Kubernetes二进制集群安装

本系列文档将介绍如何使用二进制部署`Kubernetes v1.14`集群的所有部署，而不是使用自动化部署(kubeadm)集群。在部署过程中，将详细列出各个组件启动参数，以及相关配置说明。在学习完本文档后，将理解k8s各个组件的交互原理，并且可以快速解决实际问题。

> 本文档适用于`Centos7.4`及以上版本，随着各个组件的更新，本文档提供了相关镜像的包，及时版本更新也不会影响文档的使用。 如果有文档相关问题可以直接在网站下面注册回复，或者点击右下角加群，我将在12小时内回复您。 并且建议您使用的环境及配置和我相同！

组件版本

- Kubernetes 1.14.2
- Docker 18.09 (docker使用官方的脚本安装，后期可能升级为新的版本，但是不影响)
- Etcd 3.3.13
- Flanneld 0.11.0

组件说明

kube-apiserver

- 使用节点本地Nginx 4层透明代理实现高可用 (也可以使用haproxy，只是起到代理apiserver的作用)
- 关闭非安全端口8080和匿名访问
- 使用安全端口6443接受https请求
- 严格的认知和授权策略 (x509、token、rbac)
- 开启bootstrap token认证，支持kubelet TLS bootstrapping；
- 使用https访问kubelet、etcd

kube-controller-manager

- 3节点高可用 (在k8s中，有些组件需要选举，所以使用奇数为集群高可用方案)
- 关闭非安全端口，使用10252接受https请求

- 使用kubeconfig访问apiserver的安全扣
- 使用approve kubelet证书签名请求(CSR)，证书过期后自动轮转
- 各controller使用自己的ServiceAccount访问apiserver

kube-scheduler

- 3节点高可用；
- 使用kubeconfig访问apiserver安全端口

kubelet

- 使用kubeadm动态创建bootstrap token
- 使用TLS bootstrap机制自动生成client和server证书，过期后自动轮转
- 在kubeletConfiguration类型的JSON文件配置主要参数
- 关闭只读端口，在安全端口10250接受https请求，对请求进行认真和授权，拒绝匿名访问和非授权访问
- 使用kubeconfig访问apiserver的安全端口

kube-proxy

- 使用kubeconfig访问apiserver的安全端口
- 在KubeProxyConfiguration类型JSON文件配置为主要参数
- 使用ipvs代理模式

集群插件

- DNS 使用功能、性能更好的coredns
- 网络 使用Flanneld 作为集群网络插件

一、初始化环境

集群机器

```
192.168.0.50 k8s-01
192.168.0.51 k8s-02
192.168.0.52 k8s-03
#node节点
192.168.0.53 k8s-04      #node节点只运行node，但是设置证书的时候要添加这个ip
```

本文档的所有etcd集群、master集群、worker节点均使用以上三台机器，并且初始化步骤需要在所有机器上执行命令。如果没有特殊命令，所有操作均在**192.168.0.50**上进行操作

> node节点后面会有操作，但是在初始化这步，是所有集群机器。包括node节点，我上面没有列出node节点

修改主机名

所有机器设置永久主机名

```
hostnamectl set-hostname k8s01  #所有机器按照要求修改
bash        #刷新主机名
```

接下来我们需要在所有机器上添加hosts解析

```
cat >> /etc/hosts <<EOF
192.168.0.50  k8s-01
192.168.0.51  k8s-02
192.168.0.52  k8s-03
192.168.0.53  k8s-04
EOF
```

设置免密
我们只在k8s-01上设置免密即可

```
wget -O /etc/yum.repos.d/epel.repo http://mirrors.aliyun.com/repo/epel-7.repo
curl -o /etc/yum.repos.d/CentOS-Base.repo http://mirrors.aliyun.com/repo/Centos-7.repo
yum install -y expect
#分发公钥
ssh-keygen -t rsa -P "" -f /root/.ssh/id_rsa
for i in k8s-01 k8s-02 k8s-03 k8s-04;do
expect -c "
spawn ssh-copy-id -i /root/.ssh/id_rsa.pub root@$i
        expect {
                \"*yes/no*\" {send \"yes\r\"; exp_continue}
                \"*password*\" {send \"123456\r\"; exp_continue}
                \"*Password*\" {send \"123456\r\";}
        } "
done
#我这里密码是123456  大家按照自己主机的密码进行修改就可以
```

更新PATH变量

本次的k8s软件包的目录全部存放在`/opt`下

```
[root@k8s01 ~]# echo 'PATH=/opt/k8s/bin:$PATH' >>/etc/profile
[root@k8s01 ~]# source  /etc/profile
[root@k8s01 ~]# env|grep PATH
PATH=/opt/k8s/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/bin
```

安装依赖包

在**每台**服务器上安装依赖包

```
yum install -y conntrack ntpdate ntp ipvsadm ipset jq iptables curl sysstat libseccomp wget
```

关闭防火墙 Linux 以及swap分区

```
systemctl stop firewalld
systemctl disable firewalld
iptables -F && iptables -X && iptables -F -t nat && iptables -X -t nat
iptables -P FORWARD ACCEPT
swapoff -a
sed -i '/ swap / s/^\(.*\)$/#\1/g' /etc/fstab
setenforce 0
sed -i 's/^SELINUX=.*/SELINUX=disabled/' /etc/selinux/config
```

#如果开启了swap分区，kubelet会启动失败(可以通过设置参数——-fail-swap-on设置为false)

升级内核

Docker overlay2需要使用kernel 4.x版本，所以我们需要升级内核

我这里的内核使用4.18.9

CentOS 7.x 系统自带的 3.10.x 内核存在一些 Bugs，导致运行的 Docker、Kubernetes 不稳定，例如：

```
高版本的 docker(1.13 以后) 启用了 3.10 kernel 实验支持的 kernel memory acco
```

```
unt 功能(无法关闭)，当节点压力大如频繁启动和停止容器时会导致 cgroup memory leak;
网络设备引用计数泄漏，会导致类似于报错："kernel:unregister_netdevice: waiting for eth0 to become free. Usage count = 1";
```

解决方案如下：

```
升级内核到 4.4.X 以上;
或者，手动编译内核，disable CONFIG_MEMCG_KMEM 特性;
或者，安装修复了该问题的 Docker 18.09.1 及以上的版本。但由于 kubelet 也会设置 kmem（它 vendor 了 runc），所以需要重新编译 kubelet 并指定 GOFLAGS="-tags=nokmem";
export Kernel_Version=4.18.9-1
wget  http://mirror.rc.usf.edu/compute_lock/elrepo/kernel/el7/x86_64/RPMS/kernel-ml{,-devel}-${Kernel_Version}.el7.elrepo.x86_64.rpm
yum localinstall -y kernel-ml*
#如果是手动下载内核rpm包，直接执行后面yum install -y kernel-ml*即可
```

修改内核启动顺序,默认启动的顺序应该为1,升级以后内核是往前面插入,为0（如果每次启动时需要手动选择哪个内核,该步骤可以省略）

```
grub2-set-default  0 && grub2-mkconfig -o /etc/grub2.cfg
```

使用下面命令看看确认下是否启动默认内核指向上面安装的内核

```
grubby --default-kernel
#这里的输出结果应该为我们升级后的内核信息
```

重启加载新内核 （升级完内核顺便update一下）

```
reboot
```

加载内核模块

首先我们要检查是否存在所需的内核模块

```
find  /lib/modules/`uname -r`/ -name "ip_vs_rr*"
find  /lib/modules/`uname -r`/ -name "br_netfilter*"
```

1.加载内核，加入开机启动（2选1即可）

```
cat > /etc/rc.local  << EOF
modprobe ip_vs_rr
modprobe br_netfilter
EOF
```

2.使用systemd-modules-load加载内核模块

```
cat > /etc/modules-load.d/ipvs.conf << EOF
 ip_vs_rr
 br_netfilter
EOF
systemctl enable --now systemd-modules-load.service
```

验证模块是否加载成功

```
lsmod |egrep " ip_vs_rr|br_netfilter"
为什么要使用IPVS,从k8s的1.8版本开始，kube-proxy引入了IPVS模式，IPVS模式与iptables同样基于Netfilter，但是采用的hash表，因此当service数量达到一定规模时，hash查表的速度优势就会显现出来，从而提高service的服务性能。
ipvs依赖于nf_conntrack_ipv4内核模块,4.19包括之后内核里改名为nf_conntrack,1.13.1之前的kube-proxy的代码里没有加判断一直用的nf_conntrack_ipv4,好像是1.13.1后的kube-proxy代码里增加了判断,我测试了是会去load nf_conntrack使用ipvs正常
```

优化内核参数

```
cat > kubernetes.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 # 禁止使用 swap 空间，只有当系统 OOM 时才允许使用它
vm.overcommit_memory=1 # 不检查物理内存是否够用
vm.panic_on_oom=0 # 开启 OOM
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
cp kubernetes.conf  /etc/sysctl.d/kubernetes.conf
sysctl -p /etc/sysctl.d/kubernetes.conf
```

需要关闭`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

设置系统时区

```
timedatectl set-timezone Asia/Shanghai
 #将当前的 UTC 时间写入硬件时钟
timedatectl set-local-rtc 0
 #重启依赖于系统时间的服务
systemctl restart rsyslog
systemctl restart crond
```

创建相关目录

```
mkdir -p  /opt/k8s/{bin,work} /etc/{kubernetes,etcd}/cert
```

#在所有节点上执行，因为flanneld是在所有节点运行的

设置分发脚本参数

后续所有的使用环境变量都定义在environment.sh中，需要根据个人机器及网络环境修改。并且需要拷贝到所有节点的/opt/k8s/bin目录下

```bash
#!/usr/bin/bash
# 生成 EncryptionConfig 所需的加密 key
export ENCRYPTION_KEY=$(head -c 32 /dev/urandom | base64)
# 集群各机器 IP 数组
export NODE_IPS=( 192.168.0.50 192.168.0.51 192.168.0.52 192.168.0.53 )
# 集群各 IP 对应的主机名数组
export NODE_NAMES=(k8s-01 k8s-02 k8s-03 k8s-04)
# 集群MASTER机器 IP 数组
export MASTER_IPS=(192.168.0.50 192.168.0.51 192.168.0.52 )
# 集群所有的master Ip对应的主机
export MASTER_NAMES=(k8s-01 k8s-02 k8s-03)
# etcd 集群服务地址列表
export ETCD_ENDPOINTS="https://192.168.0.50:2379,https://192.168.0.51:2379,https://192.168.0.52:2379"
# etcd 集群间通信的 IP 和端口
export ETCD_NODES="k8s-01=https://192.168.0.50:2380,k8s-02=https://192.168.0.51:2380,k8s-03=https://192.168.0.52:2380"
# etcd 集群所有node ip
export ETCD_IPS=(192.168.0.50 192.168.0.51 192.168.0.52 192.168.0.53)
# kube-apiserver 的反向代理(kube-nginx)地址端口
export KUBE_APISERVER="https://192.168.0.54:8443"
# 节点间互联网络接口名称
export IFACE="eth0"
# etcd 数据目录
export ETCD_DATA_DIR="/data/k8s/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="/data/k8s/etcd/wal"
# k8s 各组件数据目录
export K8S_DIR="/data/k8s/k8s"
# docker 数据目录
#export DOCKER_DIR="/data/k8s/docker"
## 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="41f7e4ba8b7be874fcff18bf5cf41a7c"
# 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="1024-32767"
# flanneld 网络配置前缀
export FLANNEL_ETCD_PREFIX="/kubernetes/network"
# kubernetes 服务 IP (一般是 SERVICE_CIDR 中第一个IP)
export CLUSTER_KUBERNETES_SVC_IP="10.254.0.1"
# 集群 DNS 服务 IP (从 SERVICE_CIDR 中预分配)
export CLUSTER_DNS_SVC_IP="10.254.0.2"
# 集群 DNS 域名 (末尾不带点号)
export CLUSTER_DNS_DOMAIN="cluster.local"
# 将二进制目录 /opt/k8s/bin 加到 PATH 中
export PATH=/opt/k8s/bin:$PATH
```

请根据IP进行修改

分发环境变量脚本

```
source environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    scp environment.sh root@${node_ip}:/opt/k8s/bin/
    ssh root@${node_ip} "chmod +x /opt/k8s/bin/* "
done
```

二、k8s集群部署

创建CA证书和秘钥

为确保安全，kubernetes各个组件需要使用x509证书对通信进行加密和认证

CA(Certificate Authority)是自签名的根证书，用来签名后续创建的其他证书。本文章使用CloudFlare的PKI工具cfssl创建所有证书。

> **注意：** 如果没有特殊指明，本文档的所有操作均在k8s-01节点执行，远程分发到其他节点

安装cfssl工具集

```
mkdir -p /opt/k8s/cert && cd /opt/k8s
wget https://pkg.cfssl.org/R1.2/cfssl_linux-amd64
mv cfssl_linux-amd64 /opt/k8s/bin/cfssl
wget https://pkg.cfssl.org/R1.2/cfssljson_linux-amd64
mv cfssljson_linux-amd64 /opt/k8s/bin/cfssljson
wget https://pkg.cfssl.org/R1.2/cfssl-certinfo_linux-amd64
mv cfssl-certinfo_linux-amd64 /opt/k8s/bin/cfssl-certinfo
chmod +x /opt/k8s/bin/*
export PATH=/opt/k8s/bin:$PATH
```

创建根证书 (CA)

> 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是由它签名

创建配置文件

> 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)

```
cd /opt/k8s/work
cat > ca-config.json <<EOF
{
  "signing": {
    "default": {
      "expiry": "87600h"
    },
    "profiles": {
      "kubernetes": {
        "usages": [
            "signing",
            "key encipherment",
            "server auth",
            "client auth"
        ],
        "expiry": "87600h"
      }
    }
  }
}
EOF

```

```
######################
signing 表示该证书可用于签名其它证书，生成的ca.pem证书找中CA=TRUE
server auth 表示client可以用该证书对server提供的证书进行验证
client auth 表示server可以用该证书对client提供的证书进行验证
```

