this
file will be

\# reopened with the relevant failures.

\#

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRoleBinding

metadata:

 creationTimestamp: 2018-04-17T08:01:22Z

 name: kubelet-bootstrap

 resourceVersion: "528680"

 selfLink: /apis/rbac.authorization.k8s.io/v1/clusterrolebindings/kubelet-
bootstrap

 uid: 851e77fc-4215-11e8-b786-000c2948d8a8

roleRef:

 apiGroup: rbac.authorization.k8s.io

 kind: ClusterRole

 name: system:node-bootstrapper

subjects:

\- apiGroup: rbac.authorization.k8s.io

 kind: User

 name: kubelet-bootstrap
```

不需要修改，直接退出即可。

6.1.5、创建kubelet配置文件

在k8s-node01上面执行**

创建kubelet配置文件模板：

配置文件地址为：/etc/kubernetes/kubelet

```
[root@k8s-node01 ~]#

cat > /etc/kubernetes/kubelet.template <<EOF
```

```
\###

\## kubernetes kubelet (minion) config

\#

\## The address for the info server to serve on (set to 0.0.0.0 or "" for all
interfaces)

KUBELET_ADDRESS="--address=`##NODE_NODE_IP##`"

\#

\## The port for the info server to serve on

\#KUBELET_PORT="--port=10250"

\#

\## You may leave this blank to use the actual hostname

KUBELET_HOSTNAME="--hostname-override=`##NODE_NODE_NAME##`"

\#KUBELET_API_SERVER="--api-servers=http://192.168.10.55:8080"

\#

\## pod infrastructure container

KUBELET_POD_INFRA_CONTAINER="--pod-infra-container-
image=registry.access.redhat.com/rhel7/pod-infrastructure:latest"

\#

\## Add your own!

KUBELET_ARGS="--runtime-cgroups=/systemd/system.slice --kubelet-
cgroups=/systemd/system.slice --cgroup-driver=systemd --fail-swap-on=false --
tls-cert-file=/etc/kubernetes/ssl/kubernetes.pem --tls-private-key-
file=/etc/kubernetes/ssl/kubernetes-key.pem --client-ca-
file=/etc/kubernetes/ssl/ca.pem --experimental-bootstrap-
kubeconfig=/etc/kubernetes/bootstrap.kubeconfig --
kubeconfig=/etc/kubernetes/kubelet.kubeconfig --cluster-dns=10.254.0.2 --cert-
dir=/etc/kubernetes/ssl --cluster-domain=cluster.local. --hairpin-mode
promiscuous-bridge --serialize-image-pulls=false --logtostderr false --log-dir
/var/log/kubernetes --v 2"

EOF
```

KUBELET_ADDRESS: 填写本节点的IP地址。 注意在每个节点自己修改本服务器的对应的地址。
KUBELET_HOSTNAME: 填写本节点的主机名,配置这里明显的影响是‘kubectl get nodes’这个命令的输出。
KUBELET_API_SERVER: 填写我们前面配置的apiserver地址。
cert-dir： 自动生成证书的存放路径。
tls-cert-file： 指向apiserver证书
tls-private-key-file： 指向apiserver key

这里只需要建个模板，不用单独复制为kubelet**文件，后面使用***for语句分发并复制kubelet文件即可。**kubelet**内容应该为：（KUBELET_HOSTNAME不同）**

```
[root@k8s-node01 kubernetes]# cat kubelet

\###

\## kubernetes kubelet (minion) config

\#

\## The address for the info server to serve on (set to 0.0.0.0 or "" for all
interfaces)
```

KUBELET_ADDRESS="--address=192.168.10.56"

```
#

\## The port for the info server to serve on

\#KUBELET_PORT="--port=10250"

\#

\## You may leave this blank to use the actual hostname

KUBELET_HOSTNAME="--hostname-override=**k8s-node01**"

\#KUBELET_API_SERVER="--api-servers=http://192.168.10.55:8080"

\#

\## pod infrastructure container

KUBELET_POD_INFRA_CONTAINER="--pod-infra-container-
image=registry.access.redhat.com/rhel7/pod-infrastructure:latest"

\#

\## Add your own!

KUBELET_ARGS="--runtime-cgroups=/systemd/system.slice --kubelet-
cgroups=/systemd/system.slice --cgroup-driver=systemd --fail-swap-on=false --
tls-cert-file=/etc/kubernetes/ssl/kubernetes.pem --tls-private-key-
file=/etc/kubernetes/ssl/kubernetes-key.pem --client-ca-
file=/etc/kubernetes/ssl/ca.pem --experimental-bootstrap-
kubeconfig=/etc/kubernetes/bootstrap.kubeconfig --
kubeconfig=/etc/kubernetes/kubelet.kubeconfig --cluster-dns=10.254.0.2 --cert-
dir=/etc/kubernetes/ssl --cluster-domain=cluster.local. --hairpin-mode
promiscuous-bridge --serialize-image-pulls=false --logtostderr false --log-dir
/var/log/kubernetes --v 2"
```

在k8s-master01上面执行:**

先到把证书和秘钥 `kubernetes*.pem ca.pem``分发` **到各**node节点：**

```
[root@k8s-master01 ~]# cd /etc/kubernetes/
[root@k8s-master01 kubernetes]# source /etc/kubernetes/environment.sh
[root@k8s-master01 kubernetes]# for node_node_ip in ${NODE_NODE_IPS[@]}
 do
  echo` `">>> ${node_node_ip}"
  scp` -r `/etc/kubernetes/ssl/kubernetes*.pem /etc/kubernetes/ssl/ca.pem
root@${node_node_ip}:/etc/kubernetes/ssl/
 done
```

在k8s-node01上面执行:**

分发 kubelet 配置文件到各node节点：**

```
[root@k8s-node01 ~]# cd /etc/kubernetes/
[root@k8s-node01 kubernetes]# source /etc/kubernetes/environment.sh
[root@k8s-node01 kubernetes]# for node_node_ip in ${NODE_NODE_IPS[@]}
 do
  echo` `">>> ${node_node_ip}"
  scp` -r `/etc/kubernetes/kubelet.template
root@${node_node_ip}:/etc/kubernetes/
  ssh` `root@${node_node_ip} "sed` `-e "s/##NODE_NODE_IP##/${node_node_ip}/"
/etc/kubernetes/kubelet.template > /etc/kubernetes/kubelet"
 done