创建证书签名请求文件

```
cd /opt/k8s/work
cat > ca-csr.json <<EOF
{
  "CN": "kubernetes",
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "CN",
      "ST": "BeiJing",
      "L": "BeiJing",
      "O": "k8s",
      "OU": "xuyuntech"
    }
  ],
  "ca": {
    "expiry": "876000h"
 }
}
EOF

#######################
CN CommonName,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和所属组(Group)
kube-apiserver将提取的User、Group作为RBAC授权的用户和标识
```

生成CA证书和私钥

```
cd /opt/k8s/work
cfssl gencert -initca ca-csr.json | cfssljson -bare ca
ls ca*
```

分发证书

#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/cert目录下

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh root@${node_ip} "mkdir -p /etc/kubernetes/cert"
    scp ca*.pem ca-config.json root@${node_ip}:/etc/kubernetes/cert
  done
```

部署kubectl命令行工具

kubectl默认从`~/.kube/config`读取kube-apiserver地址和认证信息。kube/config只需要部署一次，生成的kubeconfig文件是通用的

下载和解压kubectl

```
cd /opt/k8s/work
wget http://down.i4t.com/k8s1.14/kubernetes-client-linux-amd64.tar.gz
tar -xzvf kubernetes-client-linux-amd64.tar.gz
```

分发所有使用kubectl节点

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    scp kubernetes/client/bin/kubectl root@${node_ip}:/opt/k8s/bin/
    ssh root@${node_ip} "chmod +x /opt/k8s/bin/*"
  done
```

创建admin证书和私钥

kubectl与apiserver https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 /opt/k8s/work
cat > admin-csr.json <<EOF
{
  "CN": "admin",
  "hosts": [],
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "CN",
      "ST": "BeiJing",
      "L": "BeiJing",
      "O": "system:masters",
      "OU": "xuyuntech"
    }
  ]
}
EOF

###################
● O 为system:masters，kube-apiserver收到该证书后将请求的Group设置为system:masters
● 预定的ClusterRoleBinding cluster-admin将Group system:masters与Role cluster-admin绑定，该Role授予API的权限
● 该证书只有被kubectl当做client证书使用，所以hosts字段为空
```

生成证书和私钥

```
cd /opt/k8s/work
cfssl gencert -ca=/opt/k8s/work/ca.pem \
  -ca-key=/opt/k8s/work/ca-key.pem \
  -config=/opt/k8s/work/ca-config.json \
  -profile=kubernetes admin-csr.json | cfssljson -bare admin
ls admin*
```

创建kubeconfig文件

kubeconfig为kubectl的配置文件，包含访问apiserver的所有信息，如apiserver地址、CA证书和自身使用的证书

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
# 设置集群参数
kubectl config set-cluster kubernetes \
  --certificate-authority=/opt/k8s/work/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=/opt/k8s/work/admin.pem \
  --client-key=/opt/k8s/work/admin-key.pem \
  --embed-certs=true \
  --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

################
--certificate-authority 验证kube-apiserver证书的根证书
--client-certificate、--client-key 刚生成的admin证书和私钥，连接kube-apiserver时使用
--embed-certs=true 将ca.pem和admin.pem证书嵌入到生成的kubectl.kubeconfig
```

```
    文件中（如果不加入，写入的是证书文件路径，后续拷贝kubeconfig到其它机器时，还需要单
    独拷贝证书）
```

分发kubeconfig文件
分发到所有使用kubectl命令的节点

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh root@${node_ip} "mkdir -p ~/.kube"
    scp kubectl.kubeconfig root@${node_ip}:~/.kube/config
  done

#保存文件名为~/.kube/config
```

部署ETCD集群
这里使用的ETCD为三节点高可用集群，步骤如下

- 下载和分发etcd二进制文件
- 创建etcd集群各节点的x509证书，用于加密客户端(如kubectl)与etcd集群、etcd集群之间的数据流
- 创建etcd的system unit文件，配置服务参数
- 检查集群工作状态
- etcd集群各节点的名称和IP如下
- k8s-01 192.168.0.50
- k8s-02 192.168.0.51
- k8s-03 192.168.0.52
- 注意: 没有特殊说明都在k8s-01节点操作

下载和分发etcd二进制文件

```
cd /opt/k8s/work
wget http://down.i4t.com/k8s1.14/etcd-v3.3.13-linux-amd64.tar.gz
tar -xvf etcd-v3.3.13-linux-amd64.tar.gz
```

分发二进制文件到集群节点

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
for node_ip in ${ETCD_IPS[@]}
  do
    echo ">>> ${node_ip}"
    scp etcd-v3.3.13-linux-amd64/etcd* root@${node_ip}:/opt/k8s/bin
    ssh root@${node_ip} "chmod +x /opt/k8s/bin/*"
  done
```

创建etcd证书和私钥

```
cd /opt/k8s/work
cat > etcd-csr.json <<EOF
{
  "CN": "etcd",
  "hosts": [
    "127.0.0.1",
    "192.168.0.50",
    "192.168.0.51",
    "192.168.0.52"
  ],
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "CN",
      "ST": "BeiJing",
      "L": "BeiJing",
      "O": "k8s",
      "OU": "xuyuntech"
    }
  ]
}
EOF
#host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3个节点都添加其中
```

生成证书和私钥

```
cd /opt/k8s/work
cfssl gencert -ca=/opt/k8s/work/ca.pem \
    -ca-key=/opt/k8s/work/ca-key.pem \
    -config=/opt/k8s/work/ca-config.json \
    -profile=kubernetes etcd-csr.json | cfssljson -bare etcd
ls etcd*pem
```

分发证书和私钥到etcd各个节点

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
for node_ip in ${ETCD_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh root@${node_ip} "mkdir -p /etc/etcd/cert"
    scp etcd*.pem root@${node_ip}:/etc/etcd/cert/
  done
```

创建etcd的启动文件（这里将配置文件也存放在启动文件里）

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
cat > etcd.service.template <<EOF
[Unit]
Description=Etcd Server
After=network.target
After=network-online.target
Wants=network-online.target
Documentation=https://github.com/coreos
[Service]
Type=notify
WorkingDirectory=${ETCD_DATA_DIR}
ExecStart=/opt/k8s/bin/etcd \\
  --data-dir=${ETCD_DATA_DIR} \\
  --wal-dir=${ETCD_WAL_DIR} \\
  --name=##NODE_NAME## \\
  --cert-file=/etc/etcd/cert/etcd.pem \\
  --key-file=/etc/etcd/cert/etcd-key.pem \\
  --trusted-ca-file=/etc/kubernetes/cert/ca.pem \\
  --peer-cert-file=/etc/etcd/cert/etcd.pem \\
  --peer-key-file=/etc/etcd/cert/etcd-key.pem \\
  --peer-trusted-ca-file=/etc/kubernetes/cert/ca.pem \\
  --peer-client-cert-auth \\
  --client-cert-auth \\
  --listen-peer-urls=https://##NODE_IP##:2380 \\
  --initial-advertise-peer-urls=https://##NODE_IP##:2380 \\
  --listen-client-urls=https://##NODE_IP##:2379,http://127.0.0.1:2379 \\
  --advertise-client-urls=https://##NODE_IP##:2379 \\
  --initial-cluster-token=etcd-cluster-0 \\
  --initial-cluster=${ETCD_NODES} \\
  --initial-cluster-state=new \\
  --auto-compaction-mode=periodic \\
  --auto-compaction-retention=1 \\
  --max-request-bytes=33554432 \\
  --quota-backend-bytes=6442450944 \\
  --heartbeat-interval=250 \\
  --election-timeout=2000
Restart=on-failure
RestartSec=5
LimitNOFILE=65536
[Install]
WantedBy=multi-user.target
EOF
```

配置说明（此处不需要修改任何配置）

- WorkDirectory、—data-dir 指定etcd工作目录和数据存储为${ETCD_DATA_DIR},需要在启动前创建这个目录（后面跟着我操作就可以，会有创建步骤）
- —wal-dir 指定wal目录，为了提高性能，一般使用SSD和—data-dir不同的盘
- —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

为各个节点分发启动文件

```
#分发会将配置文件中的#替换成ip
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
for (( i=0; i < 3; i++ ))
  do
    sed -e "s/##NODE_NAME##/${MASTER_NAMES[i]}/" -e "s/##NODE_IP##/${ETCD_IPS[i]}/" etcd.service.template > etcd-${ETCD_IPS[i]}.service
  done
ls *.service
#NODE_NAMES 和 NODE_IPS 为相同长度的 bash 数组，分别为节点名称和对应的 IP；
```

分发生成的etcd启动文件到对应的服务器

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
for node_ip in ${MASTER_IPS[@]}
  do
    echo ">>> ${node_ip}"
    scp etcd-${node_ip}.service root@${node_ip}:/etc/systemd/system/etcd.service
  done
```

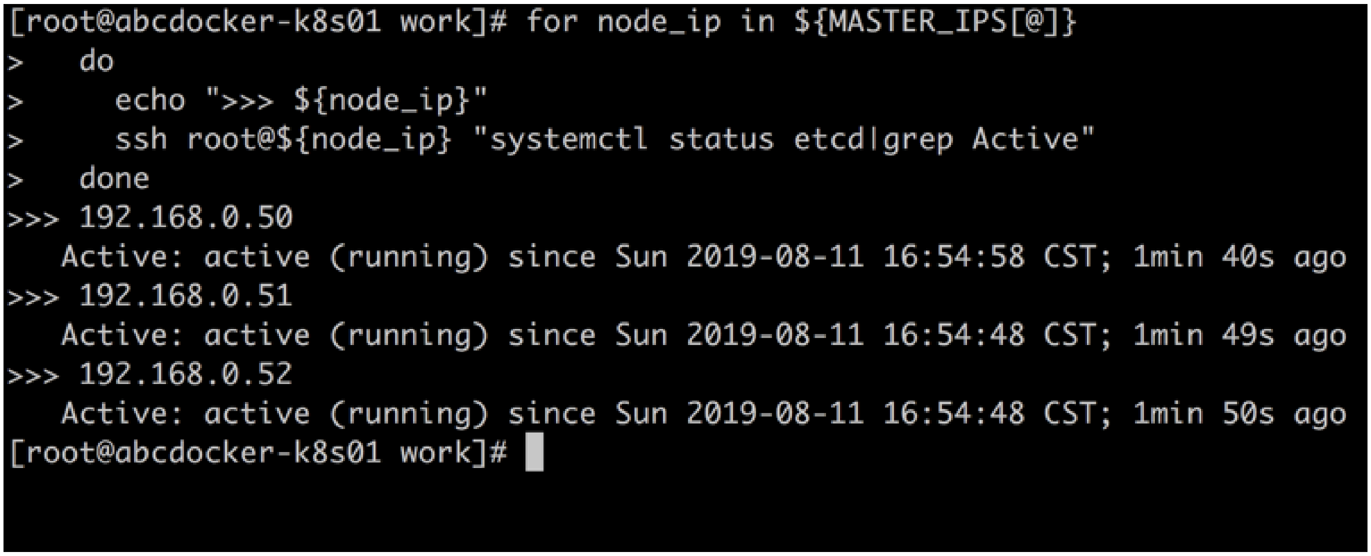
重命名etcd启动文件并启动etcd服务
etcd首次进程启动会等待其他节点加入etcd集群，执行启动命令会卡顿一会，为正常现象

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
for node_ip in ${MASTER_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh root@${node_ip} "mkdir -p ${ETCD_DATA_DIR} ${ETCD_WAL_DIR}"
    ssh root@${node_ip} "systemctl daemon-reload && systemctl enable etcd && systemctl restart etcd " &
  done
  #这里我们创建了etcd的工作目录
```

检查启动结果

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
for node_ip in ${MASTER_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh root@${node_ip} "systemctl status etcd|grep Active"
  done
```

正常状态