查看一下：
cat /etc/kubernetes/kubelet

修改KUBELET_HOSTNAME：
[root@k8s-node01 kubernetes]# for node_node_name in ${NODE_NODE_NAMES[@]}
 do
  echo` `">>> ${node_node_name}"
  ssh` `root@${node_node_name} "sed` `-i
"s/##NODE_NODE_NAME##/${node_node_name}/" /etc/kubernetes/kubelet"
 done
```

```
查看一下：
cat /etc/kubernetes/kubelet
```

6.1.6、创建config配置文件

在k8s-node01上面执行：**

```
[root@k8s-node01 ~]#

cat > /etc/kubernetes/config <<EOF

\###

\# kubernetes system config
```

```
\#

\# The following values are used to configure various aspects of all

\# kubernetes services, including

\#

\#  kube-apiserver.service

\#  kube-controller-manager.service

\#  kube-scheduler.service

\#  kubelet.service

\#  kube-proxy.service

\# logging to stderr means we get it in the systemd journal

KUBE_LOGTOSTDERR="--logtostderr=true"

\# journal message level, 0 is debug

KUBE_LOG_LEVEL="--v=0"

\# Should this cluster be allowed to run privileged docker containers

KUBE_ALLOW_PRIV="--allow-privileged=true"

EOF
```

从k8s-node01分发** **kubelet 配置文件**config到各node节点：**

```
[root@k8s-master01 ~]# cd /etc/kubernetes/
[root@k8s-master01 kubernetes]# source /etc/kubernetes/environment.sh
[root@k8s-master01 kubernetes]# for node_node_ip in ${NODE_NODE_IPS[@]}
 do
  echo` `">>> ${node_node_ip}"
  scp` -r `/etc/kubernetes/config root@${node_node_ip}:/etc/kubernetes/
 done
```

6.1.7、 创建service配置文件

在k8s-node01上面执行：**

创建配置文件: /usr/lib/systemd/system/kubelet.service

```
[root@k8s-node01 ~]*#* *vi /usr/lib/systemd/system/kubelet.service*

[Unit]

Description=Kubernetes Kubelet Server
```

```
Documentation=https:*//github.com/GoogleCloudPlatform/kubernetes*

After=docker.service

Requires=docker.service

[Service]

WorkingDirectory=/var/lib/*kube-*kubelet

EnvironmentFile=-/etc/kubernetes/config

EnvironmentFile=-/etc/kubernetes/kubelet

ExecStart=/usr/bin/kubelet \

•        $KUBE_LOGTOSTDERR \

•       $KUBE_LOG_LEVEL \

•       $KUBELET_API_SERVER \

•       $KUBELET_ADDRESS \

•       $KUBELET_PORT \

•       $KUBELET_HOSTNAME \

•       $KUBE_ALLOW_PRIV \

•       $KUBELET_POD_INFRA_CONTAINER \

•       $KUBELET_ARGS

Restart=on-failure

[Install]

WantedBy=multi-user.target
```

从k8s-node01**分发 kubelet 配置文件****kubelet.service到各node节点：**

```
[root@k8s-node01 ~]# cd `*/usr/lib/systemd/system/*
[root@k8s-node01 kubernetes]# source /etc/kubernetes/environment.sh
[root@k8s-node01 kubernetes]# for node_node_ip in ${NODE_NODE_IPS[@]}
 do
  echo">>> ${node_node_ip}"
  sshroot@${node_node_ip} "mkdir -p /var/lib/kube-kubelet"、
  scp -r`*/usr/lib/systemd/system/kubelet.service*`
root@${node_node_ip}:`*/usr/lib/systemd/system/*``
 done
```

注意：在node节点上要创建kubelet工作目录。

启动 kubelet 服务：

```
[root@k8s-node01 ~]# source /etc/kubernetes/environment.sh
[root@k8s-node01 ~]# for node_node_ip in ${NODE_NODE_IPS[@]}
 do
  echo` `">>> ${node_node_ip}"
  ssh` `root@${node_node_ip} "systemctl daemon-reload && systemctl enable
kubelet && systemctl restart kubelet && systemctl status kubelet"
 done
```

在k8s-master01上：

```
[root@k8s-master01 ~]# kubectl get csr
[root@k8s-master01 ~]# kubectl get nodes
```

```
[root@k8s-master01 ssl]# kubectl get csr
NAME                                                   AGE       REQUESTOR           CONDITION
node-csr-FaaJE8LvB1nZzZ9CfFame197LojzlEicKBDLyeQczI4   14s       kubelet-bootstrap   Pending
node-csr-lWgqmCdbAA0b3Z3l6cwStYs2ahZQ65Q89lz2eINu98M   13s       kubelet-bootstrap   Pending
node-csr-tVogQDG1CX5L1iV1bx_SVhvB2GacEKcwTjxdXFVrKQU   12s       kubelet-bootstrap   Pending
[root@k8s-master01 ssl]# kubectl get nodes
No resources found.
[root@k8s-master01 ssl]#
```

6.2、配置与启动kube-proxy

6.2.1、配置kube-proxy

准备kube-proxy**

在k8s-master01操作：

把二进制文件分发到node节点：

```
echo $NODE_NODE_IPS

[root@k8s-master01 ~]# cd /setups/kubernetes-bins/

[root@k8s-master01 kubernetes-bins]# source /etc/kubernetes/environment.sh

[root@k8s-master01 kubernetes-bins]# for node_node_ip in ${NODE_NODE_IPS[@]}

 do

  echo ">>> ${node_node_ip}"

  scp -r /setups/kubernetes-bins/kube-proxy root@${node_node_ip}:/usr/bin/

  ssh root@${node_node_ip} "chmod +x /usr/bin/*"

 done
```

*准备证书文件**

我们需要再创建proxy的证书文件。
和前面一样，还是回到服务器k8s-etcd01中进行创建。

创建kube-proxy-csr.json**

```
```

```
[root@k8s-etcd01 key]*#* *cd /home/key*

[root@k8s-etcd01 key]*#*` cat > `*kube-proxy-csr.json*` <

{

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

 "hosts": [],

 "key": {

  "algo": "rsa",

  "size": 2048

 },

 "names": [

  {

   "C": "CN",

   "ST": "chengdu",

   "L": "chengdu",

   "O": "k8s",

   "OU": "System"

  }

 ]

}
EOF
```

CN 名称为 system:kube-proxy，需要与 metrics-server 的 --requestheader-allowed-names 参数配置一致，否则访问会被 metrics-server 拒绝；

```
预定义的 RoleBinding system:node-proxier 将User system:kube-proxy 与 Role system:node-proxier 绑定，该 Role 授予了调用 kube-apiserver Proxy 相关 API 的权限；
该证书只会被 kube-proxy 当做 client 证书使用，所以 hosts 字段为空；
```

使用**cfssl命令创建证书****

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

查看创建的证书