```
[root@abcdocker-k8s01 work]# for node_ip in ${MASTER_IPS[@]}
>   do
>     echo ">>> ${node_ip}"
>     ssh root@${node_ip} "systemctl status etcd|grep Active"
>   done
>>> 192.168.0.50
   Active: active (running) since Sun 2019-08-11 16:54:58 CST; 1min 40s ago
>>> 192.168.0.51
   Active: active (running) since Sun 2019-08-11 16:54:48 CST; 1min 49s ago
>>> 192.168.0.52
   Active: active (running) since Sun 2019-08-11 16:54:48 CST; 1min 50s ago
[root@abcdocker-k8s01 work]# 
```

如果etcd集群状态不是active (running)，请使用下面命令查看etcd日志

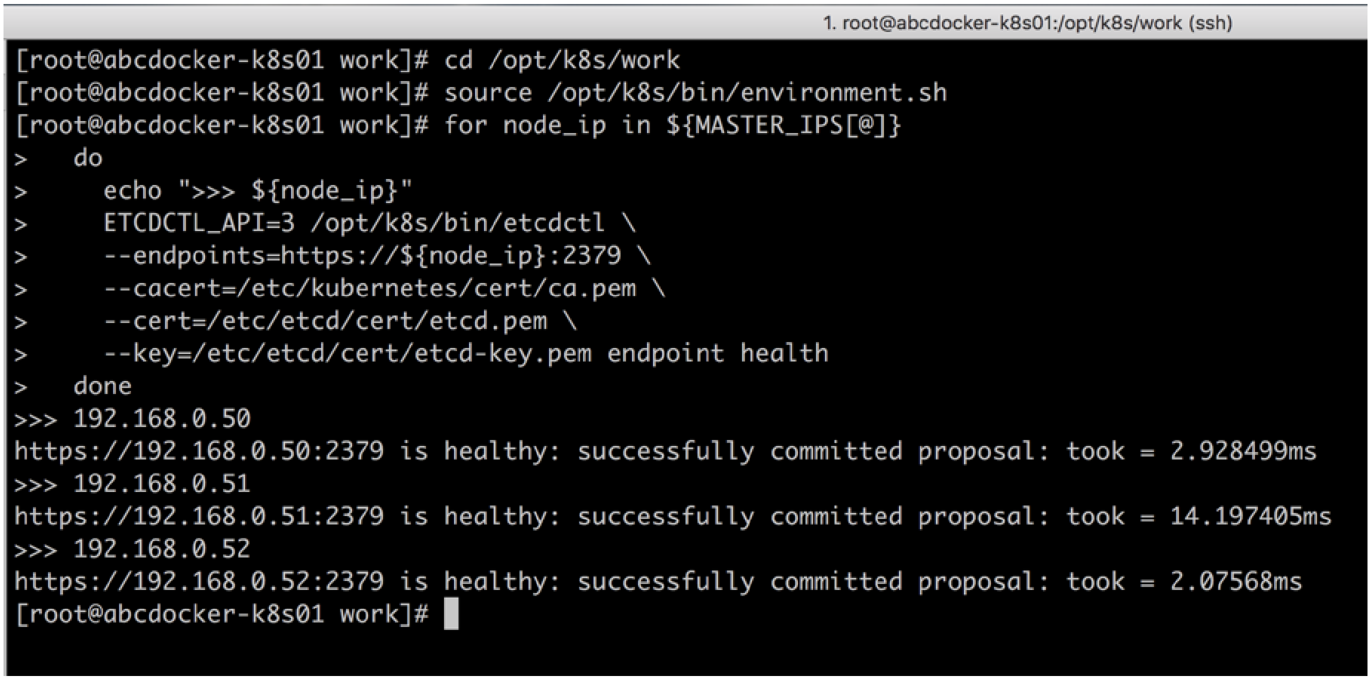
```
journalctl -fu etcd
```

验证ETCD集群状态

不是完etcd集群后，在任一etcd节点执行下命令

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
for node_ip in ${MASTER_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ETCDCTL_API=3 /opt/k8s/bin/etcdctl \
    --endpoints=https://${node_ip}:2379 \
    --cacert=/etc/kubernetes/cert/ca.pem \
    --cert=/etc/etcd/cert/etcd.pem \
    --key=/etc/etcd/cert/etcd-key.pem endpoint health
  done
```

正常状态如下



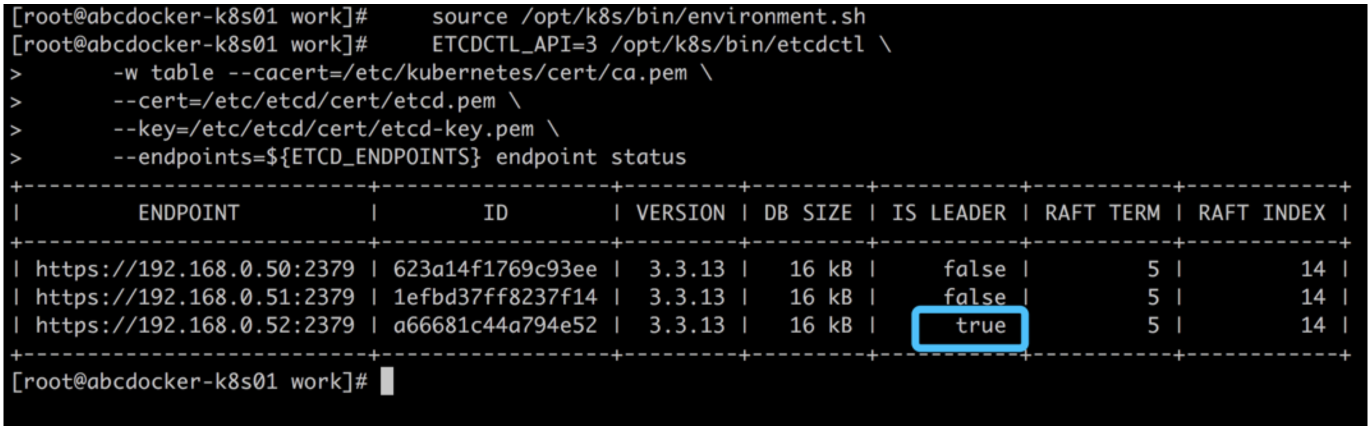
1. root@abcdocker-k8s01:/opt/k8s/work (ssh)

```
[root@abcdocker-k8s01 work]# cd /opt/k8s/work
[root@abcdocker-k8s01 work]# source /opt/k8s/bin/environment.sh
[root@abcdocker-k8s01 work]# for node_ip in ${MASTER_IPS[@]}
>   do
>     echo ">>> ${node_ip}"
>     ETCDCTL_API=3 /opt/k8s/bin/etcdctl \
>     --endpoints=https://${node_ip}:2379 \
>     --cacert=/etc/kubernetes/cert/ca.pem \
>     --cert=/etc/etcd/cert/etcd.pem \
>     --key=/etc/etcd/cert/etcd-key.pem endpoint health
>   done
>>> 192.168.0.50
https://192.168.0.50:2379 is healthy: successfully committed proposal: took = 2.928499ms
>>> 192.168.0.51
https://192.168.0.51:2379 is healthy: successfully committed proposal: took = 14.197405ms
>>> 192.168.0.52
https://192.168.0.52:2379 is healthy: successfully committed proposal: took = 2.07568ms
[root@abcdocker-k8s01 work]#
```

我们还可以通过下面命令查看当前etcd集群leader

```
source /opt/k8s/bin/environment.sh
ETCDCTL_API=3 /opt/k8s/bin/etcdctl \
  -w table --cacert=/etc/kubernetes/cert/ca.pem \
  --cert=/etc/etcd/cert/etcd.pem \
  --key=/etc/etcd/cert/etcd-key.pem \
  --endpoints=${ETCD_ENDPOINTS} endpoint status
```

正常状态如下



```
[root@abcdocker-k8s01 work]#     source /opt/k8s/bin/environment.sh
[root@abcdocker-k8s01 work]#     ETCDCTL_API=3 /opt/k8s/bin/etcdctl \
>       -w table --cacert=/etc/kubernetes/cert/ca.pem \
>       --cert=/etc/etcd/cert/etcd.pem \
>       --key=/etc/etcd/cert/etcd-key.pem \
>       --endpoints=${ETCD_ENDPOINTS} endpoint status
+---------------------------+------------------+---------+---------+-----------+-----------+------------+
|         ENDPOINT          |        ID        | VERSION | DB SIZE | IS LEADER | RAFT TERM | RAFT INDEX |
+---------------------------+------------------+---------+---------+-----------+-----------+------------+
| https://192.168.0.50:2379 | 623a14f1769c93ee |  3.3.13 |   16 kB |     false |         5 |         14 |
| https://192.168.0.51:2379 | 1efbd37ff8237f14 |  3.3.13 |   16 kB |     false |         5 |         14 |
| https://192.168.0.52:2379 | a66681c44a794e52 |  3.3.13 |   16 kB |      true |         5 |         14 |
+---------------------------+------------------+---------+---------+-----------+-----------+------------+
[root@abcdocker-k8s01 work]#
```

部署Flannel网络

Kubernetes要求集群内各个节点(包括master)能通过Pod网段互联互通，Flannel使用vxlan技术为各个节点创建一个互通的Pod网络，使用的端口为8472.第一次启动时，从etcd获取配置的Pod网络，为本节点分配一个未使用的地址段，然后创建flannel.1网络接口(也可能是其它名称)flannel将分配给自己的Pod网段信息写入`/run/flannel/docker`文件，docker后续使用这个文件中的环境变量设置Docker0网桥，从而从这个地址段为本节点的所有Pod容器分配IP
下载分发flanneld二进制文件（本次flanneld不使用Pod运行）

```
cd /opt/k8s/work
mkdir flannel
wget http://down.i4t.com/k8s1.14/flannel-v0.11.0-linux-amd64.tar.gz
tar -xzvf flannel-v0.11.0-linux-amd64.tar.gz -C flannel
```

分发二进制文件到所有集群的节点

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    scp flannel/{flanneld,mk-docker-opts.sh} root@${node_ip}:/opt/k8s/bin/
    ssh root@${node_ip} "chmod +x /opt/k8s/bin/*"
  done
```

创建Flannel证书和私钥
flanneld从etcd集群存取网段分配信息，而etcd集群开启了双向x509证书认证，所以需要为flannel生成证书和私钥
创建证书签名请求

```
cd /opt/k8s/work
cat > flanneld-csr.json <<EOF
{
  "CN": "flanneld",
  "hosts": [],
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "CN",
      "ST": "BeiJing",
      "L": "BeiJing",
      "O": "k8s",
      "OU": "xuyuntech"
    }
  ]
}
EOF
```

生成证书和私钥

```
cfssl gencert -ca=/opt/k8s/work/ca.pem \
  -ca-key=/opt/k8s/work/ca-key.pem \
  -config=/opt/k8s/work/ca-config.json \
  -profile=kubernetes flanneld-csr.json | cfssljson -bare flanneld
ls flanneld*pem
```

将生成的证书和私钥分发到所有节点

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh root@${node_ip} "mkdir -p /etc/flanneld/cert"
    scp flanneld*.pem root@${node_ip}:/etc/flanneld/cert
  done
```

向etcd写入Pod网段信息

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
etcdctl \
  --endpoints=${ETCD_ENDPOINTS} \
  --ca-file=/opt/k8s/work/ca.pem \
  --cert-file=/opt/k8s/work/flanneld.pem \
  --key-file=/opt/k8s/work/flanneld-key.pem \
  mk ${FLANNEL_ETCD_PREFIX}/config '{"Network":"'${CLUSTER_CIDR}'", "SubnetLen": 21, "Backend": {"Type": "vxlan"}}'
```

注意:
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参数一致
创建flanneld的启动文件

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
cat > flanneld.service << EOF
[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
ExecStart=/opt/k8s/bin/flanneld \\
  -etcd-cafile=/etc/kubernetes/cert/ca.pem \\
  -etcd-certfile=/etc/flanneld/cert/flanneld.pem \\
  -etcd-keyfile=/etc/flanneld/cert/flanneld-key.pem \\
  -etcd-endpoints=${ETCD_ENDPOINTS} \\
  -etcd-prefix=${FLANNEL_ETCD_PREFIX} \\
  -iface=${IFACE} \\
  -ip-masq
ExecStartPost=/opt/k8s/bin/mk-docker-opts.sh -k DOCKER_NETWORK_OPTIONS -d /run/flannel/docker
Restart=always
RestartSec=5
StartLimitInterval=0
[Install]
WantedBy=multi-user.target
```

```
RequiredBy=docker.service
EOF
```

- 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。

分发启动文件到所有节点

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    scp flanneld.service root@${node_ip}:/etc/systemd/system/
  done
```

启动flanneld服务

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh root@${node_ip} "systemctl daemon-reload && systemctl enable flanneld && systemctl restart flanneld"
  done
```

检查启动结果

```
source /opt/k8s/bin/environment.sh
```

```
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh root@${node_ip} "systemctl status flanneld|grep Active"
  done
```

检查分配给flanneld的Pod网段信息

```
source /opt/k8s/bin/environment.sh
etcdctl \
  --endpoints=${ETCD_ENDPOINTS} \
  --ca-file=/etc/kubernetes/cert/ca.pem \
  --cert-file=/etc/flanneld/cert/flanneld.pem \
  --key-file=/etc/flanneld/cert/flanneld-key.pem \
  get ${FLANNEL_ETCD_PREFIX}/config
```

查看已分配的Pod子网网段列表

```
source /opt/k8s/bin/environment.sh
etcdctl \
  --endpoints=${ETCD_ENDPOINTS} \
  --ca-file=/etc/kubernetes/cert/ca.pem \
  --cert-file=/etc/flanneld/cert/flanneld.pem \
  --key-file=/etc/flanneld/cert/flanneld-key.pem \
  ls ${FLANNEL_ETCD_PREFIX}/subnets
```

查看某Pod网段对应节点IP和flannel接口地址

```
source /opt/k8s/bin/environment.sh
etcdctl \
  --endpoints=${ETCD_ENDPOINTS} \
  --ca-file=/etc/kubernetes/cert/ca.pem \
  --cert-file=/etc/flanneld/cert/flanneld.pem \
  --key-file=/etc/flanneld/cert/flanneld-key.pem \
  get ${FLANNEL_ETCD_PREFIX}/subnets/172.30.16.0-21

  #后面节点IP需要根据我们查出来的地址进行修改
```

查看节点flannel网络信息

```
ip addr show
```

flannel.1网卡的地址为分配的pod自网段的第一个IP (.0)，且是/32的地址

```
ip addr show|grep flannel.1
```

到其它节点 Pod 网段请求都被转发到 flannel.1 网卡；
flanneld 根据 etcd 中子网段的信息，如 `${FLANNEL_ETCD_PREFIX}/subnets/172.30.80.0-21`，来决定进请求发送给哪个节点的互联 IP；
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)：

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh ${node_ip} "/usr/sbin/ip addr show flannel.1|grep -w inet"
  done
```

---

kube-apiserver 高可用

- 使用Nginx 4层透明代理功能实现k8s节点(master节点和worker节点)高可用访问kube-apiserver的步骤
- 控制节点的kube-controller-manager、kube-scheduler是多实例部署，所以只要一个实例正常，就可以保证集群高可用
- 集群内的Pod使用k8s服务域名kubernetes访问kube-apiserver，kube-dns会自动解析多个kube-apiserver节点的IP，所以也是高可用的
- 在每个Nginx进程，后端对接多个apiserver实例，Nginx对他们做健康检查和负载均衡
- kubelet、kube-proxy、controller-manager、schedule通过本地nginx (监听我们vip 192.158.0.54)访问kube-apiserver，从而实现kube-apiserver高可用

下载编译nginx (k8s-01安装就可以，后面有拷贝步骤)