```
[root@k8s-etcd01 key]# ls kube-proxy*

kube-proxy.csr kube-proxy-csr.json kube-proxy-key.pem kube-proxy.pem

[root@k8s-etcd01 key]# pwd

/home/key
```

分发到k8s-master01上：

```
scp kube-proxy.csr kube-proxy-csr.json kube-proxy-key.pem kube-proxy.pem
root@192.168.10.53:/etc/kubernetes/ssl
```

创建proxy kubeconfig**配置文件**

这一步，我们会在安装**kubectl的k8s-master01上面执行。**

*在**k8s-master01**中：*

```
cd /etc/kubernetes
```

#***首先指定****kube-api****访问入口，即****master ip*

```
[root@k8s-master01 ~]# *echo $KUBE_APISERVER*

[root@k8s-master01 ~]# source /etc/kubernetes/environment.sh

[root@k8s-master01 ~]# *echo $KUBE_APISERVER*

cd /etc/kubernetes
```

**** 配置集群 ****

```
kubectl config set-cluster kubernetes \

 --certificate-authority=/etc/kubernetes/ssl/ca.pem \

 --embed-certs=true \

 --server=https://192.168.10.55:6443 \

 --kubeconfig=kube-proxy.kubeconfig
```

**** 配置客户端认证 ****

```
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=kube-proxy.kubeconfig
```

**** 配置关联****

```
kubectl config set-context default \

 --cluster=kubernetes \

 --user=kube-proxy \

 --kubeconfig=kube-proxy.kubeconfig
```

**** 配置默认关联****

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

```
注意：--embed-certs=true：将 ca.pem 和 admin.pem 证书内容嵌入到生成的 kubectl-proxy.kubeconfig 文件中(不加时，写入的是证书文件路径)；
```

配置完成后，会生成kube-proxy.kubeconfig文件。接下来。我们把这个文件分发到所有node节点的/etc/kubernetes目录中。

*在**k8s-master01**中:*

 # 分发proxy.kubeconfig文件到所有node节点;

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

[root@k8s-master01 ~]# source /etc/kubernetes/environment.sh

[root@k8s-master01 ~]# for node_node_ip in ${NODE_NODE_IPS[@]}

 do

  echo ">>> ${node_node_ip}"

  scp -r  /etc/kubernetes/kube-proxy.kubeconfig
root@${node_node_ip}:/etc/kubernetes/

 done
```

创建**proxy配置文件****

在**k8s-node01上面执行：****

```
[root@k8s-node01 kubernetes]*#* *cd /etc/kubernetes/*

[root@k8s-node01 kubernetes]*#*

`cat > `*/etc/kubernetes/proxy*` <

 KUBE_PROXY_ARGS="--kubeconfig=/etc/kubernetes/kube-proxy.kubeconfig"
EOF
```

在**k8s-node01上：****

 # 分发proxy文件到所有node节点;

```
[root@k8s-node01 ~]# source /etc/kubernetes/environment.sh

[root@k8s-node01 ~]# for node_node_ip in ${NODE_NODE_IPS[@]}

 do

  echo ">>> ${node_node_ip}"

  scp -r  */etc/kubernetes/proxy* root@${node_node_ip}:/etc/kubernetes/

 done
```

创建**service**文件**

在**k8s-node01上面执行：**

```
[root@k8s-node01 kubernetes]*#* *vi /usr/lib/systemd/system/kube-proxy.service*

[Unit]

Description=Kubernetes Proxy

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

After=network.target

[Service]

WorkingDirectory=/var/lib/kube-proxy

EnvironmentFile=-/etc/kubernetes/config

EnvironmentFile=-/etc/kubernetes/proxy

ExecStart=/usr/bin/kube-proxy \

•    $KUBE_LOGTOSTDERR \

•    $KUBE_LOG_LEVEL \

•    $KUBE_MASTER \

•    $KUBE_PROXY_ARGS

Restart=on-failure

LimitNOFILE=65536

[Install]

WantedBy=multi-user.target
```

 # 从**k8s-node01分发kube-proxy.service文件到所有node节点；**

```
[root@k8s-node01 ~]# source /etc/kubernetes/environment.sh

[root@k8s-node01 ~]# for node_node_ip in ${NODE_NODE_IPS[@]}

 do

  echo ">>> ${node_node_ip}"

  ssh root@${node_node_ip} "mkdir -p /var/lib/kube-proxy"

  scp -r */usr/lib/systemd/system/kube-proxy.service*
root@${node_node_ip}:*/usr/lib/systemd/system/*

 done
```

安装**conntrack-tools**

在k8s-node01、k8s-node02、k8s-node03三台上面都要安装。

我们只需要在k8s-node01操作：

把整理过的epel源上传到setups目录下：

```
[root@k8s-node01 ~]# source /etc/kubernetes/environment.sh

[root@k8s-node01 ~]# for node_node_ip in ${NODE_NODE_IPS[@]}

 do

  echo ">>> ${node_node_ip}"

  scp -r */setups/epel.tar.gz* root@${node_node_ip}:*/setups/*

  ssh root@${node_node_ip} "cd /setups/

tar -xzvf epel.tar.gz"

  ssh root@${node_node_ip} "yum -y install createrepo"

  ssh root@${node_node_ip} "createrepo -p -d -o /setups/epel/ /setups/epel/"

  ssh root@${node_node_ip} "`cat > `/etc/yum.repos.d/epel.repo` <

[CentOS-epel-yum]

name=CentOS epel yum Repository

baseurl=file://**/setups**/epel/

gpgcheck=0

enabled=1

`EOF`"

  ssh root@${node_node_ip} "*yum install -y conntrack-tools*"
```

```
 done
```

6.2.2、启动服务

查看etcd**节点状态（**在三台**ETCD**节点上**）：**

```
systemctl status etcd
```

查看master上的几个服务状态（两台master节点上）：

```
systemctl status kube-apiserver.service

systemctl status kube-controller-manager.service

systemctl status kube-scheduler.service
```

如果三个服务状态有报错，那么在k8s-master01和k8s-master02上进行以下操作：

修改文件：vi /etc/kubernetes/apiserver

将KUBE_API_ADDRESS="--insecure-bind-address=127.0.0.1"改为KUBE_API_ADDRESS="--insecure-bind-address=0.0.0.0"

sed -i "s/--insecure-bind-address=127.0.0.1/--insecure-bind-address=0.0.0.0/g" /etc/kubernetes/apiserver

然后，重启服务：

```
systemctl restart kube-apiserver

systemctl restart kube-controller-manager.service

systemctl restart kube-scheduler.service

systemctl status kube-apiserver.service

systemctl status kube-controller-manager.service

systemctl status kube-scheduler.service
```

在k8s-node01上面执行：

```
**启动 kube-proxy 服务：**
```

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

```
[root@k8s-master01 ~]# source /etc/kubernetes/environment.sh
[root@k8s-master01 ~]# for node_node_ip in ${NODE_NODE_IPS[@]}
 do
  echo` `">>> ${node_node_ip}"
  ssh` `root@${node_node_ip} "systemctl daemon-reload && systemctl enable kube-
proxy && systemctl restart kube-proxy && systemctl status kube-proxy"
 done
```

解释说明：
-> ``启动服务前必须先创建工作目录；
-> ``关闭 swap 分区，否则 kubelet 会启动失败 (使用"journalctl -u kubelet |tail"命令查看错误日志)

kubelet ``启动后使用 --bootstrap-kubeconfig 向 kube-apiserver 发送 CSR 请求，
当这个 CSR 被 approve 后，kube-controller-manager 为 kubelet 创建 TLS 客户端证书、私钥和 --kubeletconfig 文件。

注意：kube-controller-manager 需要配置 --cluster-signing-cert-file` `和 --cluster-signing-key-file` `参数，才会为 TLS Bootstrap 创建证书和私钥。

检查启动结果：
```
[root@k8s-master01 ~]# source /etc/kubernetes/environment.sh
[root@k8s-master01 ~]# for node_node_ip in ${NODE_NODE_IPS[@]}
 do
  echo` `">>> ${node_node_ip}"
  ssh` `root@${node_node_ip} "systemctl status kube-proxy|grep Active"
 done
```

确保状态为 active (running)，否则查看日志，确认原因（journalctl -u kube-proxy）

查看监听端口（在任意一台node节点上查看）：
```
[root@k8s-node01 ~]# netstat -lnpt|grep kube-prox
```

```
[root@k8s-node01 setups]# for node_node_ip in ${NODE_NODE_IPS[@]}
>   do
>     echo ">>> ${node_node_ip}"
>     ssh root@${node_node_ip} "systemctl status kube-proxy|grep Active"
>   done
>>> 192.168.10.56
   Active: active (running) since Fri 2019-12-27 01:42:10 EST; 10s ago
>>> 192.168.10.57
   Active: active (running) since Fri 2019-12-27 01:42:11 EST; 9s ago
>>> 192.168.10.58
   Active: active (running) since Fri 2019-12-27 01:42:12 EST; 8s ago
[root@k8s-node01 setups]# netstat -lnpt|grep kube-prox
tcp        0      0 127.0.0.1:10249         0.0.0.0:*               LISTEN      10502/kube-proxy
tcp6       0      0 :::10256                :::*                    LISTEN      10502/kube-proxy
[root@k8s-node01 setups]#
```

查看 ipvs 路由规则：
```
[root@k8s-master01 ~]# source /etc/kubernetes/environment.sh
[root@k8s-master01 ~]# for node_node_ip in ${NODE_NODE_IPS[@]}
 do
  echo` `">>> ${node_node_ip}"
  ssh` `root@${node_node_ip} "/usr/sbin/ipvsadm -ln"
 done
```

发送证书签名请求

kubelet 首次启动时会向apiserver发送证书签名请求，apiserver通过才会将该 Node 加入到集群。

在k8s-master01上操作：
查看节点发送的证书签名请求命令为：
kubectl get certificatesigningrequests 或者
kubectl get csr 这两个命令是一样的。

```
[root@k8s-master01 ~]# kubectl get certificatesigningrequests
[root@k8s-master01 ~]# kubectl get csr
```

同意签名请求

由于需要apiserver同意签名请求，因此，我们需要通过kubectl工具来执行。这里我们在服务器k8s-master01中执行。

```
[root@k8s-master01 ~]*#* *kubectl certificate approve* *node-csr-FaaJE8LvB1nZzZ9CfFame197LojzlEicKBDLyeQczI4*
```

注意这里改为自己的发送请求的三个*node**节点的签名，

三个节点执行三次，分别改好，再执行。

```
*kubectl certificate approve xxx*
*kubectl certificate approve xxxx*
*kubectl certificate approve xxxxx*
```

kubectl certificate approve xxxx

如下图所示：

```
[root@k8s-master01 kubernetes]# kubectl get csr
NAME                                                   AGE       REQUESTOR           CONDITION
node-csr-FaaJE8LvB1nZzZ9CfFame197LojzlEicKBDLyeQczI4   26m       kubelet-bootstrap   Pending
node-csr-lWgqmCdbAA0b3Z3l6cwStYs2ahZQ65Q89lz2eINu98M   26m       kubelet-bootstrap   Pending
node-csr-tVogQDG1CX5L1iV1bx_SVhvB2GacEKcwTjxdXFVrKQU   26m       kubelet-bootstrap   Pending
[root@k8s-master01 kubernetes]# kubectl certificate approve node-csr-FaaJE8LvB1nZzZ9CfFame197LojzlEicKBDLyeQczI4
certificatesigningrequest "node-csr-FaaJE8LvB1nZzZ9CfFame197LojzlEicKBDLyeQczI4" approved
[root@k8s-master01 kubernetes]# kubectl certificate approve node-csr-lWgqmCdbAA0b3Z3l6cwStYs2ahZQ65Q89lz2eINu98M
certificatesigningrequest "node-csr-lWgqmCdbAA0b3Z3l6cwStYs2ahZQ65Q89lz2eINu98M" approved
[root@k8s-master01 kubernetes]# kubectl certificate approve node-csr-tVogQDG1CX5L1iV1bx_SVhvB2GacEKcwTjxdXFVrKQU
certificatesigningrequest "node-csr-tVogQDG1CX5L1iV1bx_SVhvB2GacEKcwTjxdXFVrKQU" approved
```

检查证书生成

k8s-node01、**k8s-node02、k8s-node03**上：

我们在同意签名请求后，节点服务器会自动生成证书文件，证书文件存放目录在我们前面的配置文件中已经配置的/etc/kubernetes/ssl。现在我们看下这个目录中的生成文件。

```
[root@k8s-node01 ssl]# ls /etc/kubernetes/ssl/kubelet*
```