```
cd /opt/k8s/work
wget http://down.i4t.com/k8s1.14/nginx-1.15.3.tar.gz
```

```
tar -xzvf nginx-1.15.3.tar.gz
#编译
cd /opt/k8s/work/nginx-1.15.3
mkdir nginx-prefix
./configure --with-stream --without-http --prefix=$(pwd)/nginx-prefix --without-http_uwsgi_module
make && make install
#############
--without-http_scgi_module --without-http_fastcgi_module
--with-stream：开启 4 层透明转发(TCP Proxy)功能；
--without-xxx：关闭所有其他功能，这样生成的动态链接二进制程序依赖最小；
```

查看 nginx 动态链接的库：

```
ldd ./nginx-prefix/sbin/nginx
```

由于只开启了 4 层透明转发功能，所以除了依赖 libc 等操作系统核心 lib 库外，没有对其它 lib 的依赖(如 libz、libssl 等)，这样可以方便部署到各版本操作系统中
创建目录结构

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    mkdir -p /opt/k8s/kube-nginx/{conf,logs,sbin}
  done
```

拷贝二进制程序到其他主机（有报错执行2遍就可以）

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    scp /opt/k8s/work/nginx-1.15.3/nginx-prefix/sbin/nginx  root@${node_ip}:/opt/k8s/kube-nginx/sbin/kube-nginx
    ssh root@${node_ip} "chmod a+x /opt/k8s/kube-nginx/sbin/*"
    ssh root@${node_ip} "mkdir -p /opt/k8s/kube-nginx/{conf,logs,sbin}"
    sleep 3
  done
```

配置Nginx文件，开启4层透明转发

```
cd /opt/k8s/work
cat > kube-nginx.conf <<EOF
worker_processes 1;
events {
    worker_connections  1024;
}
stream {
    upstream backend {
        hash $remote_addr consistent;
        server 192.168.0.50:6443        max_fails=3 fail_timeout=30s;
        server 192.168.0.51:6443        max_fails=3 fail_timeout=30s;
        server 192.168.0.52:6443        max_fails=3 fail_timeout=30s;
    }
    server {
        listen *:8443;
        proxy_connect_timeout 1s;
        proxy_pass backend;
    }
}
EOF
#这里需要将server替换我们自己的地址
```

分发配置文件

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
for node_ip in ${MASTER_IPS[@]}
```

```
  do
    echo ">>> ${node_ip}"
    scp kube-nginx.conf  root@${node_ip}:/opt/k8s/kube-nginx/conf/kube-nginx.conf
  done
```

配置Nginx启动文件

```
cd /opt/k8s/work
cat > kube-nginx.service <<EOF
[Unit]
Description=kube-apiserver nginx proxy
After=network.target
After=network-online.target
Wants=network-online.target
[Service]
Type=forking
ExecStartPre=/opt/k8s/kube-nginx/sbin/kube-nginx -c /opt/k8s/kube-nginx/conf/kube-nginx.conf -p /opt/k8s/kube-nginx -t
ExecStart=/opt/k8s/kube-nginx/sbin/kube-nginx -c /opt/k8s/kube-nginx/conf/kube-nginx.conf -p /opt/k8s/kube-nginx
ExecReload=/opt/k8s/kube-nginx/sbin/kube-nginx -c /opt/k8s/kube-nginx/conf/kube-nginx.conf -p /opt/k8s/kube-nginx -s reload
PrivateTmp=true
Restart=always
RestartSec=5
StartLimitInterval=0
LimitNOFILE=65536
[Install]
WantedBy=multi-user.target
EOF
```

分发nginx启动文件

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
for node_ip in ${MASTER_IPS[@]}
  do
```

```
    echo ">>> ${node_ip}"
    scp kube-nginx.service  root@${node_ip}:/etc/systemd/system/
  done
```

启动 kube-nginx 服务

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
for node_ip in ${MASTER_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh root@${node_ip} "systemctl daemon-reload && systemctl enable kube-nginx && systemctl start kube-nginx"
  done
```

检查 kube-nginx 服务运行状态

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
for node_ip in ${MASTER_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh root@${node_ip} "systemctl status kube-nginx |grep 'Active:'"
  done
```

KeepLived 部署

前面我们也说了，高可用方案需要一个VIP，供集群内部访问

在所有**master**节点安装keeplived

```
yum  install -y keepalived
```

接下来我们要配置keeplive服务

`192.168.0.50`配置

```
cat > /etc/keepalived/keepalived.conf <<EOF
! Configuration File for keepalived
```

```
global_defs {
   router_id 192.168.0.50
}
vrrp_script chk_nginx {
    script "/etc/keepalived/check_port.sh 8443"
    interval 2
    weight -20
}
vrrp_instance VI_1 {
    state MASTER
    interface eth0
    virtual_router_id 251
    priority 100
    advert_int 1
    mcast_src_ip 192.168.0.50
    nopreempt
    authentication {
        auth_type PASS
        auth_pass 11111111
    }
    track_script {
         chk_nginx
    }
    virtual_ipaddress {
        192.168.0.54
    }
}
EOF

## 192.168.0.50 为节点IP，192.168.0.54位VIP
```

将配置拷贝到其他节点，并替换相关IP

```
for node_ip in 192.168.0.50 192.168.0.51 192.168.0.52
  do
    echo ">>> ${node_ip}"
    scp  /etc/keepalived/keepalived.conf $node_ip:/etc/keepalived/keepalived.conf
  done
```

```
#替换IP
ssh root@192.168.0.51 sed -i 's#192.168.0.50#192.168.0.51#g'  /etc/keepalived/keepalived.conf
ssh root@192.168.0.52 sed -i 's#192.168.0.50#192.168.0.52#g'  /etc/keepalived/keepalived.conf
#192.168.0.50不替换是因为已经修改好了
```

创建健康检查脚本

```
vim  /opt/check_port.sh
CHK_PORT=$1
 if [ -n "$CHK_PORT" ];then
        PORT_PROCESS=`ss -lt|grep $CHK_PORT|wc -l`
        if [ $PORT_PROCESS -eq 0 ];then
                echo "Port $CHK_PORT Is Not Used,End."
                exit 1
        fi
 else
        echo "Check Port Cant Be Empty!"
 fi
```

启动keeplived

```
for NODE in k8s-01 k8s-02 k8s-03; do
    echo "--- $NODE ---"
    scp -r /opt/check_port.sh  $NODE:/etc/keepalived/
    ssh $NODE 'systemctl enable --now keepalived'
done
```

启动完毕后ping 192.168.0.54 (VIP)

```
[root@k8s03 ~]# ping 192.168.0.54
PING 192.168.0.54 (192.168.0.54) 56(84) bytes of data.
64 bytes from 192.168.0.54: icmp_seq=1 ttl=64 time=0.055 ms
^C
--- 192.168.0.54 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
```

```
rtt min/avg/max/mdev = 0.055/0.055/0.055/0.000 ms
#如果没有启动，请检查原因。 ps -ef|grep keep 检查是否启动成功
#没有启动成功，请执行下面命令，从新启动。启动成功vip肯定就通了
systemctl start keepalived
```

部署master节点

kubernetes master节点运行组件如下:kube-apiserver、kube-scheduler、kube-controller-manager、kube-nginx

- kube-apiserver、kube-scheduler、kube-controller-manager均以多实例模式运行
- kube-scheduler和kube-controller-manager会自动选举一个leader实例，其他实例处于阻塞模式，当leader挂了后，重新选举产生的leader，从而保证服务可用性
- kube-apiserver是无状态的，需要通过kube-nginx进行代理访问，从而保证服务可用性

> 以下操作都在K8s-01操作

下载kubernetes二进制包，并分发到所有master节点

```
cd /opt/k8s/work
wget http://down.i4t.com/k8s1.14/kubernetes-server-linux-amd64.tar.gz
tar -xzvf kubernetes-server-linux-amd64.tar.gz
cd kubernetes
tar -xzvf  kubernetes-src.tar.gz
```

cd /opt/k8s/work
将压缩包的文件拷贝到所有master节点上

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
for node_ip in ${MASTER_IPS[@]}
  do
    echo ">>> ${node_ip}"
    scp kubernetes/server/bin/kube-apiserver root@${node_ip}:/opt/k8s/bin/
    scp kubernetes/server/bin/{apiextensions-apiserver,cloud-controller-manager,kube-controller-manager,kube-proxy,kube-scheduler,kubeadm,kubectl,kubelet,mounter} root@${node_ip}:/opt/k8s/bin/
    ssh root@${node_ip} "chmod +x /opt/k8s/bin/*"
  done
#同时将kubelet kube-proxy拷贝到所有节点
```

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    scp kubernetes/server/bin/{kubelet,kube-proxy} root@${node_ip}:/opt/k8s/bin/
    ssh root@${node_ip} "chmod +x /opt/k8s/bin/*"
  done
```

创建Kubernetes 证书和私钥
创建签证签名请求

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
cat > kubernetes-csr.json <<EOF
{
  "CN": "kubernetes",
  "hosts": [
    "127.0.0.1",
    "192.168.0.50",
    "192.168.0.51",
    "192.168.0.52",
    "192.168.0.54",
    "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": "BeiJing",
      "L": "BeiJing",
      "O": "k8s",
      "OU": "xuyuntech"
    }
  ]
}
EOF
#需要将集群的所有IP及VIP添加进去
#如果要添加注意最后的逗号，不要忘记添加，否则下一步报错
```

hosts 字段指定授权使用该证书的IP和域名列表，这里列出了master节点IP、kubernetes服务的IP和域名
kubernetes serviceIP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
$ kubectl get svc kubernetes

```
$ kubectl get svc kubernetes
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.254.0.1           443/TCP   31d

#目前我们是看不到
```

生成证书和私钥

```
cfssl gencert -ca=/opt/k8s/work/ca.pem \
      -ca-key=/opt/k8s/work/ca-key.pem \
      -config=/opt/k8s/work/ca-config.json \
      -profile=kubernetes kubernetes-csr.json | cfssljson -bare kubernetes
    ls kubernetes*pem
```

将生成的证书和私钥文件拷贝到所有master节点

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
for node_ip in ${MASTER_IPS[@]}
```

```
  do
    echo ">>> ${node_ip}"
    ssh root@${node_ip} "mkdir -p /etc/kubernetes/cert"
    scp kubernetes*.pem root@${node_ip}:/etc/kubernetes/cert/
  done
```

创建加密配置文件

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
cat > encryption-config.yaml <<EOF
kind: EncryptionConfig
apiVersion: v1
resources:
  - resources:
      - secrets
    providers:
      - aescbc:
          keys:
            - name: key1
              secret: ${ENCRYPTION_KEY}
      - identity: {}
EOF
```

将加密配置文件拷贝到master节点的`/etc/kubernetes`目录下

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
for node_ip in ${MASTER_IPS[@]}
  do
    echo ">>> ${node_ip}"
    scp encryption-config.yaml root@${node_ip}:/etc/kubernetes/
  done
```

创建审计策略文件

```
cd /opt/k8s/work
```

```
source /opt/k8s/bin/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:
```

```yaml
      - 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
```

分发审计策略文件：

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
for node_ip in ${MASTER_IPS[@]}
  do
    echo ">>> ${node_ip}"
    scp audit-policy.yaml root@${node_ip}:/etc/kubernetes/audit-policy.yaml
  done
```

创建证书签名请求

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

- CN名称需要位于kube-apiserver的—requestherader-allowed-names参数中，否则后续访问metrics时会提示权限不足

生成证书和私钥

```
cfssl gencert -ca=/etc/kubernetes/cert/ca.pem \
  -ca-key=/etc/kubernetes/cert/ca-key.pem  \
  -config=/etc/kubernetes/cert/ca-config.json  \
  -profile=kubernetes proxy-client-csr.json | cfssljson -bare proxy-client
ls proxy-client*.pem
```

将生成的证书和私钥文件拷贝到master节点

```
source /opt/k8s/bin/environment.sh
for node_ip in ${MASTER_IPS[@]}
  do
    echo ">>> ${node_ip}"
    scp proxy-client*.pem root@${node_ip}:/etc/kubernetes/cert/
  done
```

创建kube-apiserver启动文件

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
cat > kube-apiserver.service.template <<EOF
[Unit]
Description=Kubernetes API Server
Documentation=https://github.com/GoogleCloudPlatform/kubernetes
After=network.target
[Service]
WorkingDirectory=${K8S_DIR}/kube-apiserver
ExecStart=/opt/k8s/bin/kube-apiserver \\
  --advertise-address=##NODE_IP## \\
  --default-not-ready-toleration-seconds=360 \\
  --default-unreachable-toleration-seconds=360 \\
  --feature-gates=DynamicAuditing=true \\
  --max-mutating-requests-inflight=2000 \\
  --max-requests-inflight=4000 \\
  --default-watch-cache-size=200 \\
  --delete-collection-workers=2 \\
  --encryption-provider-config=/etc/kubernetes/encryption-config.yaml \\
  --etcd-cafile=/etc/kubernetes/cert/ca.pem \\
  --etcd-certfile=/etc/kubernetes/cert/kubernetes.pem \\
  --etcd-keyfile=/etc/kubernetes/cert/kubernetes-key.pem \\
  --etcd-servers=${ETCD_ENDPOINTS} \\
  --bind-address=##NODE_IP## \\
  --secure-port=6443 \\
  --tls-cert-file=/etc/kubernetes/cert/kubernetes.pem \\
  --tls-private-key-file=/etc/kubernetes/cert/kubernetes-key.pem \\
  --insecure-port=0 \\
  --audit-dynamic-configuration \\
  --audit-log-maxage=15 \\
  --audit-log-maxbackup=3 \\
  --audit-log-maxsize=100 \\
  --audit-log-truncate-enabled \\
  --audit-log-path=${K8S_DIR}/kube-apiserver/audit.log \\
  --audit-policy-file=/etc/kubernetes/audit-policy.yaml \\
  --profiling \\
  --anonymous-auth=false \\
  --client-ca-file=/etc/kubernetes/cert/ca.pem \\
  --enable-bootstrap-token-auth \\
  --requestheader-allowed-names="aggregator" \\
  --requestheader-client-ca-file=/etc/kubernetes/cert/ca.pem \\
  --requestheader-extra-headers-prefix="X-Remote-Extra-" \\
  --requestheader-group-headers=X-Remote-Group \\
  --requestheader-username-headers=X-Remote-User \\
  --service-account-key-file=/etc/kubernetes/cert/ca.pem \\
  --authorization-mode=Node,RBAC \\
  --runtime-config=api/all=true \\
  --enable-admission-plugins=NodeRestriction \\
  --allow-privileged=true \\
  --apiserver-count=3 \\
  --event-ttl=168h \\
  --kubelet-certificate-authority=/etc/kubernetes/cert/ca.pem \\
  --kubelet-client-certificate=/etc/kubernetes/cert/kubernetes.pem \\
  --kubelet-client-key=/etc/kubernetes/cert/kubernetes-key.pem \\
  --kubelet-https=true \\
  --kubelet-timeout=10s \\
  --proxy-client-cert-file=/etc/kubernetes/cert/proxy-client.pem \\
  --proxy-client-key-file=/etc/kubernetes/cert/proxy-client-key.pem \\
  --service-cluster-ip-range=${SERVICE_CIDR} \\
  --service-node-port-range=${NODE_PORT_RANGE} \\
  --logtostderr=true \\
  --v=2
Restart=on-failure
RestartSec=10
Type=notify
LimitNOFILE=65536
[Install]
WantedBy=multi-user.target
EOF
```

参数配置说明

```
--advertise-address: apiserver 对外通告的 IP（kubernetes 服务后端节点 IP）
```

```
；
--default-*-toleration-seconds：设置节点异常相关的阈值；
--max-*-requests-inflight：请求相关的最大阈值；
--etcd-*：访问 etcd 的证书和 etcd 服务器地址；
--experimental-encryption-provider-config：指定用于加密 etcd 中 secret 的配置；
--bind-address： https 监听的 IP，不能为 127.0.0.1，否则外界不能访问它的安全端口 6443；
--secret-port：https 监听端口；
--insecure-port=0：关闭监听 http 非安全端口(8080)；
--tls-*-file：指定 apiserver 使用的证书、私钥和 CA 文件；
--audit-*：配置审计策略和审计日志文件相关的参数；
--client-ca-file：验证 client (kue-controller-manager、kube-scheduler、kubelet、kube-proxy 等)请求所带的证书；
--enable-bootstrap-token-auth：启用 kubelet bootstrap 的 token 认证；
--requestheader-*：kube-apiserver 的 aggregator layer 相关的配置参数，proxy-client & HPA 需要使用；
--requestheader-client-ca-file：用于签名 --proxy-client-cert-file 和 --proxy-client-key-file 指定的证书；在启用了 metric aggregator 时使用；
--requestheader-allowed-names：不能为空，值为逗号分割的 --proxy-client-cert-file 证书的 CN 名称，这里设置为 "aggregator"；
--service-account-key-file：签名 ServiceAccount Token 的公钥文件，kube-controller-manager 的 --service-account-private-key-file 定私钥文件，两者配对使用；
--runtime-config=api/all=true： 启用所有版本的 APIs，如 autoscaling/v2alpha1；
--authorization-mode=Node,RBAC、--anonymous-auth=false： 开启 Node 和 RBAC 授权模式，拒绝未授权的请求；
--enable-admission-plugins：启用一些默认关闭的 plugins；
--allow-privileged：运行执行 privileged 权限的容器；
--apiserver-count=3：指定 apiserver 实例的数量；
--event-ttl：指定 events 的保存时间；
--kubelet-：如果指定，则使用 https 访问 kubelet APIs；需要为证书对应的用户(上面 kubernetes.pem 证书的用户为 kubernetes) 用户定义 RBAC 规则，否则访问 kubelet API 时提示未授权；
--proxy-client-*：apiserver 访问 metrics-server 使用的证书；
--service-cluster-ip-range： 指定 Service Cluster IP 地址段；
--service-node-port-range： 指定 NodePort 的端口范围；
如果 kube-apiserver 机器没有运行 kube-proxy，则还需要添加 --enable-aggregator-routing=true 参数；
```

```
 关于 --requestheader-XXX 相关参数，参考：
https://github.com/kubernetes-incubator/apiserver-builder/blob/master/docs/concepts/auth.md
https://docs.bitnami.com/kubernetes/how-to/configure-autoscaling-custom-metrics/
```

注意: requestheader-client-ca-file指定的CA证书，必须具有client auth and server auth
如果—requestheader-allowed-names为空，或者—proxy-client-cert-file证书的CN名称不在allowed-names中，则后续查看node或者Pods的metrics失败

为各个节点创建和分发kube-apiserver启动文件

替换模板文件的变量，为各个节点生成启动文件

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
for (( i=0; i < 3; i++ ))  #这里是三个节点所以为3,请根据实际情况修改,后边不在提示,同理
  do
    sed -e "s/##NODE_NAME##/${MASTER_NAMES[i]}/" -e "s/##NODE_IP##/${MASTER_IPS[i]}/" kube-apiserver.service.template > kube-apiserver-${MASTER_IPS[i]}.service
  done
ls kube-apiserver*.service
```

cd /opt/k8s/work
分发apiserver启动文件

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
for node_ip in ${MASTER_IPS[@]}
  do
    echo ">>> ${node_ip}"
    scp kube-apiserver-${node_ip}.service root@${node_ip}:/etc/systemd/system/kube-apiserver.service
  done
启动apiserver
source /opt/k8s/bin/environment.sh
for node_ip in ${MASTER_IPS[@]}
  do
```

```
    echo ">>> ${node_ip}"
    ssh root@${node_ip} "mkdir -p ${K8S_DIR}/kube-apiserver"
    ssh root@${node_ip} "systemctl daemon-reload && systemctl enable kube-apiserver && systemctl restart kube-apiserver"
  done
```

检查服务是否正常

```
source /opt/k8s/bin/environment.sh
for node_ip in ${MASTER_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh root@${node_ip} "systemctl status kube-apiserver |grep 'Active:'"
  done
```

确保状态为active (running)，否则查看日志，确认原因
journalctl -u kube-apiserver
打印kube-apiserver写入etcd数据

```
source /opt/k8s/bin/environment.sh
ETCDCTL_API=3 etcdctl \
    --endpoints=${ETCD_ENDPOINTS} \
    --cacert=/opt/k8s/work/ca.pem \
    --cert=/opt/k8s/work/etcd.pem \
    --key=/opt/k8s/work/etcd-key.pem \
    get /registry/ --prefix --keys-only
```

检查kube-apiserver监听的端口

```
netstat -lntup|grep kube
tcp        0      0 192.168.0.50:6443       0.0.0.0:*               LISTEN      11739/kube-apiserve
```

检查集群信息

```
$ kubectl cluster-info
```

```
Kubernetes master is running at https://192.168.0.54:8443
To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
$ kubectl get all --all-namespaces
NAMESPACE   NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
default     service/kubernetes   ClusterIP   10.254.0.1                 443/TCP   3m5s
$ kubectl get componentstatuses
NAME                 STATUS      MESSAGE                                                                                     ERROR
scheduler            Unhealthy   Get http://127.0.0.1:10251/healthz: dial tcp 127.0.0.1:10251: connect: connection refused
controller-manager   Unhealthy   Get http://127.0.0.1:10252/healthz: dial tcp 127.0.0.1:10252: connect: connection refused
etcd-2               Healthy     {"health":"true"}
etcd-0               Healthy     {"health":"true"}
etcd-1               Healthy     {"health":"true"}
```

如果提示有报错，请检查`~/.kube/config`以及配置证书是否有问题

授权kube-apiserver访问kubelet API的权限

在执行kubectl命令时，apiserver会将请求转发到kubelet的https端口。这里定义的RBAC规则，授权apiserver使用的证书(kubernetes.pem)用户名(CN:kubernetes)访问kubelet API的权限

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

部署高可用kube-controller-manager集群

该集群包含三个节点，启动后通过竞争选举机制产生一个leader节点，其他节点为阻塞状态。当leader节点不可用时，阻塞节点将会在此选举产生新的leader，从而保证服务的高可用。为保证通信安全，这里采用x509证书和私钥，kube-controller-manager在与apiserver的安全端口（http 10252）通信使用；

创建kube-controller-manager证书和私钥

创建证书签名请求

```
cd /opt/k8s/work
cat > kube-controller-manager-csr.json <<EOF
{
    "CN": "system:kube-controller-manager",
    "key": {
        "algo": "rsa",
        "size": 2048
    },
    "hosts": [
      "127.0.0.1",
      "192.168.0.50",
      "192.168.0.51",
      "192.168.0.52"
    ],
    "names": [
      {
        "C": "CN",
        "ST": "BeiJing",
        "L": "BeiJing",
        "O": "system:kube-controller-manager",
        "OU": "xuyuntech"
      }
    ]
}
EOF
```

#这里的IP地址为master ip

- host列表包含所有的kube-controller-manager节点IP(VIP不需要输入)
- CN和O均为system:kube-controller-manager，kubernetes内置的ClusterRoleBindings system:kube-controller-manager赋予kube-controller-manager工作所需权限

生成证书和私钥

```
cd /opt/k8s/work
cfssl gencert -ca=/opt/k8s/work/ca.pem \
  -ca-key=/opt/k8s/work/ca-key.pem \
  -config=/opt/k8s/work/ca-config.json \
  -profile=kubernetes kube-controller-manager-csr.json | cfssljson -bare kube-controller-manager
ls kube-controller-manager*pem
```

将生成的证书和私钥分发到所有master节点

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
for node_ip in ${MASTER_IPS[@]}
  do
    echo ">>> ${node_ip}"
    scp kube-controller-manager*.pem root@${node_ip}:/etc/kubernetes/cert/
  done
```

创建和分发kubeconfig文件
#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

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
kubectl config set-cluster kubernetes \
  --certificate-authority=/opt/k8s/work/ca.pem \
  --embed-certs=true \
  --server=${KUBE_APISERVER} \
  --kubeconfig=kube-controller-manager.kubeconfig
kubectl config set-credentials system:kube-controller-manager \
  --client-certificate=kube-controller-manager.pem \
  --client-key=kube-controller-manager-key.pem \
  --embed-certs=true \
  --kubeconfig=kube-controller-manager.kubeconfig
kubectl config set-context system:kube-controller-manager \
  --cluster=kubernetes \
  --user=system:kube-controller-manager \
  --kubeconfig=kube-controller-manager.kubeconfig
kubectl config use-context system:kube-controller-manager --kubeconfig=kube-controller-manager.kubeconfig
```

分发kubeconfig到所有master节点

```
cd /opt/k8s/work
```

```
source /opt/k8s/bin/environment.sh
for node_ip in ${MASTER_IPS[@]}
  do
    echo ">>> ${node_ip}"
    scp kube-controller-manager.kubeconfig root@${node_ip}:/etc/kubernetes/
  done
```

cd /opt/k8s/work
创建kube-controller-manager启动文件

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
cat > kube-controller-manager.service.template <<EOF
[Unit]
Description=Kubernetes Controller Manager
Documentation=https://github.com/GoogleCloudPlatform/kubernetes
[Service]
WorkingDirectory=${K8S_DIR}/kube-controller-manager
ExecStart=/opt/k8s/bin/kube-controller-manager \\
  --profiling \\
  --cluster-name=kubernetes \\
  --controllers=*,bootstrapsigner,tokencleaner \\
  --kube-api-qps=1000 \\
  --kube-api-burst=2000 \\
  --leader-elect \\
  --use-service-account-credentials\\
  --concurrent-service-syncs=2 \\
  --bind-address=0.0.0.0 \\
  #--secure-port=10252 \\
  --tls-cert-file=/etc/kubernetes/cert/kube-controller-manager.pem \\
  --tls-private-key-file=/etc/kubernetes/cert/kube-controller-manager-key.pem \\
  #--port=0 \\
  --authentication-kubeconfig=/etc/kubernetes/kube-controller-manager.kubeconfig \\
  --client-ca-file=/etc/kubernetes/cert/ca.pem \\
  --requestheader-allowed-names="" \\
  --requestheader-client-ca-file=/etc/kubernetes/cert/ca.pem \\
  --requestheader-extra-headers-prefix="X-Remote-Extra-" \\
  --requestheader-group-headers=X-Remote-Group \\
  --requestheader-username-headers=X-Remote-User \\
  --authorization-kubeconfig=/etc/kubernetes/kube-controller-manager.kubeconfig \\
  --cluster-signing-cert-file=/etc/kubernetes/cert/ca.pem \\
  --cluster-signing-key-file=/etc/kubernetes/cert/ca-key.pem \\
  --experimental-cluster-signing-duration=876000h \\
  --horizontal-pod-autoscaler-sync-period=10s \\
  --concurrent-deployment-syncs=10 \\
  --concurrent-gc-syncs=30 \\
  --node-cidr-mask-size=24 \\
  --service-cluster-ip-range=${SERVICE_CIDR} \\
  --pod-eviction-timeout=6m \\
  --terminated-pod-gc-threshold=10000 \\
  --root-ca-file=/etc/kubernetes/cert/ca.pem \\
  --service-account-private-key-file=/etc/kubernetes/cert/ca-key.pem \\
  --kubeconfig=/etc/kubernetes/kube-controller-manager.kubeconfig \\
  --logtostderr=true \\
  --v=2
Restart=on-failure
RestartSec=5
[Install]
WantedBy=multi-user.target
EOF
```

cd /opt/k8s/work
参数解释

- —port=0：关闭监听非安全端口（http），同时 —address 参数无效，—bind-address 参数有效；
- —secure-port=10252、—bind-address=0.0.0.0: 在所有网络接口监听 10252 端口的 https /metrics 请求；
- —kubeconfig：指定 kubeconfig 文件路径，kube-controller-manager 使用它连接和验证 kube-apiserver；
- —authentication-kubeconfig 和 —authorization-kubeconfig：kube-controller-manager 使用它连接 apiserver，对 client 的请求进行认证和授权。kube-controller-manager 不再使用 —tls-ca-file 对请求 https metrics 的 Client 证书进行校验。如果没有配置这两个 kubeconfig 参数，则 client 连接 kube-controller-manager https 端口的请求会被拒绝(提示权限不足)。