检查节点信息

还记得我们配置kubectl的服务器k8s-master01吗。现在我们需要在这样面执行命令。

```
[root@k8s-master01 kubernetes]# kubectl get nodes
```

```
[root@k8s-master01 kubernetes]# kubectl get nodes
NAME          STATUS    ROLES     AGE       VERSION
k8s-node01    Ready     <none>    1m        v1.9.0
k8s-node02    Ready     <none>    1m        v1.9.0
k8s-node03    Ready     <none>    1m        v1.9.0
```

七、K8s1.9生产环境高可用集群部署手册-node中DNS

7.1、下载kube-dns范本

```
[root@k8s-node01 ~]#  docker pull netonline/pause-amd64:3.0
[root@k8s-node01 ~]#  docker tag netonline/pause-amd64:3.0
gcr.io/google_containers/pause-amd64:3.0
[root@k8s-node01 ~]#  docker images
```

```
# kubedns
[root@k8s-node01 ~]# docker pull netonline/k8s-dns-kube-dns-amd64:1.14.8

# dnsmasq-nanny
[root@k8s-node01 ~]# docker pull netonline/k8s-dns-dnsmasq-nanny-amd64:1.14.8

# sidecar
[root@k8s-node01 ~]# docker pull netonline/k8s-dns-sidecar-amd64:1.14.8
```

```
在master01节点操作：
# https://github.com/kubernetes/kubernetes/tree/master/cluster/addons/dns
[root@k8s-master01 ~]# mkdir -p /usr/local/src/yaml/kubedns
[root@k8s-master01 ~]# cd /usr/local/src/yaml/kubedns
```

我们这里不用去下载，用下面的就可以了：

```
 [root@k8s-master01 setups]# vi kube-dns.yml
apiVersion: v1
kind: Service
metadata:
  name: kube-dns
  namespace: kube-system
  labels:
    k8s-app: kube-dns
    kubernetes.io/cluster-service: "true"
    addonmanager.kubernetes.io/mode: Reconcile
    kubernetes.io/name: "KubeDNS"
spec:
  selector:
    k8s-app: kube-dns
  clusterIP: 10.254.0.2
  ports:
  - name: dns
    port: 53
    protocol: UDP
  - name: dns-tcp
    port: 53
    protocol: TCP

---

apiVersion: v1
kind: ServiceAccount
metadata:
  name: kube-dns
```

```
  namespace: kube-system
  labels:
    kubernetes.io/cluster-service: "true"
    addonmanager.kubernetes.io/mode: Reconcile

---

apiVersion: v1
kind: ConfigMap
metadata:
  name: kube-dns
  namespace: kube-system
  labels:
    addonmanager.kubernetes.io/mode: EnsureExists

---

apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: kube-dns
  namespace: kube-system
  labels:
    k8s-app: kube-dns
    kubernetes.io/cluster-service: "true"
    addonmanager.kubernetes.io/mode: Reconcile
spec:
  # replicas: not specified here:
  # 1. In order to make Addon Manager do not reconcile this replicas parameter.
  # 2. Default is 1.
  # 3. Will be tuned in real time if DNS horizontal auto-scaling is turned on.
  strategy:
    rollingUpdate:
      maxSurge: 10%
      maxUnavailable: 0
  selector:
    matchLabels:
      k8s-app: kube-dns
  template:
    metadata:
      labels:
        k8s-app: kube-dns
      annotations:
        scheduler.alpha.kubernetes.io/critical-pod: ''
    spec:
      tolerations:
      - key: "CriticalAddonsOnly"
        operator: "Exists"
      volumes:
      - name: kube-dns-config
        configMap:
          name: kube-dns
          optional: true
      containers:
      - name: kubedns
        image: netonline/k8s-dns-kube-dns-amd64:1.14.8
        imagePullPolicy: IfNotPresent
        resources:
```

```
          # TODO: Set memory limits when we've profiled the container for large
          # clusters, then set request = limit to keep this container in
          # guaranteed class. Currently, this container falls into the
          # "burstable" category so the kubelet doesn't backoff from restarting
it.
          limits:
            memory: 170Mi
          requests:
            cpu: 100m
            memory: 70Mi
        livenessProbe:
          httpGet:
            path: /healthcheck/kubedns
            port: 10054
            scheme: HTTP
          initialDelaySeconds: 60
          timeoutSeconds: 5
          successThreshold: 1
          failureThreshold: 5
        readinessProbe:
          httpGet:
            path: /readiness
            port: 8081
            scheme: HTTP
          # we poll on pod startup for the Kubernetes master service and
          # only setup the /readiness HTTP server once that's available.
          initialDelaySeconds: 3
          timeoutSeconds: 5
        args:
        - --domain=cluster.local.
        - --dns-port=10053
        - --config-dir=/kube-dns-config
        - --v=2
        #__PILLAR__FEDERATIONS__DOMAIN__MAP__
        env:
        - name: PROMETHEUS_PORT
          value: "10055"
        ports:
        - containerPort: 10053
          name: dns-local
          protocol: UDP
        - containerPort: 10053
          name: dns-tcp-local
          protocol: TCP
        - containerPort: 10055
          name: metrics
          protocol: TCP
        volumeMounts:
        - name: kube-dns-config
          mountPath: /kube-dns-config
      - name: dnsmasq
        image: netonline/k8s-dns-dnsmasq-nanny-amd64:1.14.8
        imagePullPolicy: IfNotPresent
        livenessProbe:
          httpGet:
            path: /healthcheck/dnsmasq
            port: 10054
            scheme: HTTP
```

```
          initialDelaySeconds: 60
          timeoutSeconds: 5
          successThreshold: 1
          failureThreshold: 5
        args:
        - -v=2
        - -logtostderr
        - -configDir=/etc/k8s/dns/dnsmasq-nanny
        - -restartDnsmasq=true
        - --
        - -k
        - --cache-size=1000
        - --log-facility=-
        - --server=/cluster.local./127.0.0.1#10053
        - --server=/in-addr.arpa/127.0.0.1#10053
        - --server=/ip6.arpa/127.0.0.1#10053
        ports:
        - containerPort: 53
          name: dns
          protocol: UDP
        - containerPort: 53
          name: dns-tcp
          protocol: TCP
        # see: https://github.com/kubernetes/kubernetes/issues/29055 for details
        resources:
          requests:
            cpu: 150m
            memory: 20Mi
        volumeMounts:
        - name: kube-dns-config
          mountPath: /etc/k8s/dns/dnsmasq-nanny
      - name: sidecar
        image: netonline/k8s-dns-sidecar-amd64:1.14.8
        imagePullPolicy: IfNotPresent
        livenessProbe:
          httpGet:
            path: /metrics
            port: 10054
            scheme: HTTP
          initialDelaySeconds: 60
          timeoutSeconds: 5
          successThreshold: 1
          failureThreshold: 5
        args:
        - --v=2
        - --logtostderr
        - --
probe=kubedns,127.0.0.1:10053,kubernetes.default.svc.cluster.local.,5,A
        - --probe=dnsmasq,127.0.0.1:53,kubernetes.default.svc.cluster.local.,5,A
        ports:
        - containerPort: 10054
          name: metrics
          protocol: TCP
        resources:
          requests:
            memory: 20Mi
            cpu: 10m
      dnsPolicy: Default  # Don't use cluster DNS.
```

```
      serviceAccountName: kube-dns
```