- —cluster-signing-*-file：签名 TLS Bootstrap 创建的证书；
- —experimental-cluster-signing-duration：指定 TLS Bootstrap 证书的有效期；
- —root-ca-file：放置到容器 ServiceAccount 中的 CA 证书，用来对 kube-apiserver 的证书进行校验；
- —service-account-private-key-file：签名 ServiceAccount 中 Token 的私钥文件，必须和 kube-apiserver 的 —service-account-key-file 指定的公钥文件配对使用；
- —service-cluster-ip-range ：指定 Service Cluster IP 网段，必须和 kube-apiserver 中的同名参数一致；
- —leader-elect=true：集群运行模式，启用选举功能；被选为 leader 的节点负责处理工作，其它节点为阻塞状态；
- —controllers=*,bootstrapsigner,tokencleaner：启用的控制器列表，tokencleaner 用于自动清理过期的 Bootstrap token；
- —horizontal-pod-autoscaler-*：custom metrics 相关参数，支持 autoscaling/v2alpha1；
- —tls-cert-file、—tls-private-key-file：使用 https 输出 metrics 时使用的 Server 证书和秘钥；
- —use-service-account-credentials=true: kube-controller-manager 中各 controller 使用 serviceaccount 访问 kube-apiserver；

替换启动文件，并分发脚本

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
for (( i=0; i < 3; i++ ))
  do
    sed -e "s/##NODE_NAME##/${MASTER_NAMES[i]}/" -e "s/##NODE_IP##/${MASTER_IPS[i]}/" kube-controller-manager.service.template > kube-controller-manager-${MASTER_IPS[i]}.service
  done
ls kube-controller-manager*.service
```

分发到所有master节点

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
for node_ip in ${MASTER_IPS[@]}
  do
    echo ">>> ${node_ip}"
    scp kube-controller-manager-${node_ip}.service root@${node_ip}:/etc/systemd/system/kube-controller-manager.service
  done
```

启动服务

```
source /opt/k8s/bin/environment.sh
for node_ip in ${MASTER_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh root@${node_ip} "mkdir -p ${K8S_DIR}/kube-controller-manager"
    ssh root@${node_ip} "systemctl daemon-reload && systemctl enable kube-controller-manager && systemctl restart kube-controller-manager"
  done
```

检查运行状态

```
source /opt/k8s/bin/environment.sh
for node_ip in ${MASTER_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh root@${node_ip} "systemctl status kube-controller-manager|grep Active"
  done
```

检查服务状态

```
netstat -lnpt | grep kube-cont
tcp6       0      0 :::10252                :::*                    LISTEN      13279/kube-controll
tcp6       0      0 :::10257                :::*                    LISTEN      13279/kube-controll
```

kube-controller-manager 创建权限

ClusteRole system:kube-controller-manager的权限太小，只能创建secret、serviceaccount等资源，将controller的权限分散到ClusterRole system:controller:xxx中

```
$ 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
  ---------                                  -----------------  --------------  -----
  secrets                                    []                 []              [create delete get update]
  endpoints                                  []                 []              [create get update]
  serviceaccounts                            []                 []              [create get update]
  events                                     []                 []              [create patch update]
  tokenreviews.authentication.k8s.io         []                 []              [create]
  subjectaccessreviews.authorization.k8s.io  []                 []              [create]
  configmaps                                 []                 []              [get]
  namespaces                                 []                 []              [get]
  *.*                                        []                 []              [list watch]
```

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。

```
$ kubectl get clusterrole|grep controller
system:controller:attachdetach-controller                              22m
system:controller:certificate-controller                               22m
system:controller:clusterrole-aggregation-controller                   22m
system:controller:cronjob-controller
```

```
22m
system:controller:daemon-set-controller
22m
system:controller:deployment-controller
22m
system:controller:disruption-controller
22m
system:controller:endpoint-controller
22m
system:controller:expand-controller
22m
system:controller:generic-garbage-collector
22m
system:controller:horizontal-pod-autoscaler
22m
system:controller:job-controller
22m
system:controller:namespace-controller
22m
system:controller:node-controller
22m
system:controller:persistent-volume-binder
22m
system:controller:pod-garbage-collector
22m
system:controller:pv-protection-controller
22m
system:controller:pvc-protection-controller
22m
system:controller:replicaset-controller
22m
system:controller:replication-controller
22m
system:controller:resourcequota-controller
22m
system:controller:route-controller
22m
system:controller:service-account-controller
22m
system:controller:service-controller
```

```
                                                            22m
system:controller:statefulset-controller                    22m
system:controller:ttl-controller                            22m
system:kube-controller-manager                              22m
```

以 deployment controller 为例：

```
$ kubectl describe clusterrole system:controller:deployment-controller
Name:         system:controller:deployment-controller
Labels:       kubernetes.io/bootstrapping=rbac-defaults
Annotations:  rbac.authorization.kubernetes.io/autoupdate: true
PolicyRule:
  Resources                          Non-Resource URLs  Resource Names  Verbs
  ---------                          -----------------  --------------  -----
  replicasets.apps                   []                 []              [create delete get list patch update watch]
  replicasets.extensions             []                 []              [create delete get list patch update watch]
  events                             []                 []              [create patch update]
  pods                               []                 []              [get list update watch]
  deployments.apps                   []                 []              [get list update watch]
  deployments.extensions             []                 []              [get list update watch]
  deployments.apps/finalizers        []                 []              [update]
  deployments.apps/status            []                 []              [update]
  deployments.extensions/finalizers  []                 []              [update]
  deployments.extensions/status      []                 []
  [update]
```

查看当前的 leader

```
$ kubectl get endpoints kube-controller-manager --namespace=kube-system  -o yaml
apiVersion: v1
kind: Endpoints
metadata:
  annotations:
    control-plane.alpha.kubernetes.io/leader: '{"holderIdentity":"k8s01_56e187ed-bc5b-11e9-b4a3-000c291b8bf5","leaseDurationSeconds":15,"acquireTime":"2019-08-11T17:13:29Z","renewTime":"2019-08-11T17:19:06Z","leaderTransitions":0}'
  creationTimestamp: "2019-08-11T17:13:29Z"
  name: kube-controller-manager
  namespace: kube-system
  resourceVersion: "848"
  selfLink: /api/v1/namespaces/kube-system/endpoints/kube-controller-manager
  uid: 56e64ea1-bc5b-11e9-b77e-000c291b8bf5
```

部署高可用kube-scheduler

创建 kube-scheduler 证书和私钥

创建证书签名请求：

```
cd /opt/k8s/work
cat > kube-scheduler-csr.json <<EOF
{
    "CN": "system:kube-scheduler",
    "hosts": [
      "127.0.0.1",
      "192.168.0.50",
      "192.168.0.51",
      "192.168.0.52"
    ],
    "key": {
        "algo": "rsa",
        "size": 2048
    },
    "names": [
      {
        "C": "CN",
        "ST": "BeiJing",
        "L": "BeiJing",
        "O": "system:kube-scheduler",
        "OU": "xuyuntech"
      }
    ]
}
EOF
```

- hosts 列表包含所有 kube-scheduler 节点 IP；
- CN 和 O 均为 system:kube-scheduler，kubernetes 内置的 ClusterRoleBindings system:kube-scheduler 将赋予 kube-scheduler 工作所需的权限；

生成证书和私钥：

```
cd /opt/k8s/work
cfssl gencert -ca=/opt/k8s/work/ca.pem \
  -ca-key=/opt/k8s/work/ca-key.pem \
  -config=/opt/k8s/work/ca-config.json \
  -profile=kubernetes kube-scheduler-csr.json | cfssljson -bare kube-scheduler
ls kube-scheduler*pem
```

将生成的证书和私钥分发到所有 master 节点

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
for node_ip in ${MASTER_IPS[@]}
  do
    echo ">>> ${node_ip}"
    scp kube-scheduler*.pem root@${node_ip}:/etc/kubernetes/cert/
  done
```

创建和分发 kubeconfig 文件

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
kubectl config set-cluster kubernetes \
  --certificate-authority=/opt/k8s/work/ca.pem \
  --embed-certs=true \
  --server=${KUBE_APISERVER} \
  --kubeconfig=kube-scheduler.kubeconfig
kubectl config set-credentials system:kube-scheduler \
  --client-certificate=kube-scheduler.pem \
  --client-key=kube-scheduler-key.pem \
  --embed-certs=true \
  --kubeconfig=kube-scheduler.kubeconfig
kubectl config set-context system:kube-scheduler \
  --cluster=kubernetes \
  --user=system:kube-scheduler \
  --kubeconfig=kube-scheduler.kubeconfig
kubectl config use-context system:kube-scheduler --kubeconfig=kube-scheduler.kubeconfig
```

分发 kubeconfig 到所有 master 节点

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
for node_ip in ${MASTER_IPS[@]}
  do
    echo ">>> ${node_ip}"
    scp kube-scheduler.kubeconfig root@${node_ip}:/etc/kubernetes/
  done
```

创建 kube-scheduler 配置文件

```
cd /opt/k8s/work
cat >kube-scheduler.yaml.template <<EOF
apiVersion: kubescheduler.config.k8s.io/v1alpha1
kind: KubeSchedulerConfiguration
bindTimeoutSeconds: 600
```

```
clientConnection:
  burst: 200
  kubeconfig: "/etc/kubernetes/kube-scheduler.kubeconfig"
  qps: 100
enableContentionProfiling: false
enableProfiling: true
hardPodAffinitySymmetricWeight: 1
healthzBindAddress: 127.0.0.1:10251
leaderElection:
  leaderElect: true
metricsBindAddress: ##NODE_IP##:10251
EOF
```

- —kubeconfig：指定 kubeconfig 文件路径，kube-scheduler 使用它连接和验证 kube-apiserver；
- —leader-elect=true：集群运行模式，启用选举功能；被选为 leader 的节点负责处理工作，其它节点为阻塞状态；

替换模板文件中的变量：

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
for (( i=0; i < 3; i++ ))
  do
    sed -e "s/##NODE_NAME##/${NODE_NAMES[i]}/" -e "s/##NODE_IP##/${NODE_IPS[i]}/" kube-scheduler.yaml.template > kube-scheduler-${NODE_IPS[i]}.yaml
  done
ls kube-scheduler*.yaml
```

分发 kube-scheduler 配置文件到所有 master 节点：

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
for node_ip in ${MASTER_IPS[@]}
  do
    echo ">>> ${node_ip}"
    scp kube-scheduler-${node_ip}.yaml root@${node_ip}:/etc/kubernetes/kube-scheduler.yaml
```

```
 done
```

创建kube-scheduler启动文件

```
cd /opt/k8s/work
cat > kube-scheduler.service.template <<EOF
[Unit]
Description=Kubernetes Scheduler
Documentation=https://github.com/GoogleCloudPlatform/kubernetes
[Service]
WorkingDirectory=${K8S_DIR}/kube-scheduler
ExecStart=/opt/k8s/bin/kube-scheduler \\
  --config=/etc/kubernetes/kube-scheduler.yaml \\
  --bind-address=##NODE_IP## \\
  --secure-port=10259 \\
  --port=0 \\
  --tls-cert-file=/etc/kubernetes/cert/kube-scheduler.pem \\
  --tls-private-key-file=/etc/kubernetes/cert/kube-scheduler-key.pem \\
  --authentication-kubeconfig=/etc/kubernetes/kube-scheduler.kubeconfig \\
  --client-ca-file=/etc/kubernetes/cert/ca.pem \\
  --requestheader-allowed-names="" \\
  --requestheader-client-ca-file=/etc/kubernetes/cert/ca.pem \\
  --requestheader-extra-headers-prefix="X-Remote-Extra-" \\
  --requestheader-group-headers=X-Remote-Group \\
  --requestheader-username-headers=X-Remote-User \\
  --authorization-kubeconfig=/etc/kubernetes/kube-scheduler.kubeconfig \\
  --logtostderr=true \\
  --v=2
Restart=always
RestartSec=5
StartLimitInterval=0
[Install]
WantedBy=multi-user.target
EOF
```

分发配置文件

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
for (( i=0; i < 3; i++ ))
  do
    sed -e "s/##NODE_NAME##/${NODE_NAMES[i]}/" -e "s/##NODE_IP##/${NODE_IPS[i]}/" kube-scheduler.service.template > kube-scheduler-${NODE_IPS[i]}.service
  done
ls kube-scheduler*.service
```

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
for node_ip in ${MASTER_IPS[@]}
  do
    echo ">>> ${node_ip}"
    scp kube-scheduler-${node_ip}.service root@${node_ip}:/etc/systemd/system/kube-scheduler.service
  done
```

启动kube-scheduler

```
source /opt/k8s/bin/environment.sh
for node_ip in ${MASTER_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh root@${node_ip} "mkdir -p ${K8S_DIR}/kube-scheduler"
    ssh root@${node_ip} "systemctl daemon-reload && systemctl enable kube-scheduler && systemctl restart kube-scheduler"
done
```

检查服务运行状态

```
source /opt/k8s/bin/environment.sh
for node_ip in ${MASTER_IPS[@]}
```

```
do
  echo ">>> ${node_ip}"
  ssh root@${node_ip} "systemctl status kube-scheduler|grep Active"
done
```

查看输出的 metrics
注意：以下命令在 kube-scheduler 节点上执行。
kube-scheduler 监听 10251 和 10251 端口：
10251：接收 http 请求，非安全端口，不需要认证授权；
10259：接收 https 请求，安全端口，需要认证授权；
两个接口都对外提供 /metrics 和 /healthz 的访问。