关闭3**个**node的防火墙：

```
systemctl stop firewalld.service && systemctl disable firewalld.service
```

7.2、 执行apply命令 创建kubedns

在**k8s-master01**节点：

创建kubedns

```
[root@k8s-master01 setups]# kubectl apply -f kube-dns.yml

service "kube-dns" created

serviceaccount "kube-dns" created

configmap "kube-dns" created

deployment "kube-dns" created
```

7.4、 查看验证kube-dns Deployment&Service&Pod

查看kubedns pod

[root@k8s-master01 setups]# kubectl get pod --namespace=kube-system

NAME READY STATUS RESTARTS AGE

kube-dns-5c874ccb67-vqtvb 3/3 Running 0 29s

```
# kube-dns Pod 3个容器已”Ready”，服务，deployment等也正常启动
[root@kubenode1 kubedns]# kubectl get pod -n kube-system -o wide
[root@kubenode1 kubedns]# kubectl get service -n kube-system -o wide
[root@kubenode1 kubedns]# kubectl get deployment -n kube-system -o wide
```

验证kubedns：

说明: 创建个pod, 进入pod 查看/etc/resolv.conf的 nameserver是否是10.254.0.2

```
cd /setups

cat > httpd.yml << EOF

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

 name: httpd-deployment
```

```
spec:
 replicas: 1
 template:
  metadata:
   labels:
•    run: httpd
  spec:
   containers:
   \- name: httpd
•    image: daocloud.io/library/httpd
•    ports:
•    \- containerPort: 80
EOF
[root@k8s-master01 setups]# kubectl apply -f httpd.yml
deployment "httpd-deployment" created
[root@k8s-master01 setups]# kubectl get pod -o wide
NAME                 READY   STATUS  RESTARTS  AGE    IP       NODE
httpd-deployment-5c9bc776cb-x82hs  1/1    Running  0     34s    10.200.75.3
192.168.55.36
[root@k8s-master01 setups]# kubectl exec -ti httpd-deployment-5c9bc776cb-x82hs -
- /bin/bash
root@httpd-deployment-5c9bc776cb-x82hs:/usr/local/apache2# cat /etc/resolv.conf
```

八、K8s1.9生产环境高可用集群部署手册-dashboard

8.1、UI 组件 - Dashboard 部署

8.1.1、下载官方提供的 Dashboard 组件部署的 yaml 文件

在**node01****、node02、node03**拉取 kubernetes-dashboard-amd64:v1.10.1 镜像：

```
docker search kubernetes-dashboard-amd64
docker pull lizhenliang/kubernetes-dashboard-amd64:v1.10.1
```

重命名镜像：

```
docker tag docker.io/lizhenliang/kubernetes-dashboard-amd64:v1.10.1
k8s.gcr.io/kubernetes-dashboard-amd64:v1.10.1
```

在**master01****操作：**

在yaml文件中使用本地镜像：

我们这里自己创建并修改kubernetes-dashboard.yaml：

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

[root@k8s-master01 setups]# vi kubernetes-dashboard.yaml

\# Copyright 2017 The Kubernetes Authors.

\#

\# Licensed under the Apache License, Version 2.0 (the "License");

\# you may not use this file except in compliance with the License.

\# You may obtain a copy of the License at

\#

\#   http://www.apache.org/licenses/LICENSE-2.0

\#

\# Unless required by applicable law or agreed to in writing, software

\# distributed under the License is distributed on an "AS IS" BASIS,

\# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

\# See the License for the specific language governing permissions and

\# limitations under the License.



\# Configuration to deploy release version of the Dashboard UI compatible with

\# Kubernetes 1.8.

\#

\# Example usage: kubectl create -f <this_file>
\# ------------------- Dashboard Secret ------------------- #
apiVersion: v1

kind: Secret

metadata:

 labels:
```

```
  k8s-app: kubernetes-dashboard
 name: kubernetes-dashboard-certs
 namespace: kube-system
type: Opaque
\---
\# ------------------- Dashboard Service Account ------------------- #
apiVersion: v1
kind: ServiceAccount
metadata:
 labels:
  k8s-app: kubernetes-dashboard
 name: kubernetes-dashboard
 namespace: kube-system
\---
\# ------------------- Dashboard Role & Role Binding ------------------- #
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
 name: kubernetes-dashboard-minimal
 namespace: kube-system
rules:
 \# Allow Dashboard to create 'kubernetes-dashboard-key-holder' secret.
\- apiGroups: [""]
 resources: ["secrets"]
 verbs: ["create"]
 \# Allow Dashboard to create 'kubernetes-dashboard-settings' config map.
\- apiGroups: [""]
 resources: ["configmaps"]
 verbs: ["create"]
```

```
 \# Allow Dashboard to get, update and delete Dashboard exclusive secrets.

\- apiGroups: [""]

 resources: ["secrets"]

 resourceNames: ["kubernetes-dashboard-key-holder", "kubernetes-dashboard-
certs"]

 verbs: ["get", "update", "delete"]

 \# Allow Dashboard to get and update 'kubernetes-dashboard-settings' config
map.

\- apiGroups: [""]

 resources: ["configmaps"]

 resourceNames: ["kubernetes-dashboard-settings"]

 verbs: ["get", "update"]

 \# Allow Dashboard to get metrics from heapster.