```
curl -s http://192.168.0.50:10251/metrics|head
# HELP apiserver_audit_event_total Counter of audit events generated and sent to the audit backend.
# TYPE apiserver_audit_event_total counter
apiserver_audit_event_total 0
# HELP apiserver_audit_requests_rejected_total Counter of apiserver requests rejected due to an error in audit logging backend.
# TYPE apiserver_audit_requests_rejected_total counter
apiserver_audit_requests_rejected_total 0
# HELP apiserver_client_certificate_expiration_seconds Distribution of the remaining lifetime on the certificate used to authenticate a request.
# TYPE apiserver_client_certificate_expiration_seconds histogram
apiserver_client_certificate_expiration_seconds_bucket{le="0"} 0
apiserver_client_certificate_expiration_seconds_bucket{le="1800"} 0
```

查看当前leader

```
$ kubectl get endpoints kube-scheduler --namespace=kube-system  -o yaml
apiVersion: v1
kind: Endpoints
metadata:
  annotations:
    control-plane.alpha.kubernetes.io/leader: '{"holderIdentity":"k8s01_72210df0-bc5d-11e9-9ca8-000c291b8bf5","leaseDurationSeconds":1
5,"acquireTime":"2019-08-11T17:28:35Z","renewTime":"2019-08-11T17:31:06Z","leaderTransitions":0}'
  creationTimestamp: "2019-08-11T17:28:35Z"
  name: kube-scheduler
  namespace: kube-system
  resourceVersion: "1500"
  selfLink: /api/v1/namespaces/kube-system/endpoints/kube-scheduler
  uid: 72bcd72f-bc5d-11e9-b77e-000c291b8bf5
```

---

work节点安装
kubernetes work节点运行如下组件: >docker、kubelet、kube-proxy、flanneld、kube-nginx
前面已经安装flanneld这就不在安装了
安装依赖包

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh root@${node_ip} "yum install -y epel-release"
    ssh root@${node_ip} "yum install -y conntrack ipvsadm ntp ntpdate ipset jq iptables curl sysstat libseccomp && modprobe ip_vs "
  done
```

部署Docker组件
我们在所有节点安装docker，这里使用阿里云的yum安装

> **Docker步骤需要在所有节点安装**

```
yum install -y yum-utils device-mapper-persistent-data lvm2
yum-config-manager --add-repo http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo
yum makecache fast
yum -y install docker-ce
```

创建配置文件

```
mkdir -p /etc/docker/
cat > /etc/docker/daemon.json <<EOF
{
  "exec-opts": ["native.cgroupdriver=systemd"],
  "registry-mirrors": ["https://hjvrgh7a.mirror.aliyuncs.com"],
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "100m"
  },
  "storage-driver": "devicemapper"
}
EOF
```

#这里配置当时镜像加速器,可以不进行配置，但是建议配置

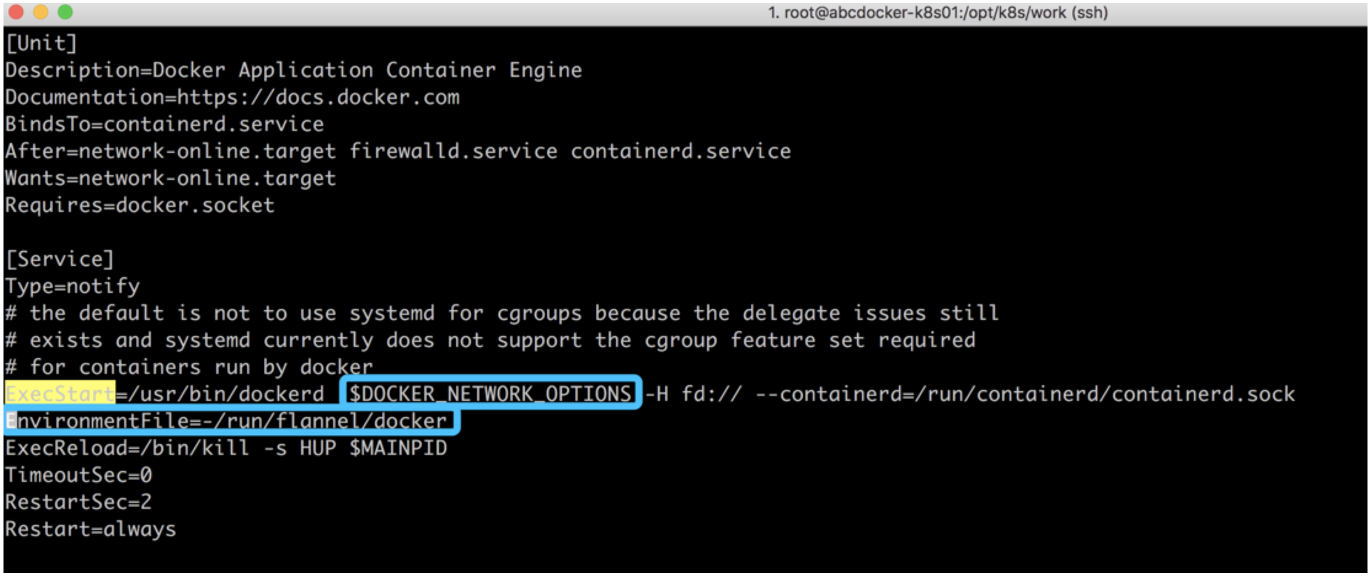
要添加我们harbor仓库需要在添加下面一行

 "insecure-registries": ["harbor.i4t.com"],

默认docker hub需要https协议，使用上面配置不需要配置https

修改Docker启动参数

> 这里需要在所有的节点上修改docker配置！！



```
[Unit]
Description=Docker Application Container Engine
Documentation=https://docs.docker.com
BindsTo=containerd.service
After=network-online.target firewalld.service containerd.service
Wants=network-online.target
Requires=docker.socket

[Service]
Type=notify
# the default is not to use systemd for cgroups because the delegate issues still
# exists and systemd currently does not support the cgroup feature set required
# for containers run by docker
ExecStart=/usr/bin/dockerd $DOCKER_NETWORK_OPTIONS -H fd:// --containerd=/run/containerd/containerd.sock
EnvironmentFile=-/run/flannel/docker
ExecReload=/bin/kill -s HUP $MAINPID
TimeoutSec=0
RestartSec=2
Restart=always
```

```
EnvironmentFile=-/run/flannel/docker
ExecStart=/usr/bin/dockerd $DOCKER_NETWORK_OPTIONS -H fd:// --containerd=/run/containerd/containerd.sock
```

完整配置如下

```
$ cat /usr/lib/systemd/system/docker.service
[Unit]
Description=Docker Application Container Engine
Documentation=https://docs.docker.com
BindsTo=containerd.service
After=network-online.target firewalld.service containerd.service
Wants=network-online.target
Requires=docker.socket
[Service]
Type=notify
# the default is not to use systemd for cgroups because the delegate issues still
# exists and systemd currently does not support the cgroup feature set required
# for containers run by docker
ExecStart=/usr/bin/dockerd  $DOCKER_NETWORK_OPTIONS -H fd:// --containerd=/run/containerd/containerd.sock
EnvironmentFile=-/run/flannel/docker
ExecReload=/bin/kill -s HUP $MAINPID
TimeoutSec=0
RestartSec=2
Restart=always
# Note that StartLimit* options were moved from "Service" to "Unit" in systemd 229.
# Both the old, and new location are accepted by systemd 229 and up, so using the old location
# to make them work for either version of systemd.
StartLimitBurst=3
# Note that StartLimitInterval was renamed to StartLimitIntervalSec in systemd 230.
# Both the old, and new name are accepted by systemd 230 and up, so using the old name to make
# this option work for either version of systemd.
StartLimitInterval=60s
# Having non-zero Limit*s causes performance problems due to accounting overhead
# in the kernel. We recommend using cgroups to do container-local accounting.
LimitNOFILE=infinity
```

```
LimitNPROC=infinity
LimitCORE=infinity
# Comment TasksMax if your systemd version does not support it.
# Only systemd 226 and above support this option.
TasksMax=infinity
# set delegate yes so that systemd does not reset the cgroups of docker containers
Delegate=yes
# kill only the docker process, not all processes in the cgroup
KillMode=process
[Install]
WantedBy=multi-user.target
```

启动 docker 服务

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh root@${node_ip} "systemctl daemon-reload && systemctl enable docker && systemctl restart docker"
  done
```

检查服务运行状态

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh root@${node_ip} "systemctl status docker|grep Active"
  done
```

检查 docker0 网桥

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
```

```
    echo ">>> ${node_ip}"
    ssh root@${node_ip} "/usr/sbin/ip addr show flannel.1 && /usr/sbin/ip addr show docker0"
  done
```

查看 docker 的状态信息

```
docker info
```

#查看docker版本以及存储引擎是否是overlay2

> 以上Docker步骤，有很多需要进入每台服务器进行修改配置文件！！

部署kubelet组件

kubelet运行在每个worker节点上，接收kube-apiserver发送的请求，管理Pod容器，执行交互命令

kubelet启动时自动向kube-apiserver注册节点信息，内置的cAdivsor统计和监控节点的资源使用资源情况。为确保安全，部署时关闭了kubelet的非安全http端口，对请求进行认证和授权，拒绝未授权的访问

创建kubelet bootstrap kubeconfig文件

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
for node_name in ${NODE_NAMES[@]}
  do
    echo ">>> ${node_name}"
    # 创建 token
    export BOOTSTRAP_TOKEN=$(kubeadm token create \
      --description kubelet-bootstrap-token \
      --groups system:bootstrappers:${node_name} \
      --kubeconfig ~/.kube/config)
    # 设置集群参数
    kubectl config set-cluster kubernetes \
      --certificate-authority=/etc/kubernetes/cert/ca.pem \
      --embed-certs=true \
      --server=${KUBE_APISERVER} \
      --kubeconfig=kubelet-bootstrap-${node_name}.kubeconfig
    # 设置客户端认证参数
    kubectl config set-credentials kubelet-bootstrap \
      --token=${BOOTSTRAP_TOKEN} \
      --kubeconfig=kubelet-bootstrap-${node_name}.kubeconfig
    # 设置上下文参数
    kubectl config set-context default \
      --cluster=kubernetes \
      --user=kubelet-bootstrap \
      --kubeconfig=kubelet-bootstrap-${node_name}.kubeconfig
    # 设置默认上下文
    kubectl config use-context default --kubeconfig=kubelet-bootstrap-${node_name}.kubeconfig
  done
```

- 向kubeconfig写入的是token，bootstrap结束后kube-controller-manager为kubelet创建client和server证书

查看kubeadm为各个节点创建的token

```
$ kubeadm token list --kubeconfig ~/.kube/config
TOKEN                     TTL       EXPIRES                     USAGES                    DESCRIPTION               EXTRA GROUPS
ds9td8.wazmxhtaznrweknk   23h       2019-08-13T01:54:57+08:00   authentication,signing    kubelet-bootstrap-token   system:bootstrappers:k8s-01
hy5ssz.4zi4e079ovxba52x   23h       2019-08-13T01:54:58+08:00   authentication,signing    kubelet-bootstrap-token   system:bootstrappers:k8s-03
pkkcl0.l7syoup3jedt7c3l   23h       2019-08-13T01:54:57+08:00   authentication,signing    kubelet-bootstrap-token   system:bootstrappers:k8s-02
tubfqq.mja239hszl4rmron   23h       2019-08-13T01:54:58+08:00   authentication,signing    kubelet-bootstrap-token   system:bootstrappers:k8s-04
```

- token有效期为1天，超期后将不能被用来bootstrap kubelet，且会被kube-controller-manager的token cleaner清理
- 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

查看各token关联的Secret

```
$ kubectl get secrets  -n kube-system|grep bootstrap-token
```

```
bootstrap-token-ds9td8                          bootstrap.kubernetes.io/token         7       3m15s
bootstrap-token-hy5ssz                          bootstrap.kubernetes.io/token         7       3m14s
bootstrap-token-pkkcl0                          bootstrap.kubernetes.io/token         7       3m15s
bootstrap-token-tubfqq                          bootstrap.kubernetes.io/token         7       3m14s
```

分发 bootstrap kubeconfig 文件到所有 worker 节点

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
for node_name in ${NODE_NAMES[@]}
  do
    echo ">>> ${node_name}"
    scp kubelet-bootstrap-${node_name}.kubeconfig root@${node_name}:/etc/kubernetes/kubelet-bootstrap.kubeconfig
  done
```

创建和分发kubelet参数配置

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
cat > kubelet-config.yaml.template <<EOF
kind: KubeletConfiguration
apiVersion: kubelet.config.k8s.io/v1beta1
address: "##NODE_IP##"
staticPodPath: ""
syncFrequency: 1m
fileCheckFrequency: 20s
httpCheckFrequency: 20s
staticPodURL: ""
port: 10250
readOnlyPort: 0
rotateCertificates: true
serverTLSBootstrap: true
authentication:
  anonymous:
    enabled: false
  webhook:
    enabled: true
  x509:
    clientCAFile: "/etc/kubernetes/cert/ca.pem"
authorization:
  mode: Webhook
registryPullQPS: 0
registryBurst: 20
eventRecordQPS: 0
eventBurst: 20
enableDebuggingHandlers: true
enableContentionProfiling: true
healthzPort: 10248
healthzBindAddress: "##NODE_IP##"
clusterDomain: "${CLUSTER_DNS_DOMAIN}"
clusterDNS:
  - "${CLUSTER_DNS_SVC_IP}"
nodeStatusUpdateFrequency: 10s
nodeStatusReportFrequency: 1m
imageMinimumGCAge: 2m
imageGCHighThresholdPercent: 85
imageGCLowThresholdPercent: 80
volumeStatsAggPeriod: 1m
kubeletCgroups: ""
systemCgroups: ""
cgroupRoot: ""
cgroupsPerQOS: true
cgroupDriver: systemd
runtimeRequestTimeout: 10m
hairpinMode: promiscuous-bridge
maxPods: 220
podCIDR: "${CLUSTER_CIDR}"
podPidsLimit: -1
resolvConf: /etc/resolv.conf
maxOpenFiles: 1000000
kubeAPIQPS: 1000
kubeAPIBurst: 2000
serializeImagePulls: false
```

```
evictionHard:
  memory.available:  "100Mi"
nodefs.available:  "10%"
nodefs.inodesFree: "5%"
imagefs.available: "15%"
evictionSoft: {}
enableControllerAttachDetach: true
failSwapOn: true
containerLogMaxSize: 20Mi
containerLogMaxFiles: 10
systemReserved: {}
kubeReserved: {}
systemReservedCgroup: ""
kubeReservedCgroup: ""
enforceNodeAllocatable: ["pods"]
EOF
```