\- apiGroups: [""]

 resources: ["services"]

 resourceNames: ["heapster"]

 verbs: ["proxy"]

\- apiGroups: [""]

 resources: ["services/proxy"]

 resourceNames: ["heapster", "http:heapster:", "https:heapster:"]

 verbs: ["get"]



\---

apiVersion: rbac.authorization.k8s.io/v1

kind: RoleBinding

metadata:

 name: kubernetes-dashboard-minimal

 namespace: kube-system

roleRef:

 apiGroup: rbac.authorization.k8s.io
```

```
 kind: Role
 name: kubernetes-dashboard-minimal
subjects:
\- kind: ServiceAccount
 name: kubernetes-dashboard
 namespace: kube-system
\---
\# ------------------- Dashboard Deployment ------------------- #
kind: Deployment
apiVersion: apps/v1beta2
metadata:
 labels:
  k8s-app: kubernetes-dashboard
 name: kubernetes-dashboard
 namespace: kube-system
spec:
 replicas: 1
 revisionHistoryLimit: 10
 selector:
  matchLabels:
   k8s-app: kubernetes-dashboard
 template:
  metadata:
   labels:
•    k8s-app: kubernetes-dashboard
  spec:
   containers:
   \- name: kubernetes-dashboard
•    image: k8s.gcr.io/kubernetes-dashboard-amd64:v1.10.1
```

```
•    imagePullPolicy: IfNotPresent
•    \#image:reg.qiniu.com/k8s/kubernetes-dashboard-amd64:v1.8.3
•    \#image:registry.cn-hangzhou.aliyuncs.com/google-containers/kubernetes-dashboard-amd64:v1.7.1
•    \#image: lizhenliang/kubernetes-dashboard-amd64:v1.10.1
•     \#image:registry.cn-hangzhou.aliyuncs.com/kubeapps/k8s-gcr-kubernetes-dashboard-amd64:v1.8.3
•    ports:
•    \- containerPort: 8443
•     protocol: TCP
•    args:
•     \- --auto-generate-certificates
•     \# Uncomment the following line to manually specify Kubernetes API server Host
•     \# If not specified, Dashboard will attempt to auto discover the API server and connect
•     \# to it. Uncomment only if the default does not work.
•     \# - --apiserver-host=http://my-address:port
•    volumeMounts:
•    \- name: kubernetes-dashboard-certs
•     mountPath: /certs
•     \# Create on-disk volume to store exec logs
•    \- mountPath: /tmp
•     name: tmp-volume
•    livenessProbe:
•     httpGet:
•      scheme: HTTPS
•      path: /
•      port: 8443
•     initialDelaySeconds: 30
•     timeoutSeconds: 30
```

```
   volumes:
   \- name: kubernetes-dashboard-certs
•    secret:
•     secretName: kubernetes-dashboard-certs
   \- name: tmp-volume
•    emptyDir: {}
   serviceAccountName: kubernetes-dashboard
   \# Comment the following tolerations if Dashboard must not be deployed on
master
   tolerations:
   \- key: node-role.kubernetes.io/master
•    effect: NoSchedule


\---
\# ------------------- Dashboard Service ------------------- #


kind: Service
apiVersion: v1
metadata:
 labels:
  k8s-app: kubernetes-dashboard
 name: kubernetes-dashboard
 namespace: kube-system
spec:
 type: NodePort
 ports:
  \- port: 443
   targetPort: 8443
   nodePort: 32096
 selector:
```

```
  k8s-app: kubernetes-dashboard
```

```
[root@k8s-master01 setups]# kubectl get pod -o wide --namespace=kube-system

[root@k8s-master01 setups]# **kubectl describe pod** **kubernetes-dashboard-
79946f5678-kptw7** **--namespace kube-system**
```

8.1.2、 yaml文件中的 Dashboard Service已修改

暴露服务使外部能够访问

添加 type: NodePort

```
\# lsof -i |grep 32096

\# netstat -ntlp
```

如果手动指定NodePort端口，必须确保端口未被占用。如下：

\

```
---

\# ------------------- Dashboard Service ------------------- #

kind: Service

apiVersion: v1

metadata:

 labels:

  k8s-app: kubernetes-dashboard

 name: kubernetes-dashboard

 namespace: kube-system

spec:

 type: NodePort

 ports:

  \- port: 443

   targetPort: 8443

•    nodePort: 32096

 selector:

  k8s-app: kubernetes-dashboar
```

关闭3**个**node的防火墙：

```
systemctl stop firewalld.service && systemctl disable firewalld.service
```

docker 从 1.13 版本开始，可能将 iptables FORWARD chain的默认策略设置为DROP，从而导致 ping 其它 Node 上的 Pod IP 失败，遇到这种情况时，需要手动设置策略为 ACCEPT：

```
# iptables -P FORWARD ACCEPT
```

并且把以下命令写入 /etc/rc.local 文件中，防止节点重启iptables FORWARD chain的默认策略又还原为DROP

```
/sbin/iptables -P FORWARD ACCEPT
```

有时发现将这句话写入了/etc/rc.local里面了，怎么没有生效？这是因为centos7等比较新的系统已经摒弃通过/etc/rc.local方式来执行开机脚本的方式。

那么我们只需要更改docker的启动服务脚本即可：

```
vim /etc/systemd/system/docker.service
```

在

```
[Service]
```

这项下面添加

```
ExecStartPost=/sbin/iptables -I FORWARD -s 0.0.0.0/0 -j ACCEPT
```

8.1.3、创建doshboard资源和查看资源并访问页面

1）创建资源

```
[root@k8s-master01 setups]# kubectl apply -f kubernetes-dashboard.yaml

secret "kubernetes-dashboard-certs" created

serviceaccount "kubernetes-dashboard" created

role "kubernetes-dashboard-minimal" created

rolebinding "kubernetes-dashboard-minimal" created

deployment "kubernetes-dashboard" created

service "kubernetes-dashboard" created
```

2）查看svc

```
[root@k8s-master01 setups]# kubectl get svc -o wide --namespace=kube-system
```

结果输出：

```
[root@k8s-master01 setups]# kubectl get services --all-namespaces

NAMESPACE   NAME          TYPE    CLUSTER-IP   EXTERNAL-IP  PORT(S)     AGE

default    kubernetes       ClusterIP  10.254.0.1   <none>    443/TCP     2d

kube-system  kube-dns        ClusterIP  10.254.0.2   <none>    53/UDP,53/TCP
13h

kube-system  kubernetes-dashboard  NodePort  10.254.163.47  <none>
443:32096/TCP  16h
```

```
可以看出：NodePort 32096 映射到 dashboard pod 443 端口；
```

3）查看pod

```
[root@k8s-master01 setups]# kubectl get pod -o wide --namespace=kube-system
```

结果输出：

```
[root@k8s-master01 setups]# kubectl get pods --all-namespaces -o wide

NAMESPACE   NAME                   READY   STATUS       RESTARTS  AGE    IP
NODE

kube-system  kube-dns-664fcd6f68-mbfql        3/3    Running      3     11h
172.30.71.2  k8s-node03

kube-system  kubernetes-dashboard-6f76b7547d-6hd9v  0/1    Running  16     10h
172.30.27.2  k8s-node01
```

查看分配的 NodePort

```
[root@k8s-master01 setups]# kubectl get deployment kubernetes-dashboard -n kube-
system
```

4）验证

web访问：

```
[root@k8s-master01 setups]#** **kubectl get svc --namespace kube-system**

*NAME          TYPE     CLUSTER-IP     EXTERNAL-IP   PORT(S)      AGE*

*kube-dns         ClusterIP   10.254.0.2         53/UDP,53/TCP   3h*

*kubernetes-dashboard   NodePort*   *10.254.204.176*   *     443:32096/TCP   2h*
```

访问HTTP非安全端口：

```
# curl http://192.168.10.55:8080 -k -I
```

```
[root@k8s-master01 kubernetes]# curl http://192.168.10.55:8080 -k -I
HTTP/1.1 200 OK
Audit-Id: 1c48de47-1be0-465c-af19-a4582f06532f
Content-Type: application/json
Date: Wed, 25 Dec 2019 15:30:56 GMT
```

访问HTTPS安全端口：

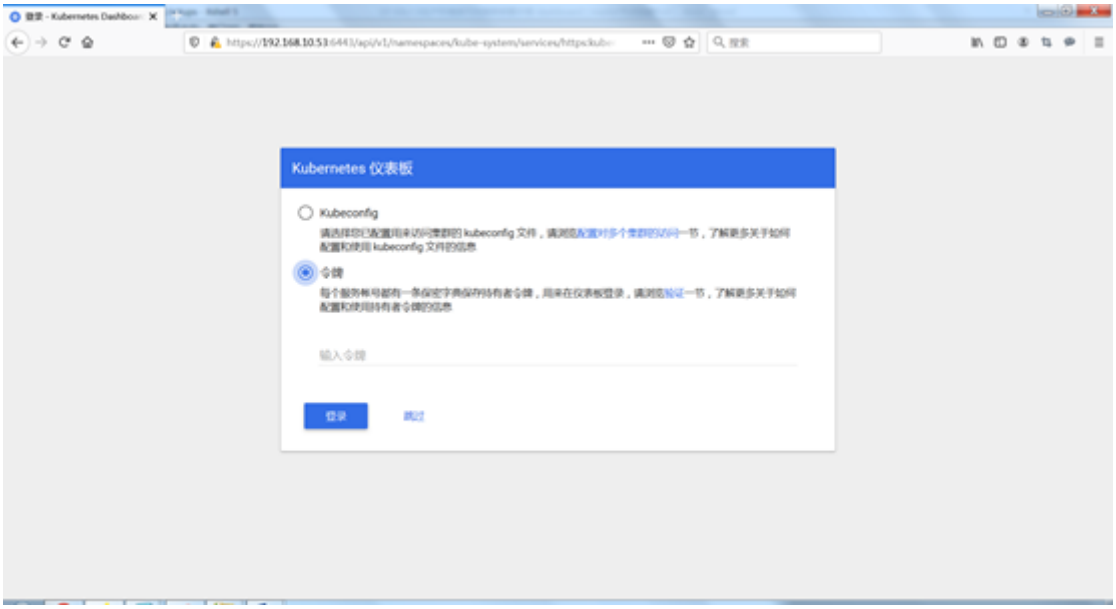
```
$ curl https://192.168.10.55:6443 -k -I
```

查看dashboard被k8s分配到了哪一台机器上：

```
[root@k8s-master01 setups]# kubectl get pods --all-namespaces -o wide
```

浏览器访问**http://node_IP:nodeport/****

https://192.168.10.57:32096



8.1.4、创建能够访问 Dashboard 的用户

在master01操作：

新建文件 account.yaml ， 内容如下：

```
cd /setups/
```

```
vi account.yaml

\# Create Service Account

apiVersion: v1

kind: ServiceAccount

metadata:

 name: admin-user

 namespace: kube-system

\---

\# Create ClusterRoleBinding

apiVersion: rbac.authorization.k8s.io/v1beta1

kind: ClusterRoleBinding

metadata:

 name: admin-user

roleRef:

 apiGroup: rbac.authorization.k8s.io

 kind: ClusterRole

 name: cluster-admin

subjects:

\- kind: ServiceAccount

 name: admin-user

 namespace: kube-system
[root@k8s-master01 setups]# kubectl apply -f account.yaml
```

8.1.5、获取登录 Dashboard 的令牌 （Token）

```
kubectl -n kube-system describe secret $(kubectl -n kube-system get secret | grep admin-user | awk '{print $1}')
```

输出如下

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

Name:     admin-user-token-pp7jb

Namespace:  kube-system
```

```
Labels:    <none>

Annotations: kubernetes.io/service-account.name=admin-user

•       kubernetes.io/service-account.uid=02b2a05d-1ace-11ea-968b-000c2949229d

Type: kubernetes.io/service-account-token

Data

====

ca.crt:   1359 bytes

namespace: 11 bytes

token:
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW5
OIiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJrdWJlLXN5c3RlbSIsImt
1YmVybmV0ZXMuaW8vc2VydmljZWFjY291bnQvc2VjcmV0Lm5hbWUiOiJhZG1pbi11c2VyLXRva2VuLXB
wN2piIiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9zZXJ2aWNlLWFjY291bnQubmFtZSI6ImF
kbWluLXVzZXIiLCJrdWJlcm5ldGVzLmlvL3NlcnZpY2VhY2NvdW50L3NlcnZpY2UtYWNjb3VudC51aWQ
iOiIwMmIyYTA1ZC0xYWNlLTExZWEtOTY4Yi0wMDBjMjk0OTIyOWQiLCJzdWIiOiJzeXN0ZW06c2Vydml
jZWFjY291bnQ6a3ViZS1zeXN0ZW06YWRtaW4tdXNlciJ9.e-
vRQMiO7b6HXuRxxJ8MoTVeGdLBLCwzYx2bbIRt48zDcv-Q7hCeutI3G7m_ijfHZnV-IQ6dOO9fRk48-
66goDGH4-I-WJnXXQuuzNtieBTcS2oWbcah4AK8goZP0NuiMAKFhys-
APofEGopLp2VMj8YqPllAbdE6XSfR5w2W8CT2YArwzyzgW-g1V16XT-H4QhgqdDqv-
oYdwtrqm88lu53ANtozIn4ZVhlJKmbPdMLcyToo0BZtf0bl2LFnmbfU8NyT2XxQqT95K2uHTbo0-
40VRYmP8G1VEZY24tNxPCYL2mHXpbY9Qp6ZSr5ewl6k_sTTerWDnGOygDwCZMUzA
```

8.1.6、 登录 Dashboard