- address：kubelet 安全端口（https，10250）监听的地址，不能为 127.0.0.1，否则 kube-apiserver、heapster 等不能调用 kubelet 的 API；
- readOnlyPort=0：关闭只读端口(默认 10255)，等效为未指定；
- authentication.anonymous.enabled：设置为 false，不允许匿名 访问 10250 端口；
- authentication.x509.clientCAFile：指定签名客户端证书的 CA 证书，开启 HTTP 证书认证；
- authentication.webhook.enabled=true：开启 HTTPs bearer token 认证；
- 对于未通过 x509 证书和 webhook 认证的请求(kube-apiserver 或其他客户端)，将被拒绝，提示 Unauthorized；
- authroization.mode=Webhook：kubelet 使用 SubjectAccessReview API 查询 kube-apiserver 某 user、group 是否具有操作资源的权限(RBAC)；
- featureGates.RotateKubeletClientCertificate、featureGates.RotateKubeletServerCertificate：自动 rotate 证书，证书的有效期取决于 kube-controller-manager 的 —experimental-cluster-signing-duration 参数；
- 需要 root 账户运行；

为各个节点创建和分发kubelet配置文件

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    sed -e "s/##NODE_IP##/${node_ip}/" kubelet-config.yaml.template >
```

```
kubelet-config-${node_ip}.yaml.template
    scp kubelet-config-${node_ip}.yaml.template root@${node_ip}:/etc/kubernetes/kubelet-config.yaml
  done
```

创建和分发kubelet启动文件

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
cat > kubelet.service.template <<EOF
[Unit]
Description=Kubernetes Kubelet
Documentation=https://github.com/GoogleCloudPlatform/kubernetes
After=docker.service
Requires=docker.service
[Service]
WorkingDirectory=${K8S_DIR}/kubelet
ExecStart=/opt/k8s/bin/kubelet \\
  --allow-privileged=true \\
  --bootstrap-kubeconfig=/etc/kubernetes/kubelet-bootstrap.kubeconfig \\
  --cert-dir=/etc/kubernetes/cert \\
  --cni-conf-dir=/etc/cni/net.d \\
  --container-runtime=docker \\
  --container-runtime-endpoint=unix:///var/run/dockershim.sock \\
  --root-dir=${K8S_DIR}/kubelet \\
  --kubeconfig=/etc/kubernetes/kubelet.kubeconfig \\
  --config=/etc/kubernetes/kubelet-config.yaml \\
  --hostname-override=##NODE_NAME## \\
  --pod-infra-container-image=registry.cn-beijing.aliyuncs.com/abcdocker/pause-amd64:3.1 \\
  --image-pull-progress-deadline=15m \\
  --volume-plugin-dir=${K8S_DIR}/kubelet/kubelet-plugins/volume/exec/ \\
  --logtostderr=true \\
  --v=2
Restart=always
RestartSec=5
StartLimitInterval=0
```

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

- 如果设置了 —hostname-override 选项，则 kube-proxy 也需要设置该选项，否则会出现找不到 Node 的情况；
- —bootstrap-kubeconfig：指向 bootstrap kubeconfig 文件，kubelet 使用该文件中的用户名和 token 向 kube-apiserver 发送 TLS Bootstrapping 请求；
- K8S approve kubelet 的 csr 请求后，在 —cert-dir 目录创建证书和私钥文件，然后写入 —kubeconfig 文件；
- —pod-infra-container-image 不使用 redhat 的 pod-infrastructure:latest 镜像，它不能回收容器的僵尸；

分发启动文件

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
for node_name in ${NODE_NAMES[@]}
  do
    echo ">>> ${node_name}"
    sed -e "s/##NODE_NAME##/${node_name}/" kubelet.service.template > kubelet-${node_name}.service
    scp kubelet-${node_name}.service root@${node_name}:/etc/systemd/system/kubelet.service
  done
```

Bootstrap Token Auth 和授予权限 kubelet 启动时查找 --kubeletconfig 参数对应的文件是否存在，如果不存在则使用 --bootstrap-kubeconfig 指定的 kubeconfig 文件向 kube-apiserver 发送证书签名请求 (CSR)。 kube-apiserver 收到 CSR 请求后，对其中的 Token 进行认证，认证通过后将请求的 user 设置为 system:bootstrap:，group 设置为 system:bootstrappers，这一过程称为 Bootstrap Token Auth。

创建user和group的CSR权限，不创建kubelet会启动失败

```
$ kubectl create clusterrolebinding kubelet-bootstrap --clusterrole=system:node-bootstrapper --group=system:bootstrappers
```

启动 kubelet 服务

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh root@${node_ip} "mkdir -p ${K8S_DIR}/kubelet/kubelet-plugins/volume/exec/"
    ssh root@${node_ip} "/usr/sbin/swapoff -a"
    ssh root@${node_ip} "systemctl daemon-reload && systemctl enable kubelet && systemctl restart kubelet"
  done
```

> 关闭 swap 分区，否则 kubelet 会启动失败；

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。

```
$ kubectl get csr
NAME        AGE   REQUESTOR                 CONDITION
csr-22kt2   38s   system:bootstrap:pkkcl0   Pending
csr-f9trc   37s   system:bootstrap:tubfqq   Pending
csr-v7jt2   38s   system:bootstrap:ds9td8   Pending
csr-zrww2   37s   system:bootstrap:hy5ssz   Pending
```

这里4个节点均处于pending(等待)状态

自动approve CSR请求

创建三个ClusterRoleBinding，分别用于自动approve client、renew client、renew server证书

```
cd /opt/k8s/work
cat > csr-crb.yaml <<EOF
 # Approve all CSRs for the group "system:bootstrappers"
 kind: ClusterRoleBinding
 apiVersion: rbac.authorization.k8s.io/v1
 metadata:
   name: auto-approve-csrs-for-group
 subjects:
 - kind: Group
```

```
   name: system:bootstrappers
   apiGroup: rbac.authorization.k8s.io
 roleRef:
   kind: ClusterRole
   name: system:certificates.k8s.io:certificatesigningrequests:nodeclient
   apiGroup: rbac.authorization.k8s.io
---
 # To let a node of the group "system:nodes" renew its own credentials
 kind: ClusterRoleBinding
 apiVersion: rbac.authorization.k8s.io/v1
 metadata:
   name: node-client-cert-renewal
 subjects:
 - kind: Group
   name: system:nodes
   apiGroup: rbac.authorization.k8s.io
 roleRef:
   kind: ClusterRole
   name: system:certificates.k8s.io:certificatesigningrequests:selfnodeclient
   apiGroup: rbac.authorization.k8s.io
---
# A ClusterRole which instructs the CSR approver to approve a node requesting a
# serving cert matching its client cert.
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: approve-node-server-renewal-csr
rules:
- apiGroups: ["certificates.k8s.io"]
  resources: ["certificatesigningrequests/selfnodeserver"]
  verbs: ["create"]
---
 # To let a node of the group "system:nodes" renew its own server credentials
 kind: ClusterRoleBinding
 apiVersion: rbac.authorization.k8s.io/v1
 metadata:
   name: node-server-cert-renewal
 subjects:
 - kind: Group
   name: system:nodes
   apiGroup: rbac.authorization.k8s.io
 roleRef:
   kind: ClusterRole
   name: approve-node-server-renewal-csr
   apiGroup: rbac.authorization.k8s.io
EOF
```

```
kubectl apply -f csr-crb.yaml
```

- auto-approve-csrs-for-group 自动approve node的第一次CSR，注意第一次CSR时，请求的Group为system:bootstrappers
- node-client-cert-renewal 自动approve node后续过期的client证书，自动生成的证书Group为system:nodes
- node-server-cert-renewal 自动approve node后续过期的server证书，自动生成的证书Group

查看kubelet

等待1-10分钟，3个节点的CSR都会自动approved

```
$ kubectl get csr
NAME         AGE     REQUESTOR                   CONDITION
csr-22kt2    4m48s   system:bootstrap:pkkcl0     Approved,Issued
csr-d8tvc    77s     system:node:k8s-01          Pending
csr-f9trc    4m47s   system:bootstrap:tubfqq     Approved,Issued
csr-kcdvx    76s     system:node:k8s-02          Pending
csr-m8k8t    75s     system:node:k8s-04          Pending
csr-v7jt2    4m48s   system:bootstrap:ds9td8     Approved,Issued
csr-wwvwd    76s     system:node:k8s-03          Pending
csr-zrww2    4m47s   system:bootstrap:hy5ssz     Approved,Issued
```

Pending的CSR用于创建kubelet serve证书，需要手动approve（后面步骤）

目前所有节点均为ready状态

```
[root@k8s01 work]# kubectl get node
```

```
NAME     STATUS   ROLES    AGE      VERSION
k8s-01   Ready      2m29s   v1.14.2
k8s-02   Ready      2m28s   v1.14.2
k8s-03   Ready      2m28s   v1.14.2
k8s-04   Ready      2m27s   v1.14.2
```

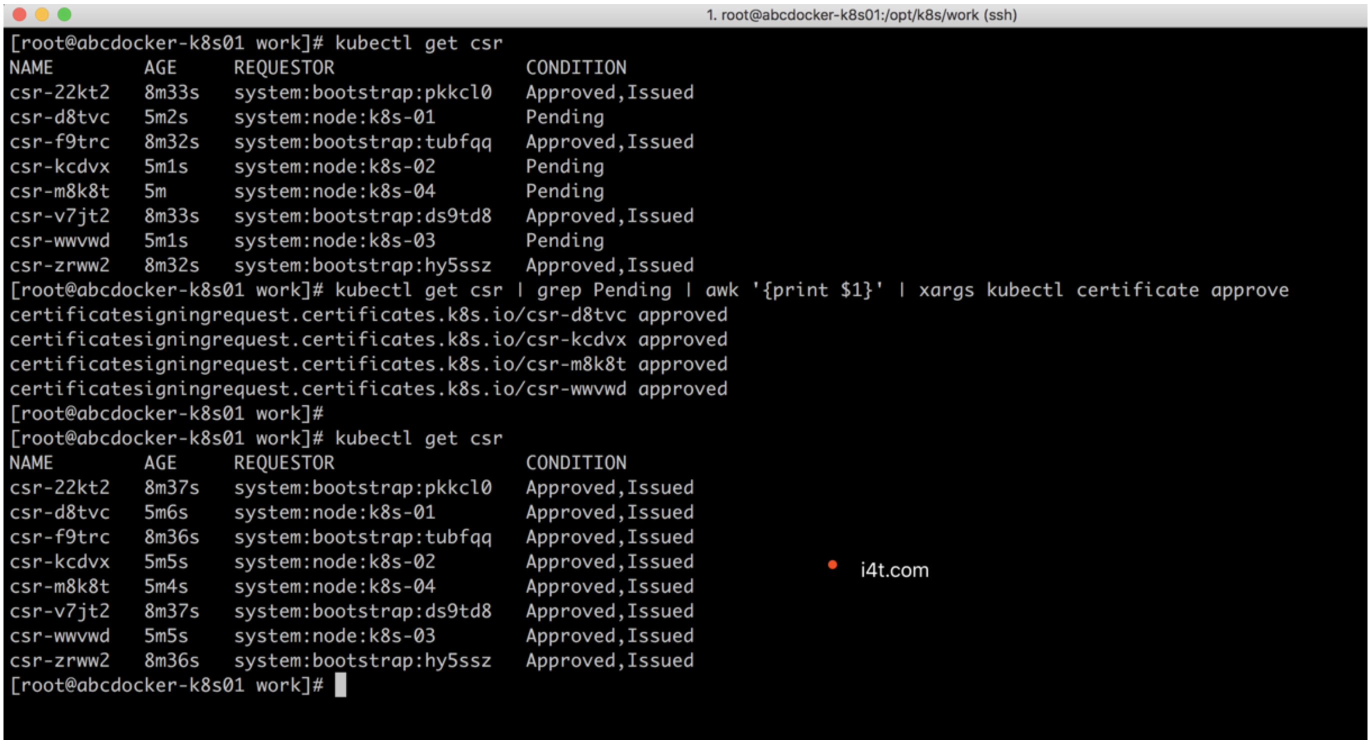
kube-controller-manager为各node生成了kubeconfig文件和公钥

```
$ ls -l /etc/kubernetes/kubelet.kubeconfig
-rw------- 1 root root 2313 Aug 12 02:04 /etc/kubernetes/kubelet.kubeconfig
$ ls -l /etc/kubernetes/cert/|grep kubelet
-rw------- 1 root root 1273 Aug 12 02:07 kubelet-client-2019-08-12-02-07-59.pem
lrwxrwxrwx 1 root root   59 Aug 12 02:07 kubelet-client-current.pem -> /etc/kubernetes/cert/kubelet-client-2019-08-12-02-07-59.pem
```

手动approve server cert csr

基于安全考虑，CSR approving controllers不会自动approve kubelet server证书签名请求，需要手动approve

kubectl get csr | grep Pending | awk '{print $1}' | xargs kubectl certificate approve



```
[root@abcdocker-k8s01 work]# kubectl get csr
NAME        AGE     REQUESTOR                  CONDITION
csr-22kt2   8m33s   system:bootstrap:pkkcl0    Approved,Issued
csr-d8tvc   5m2s    system:node:k8s-01         Pending
csr-f9trc   8m32s   system:bootstrap:tubfqq    Approved,Issued
csr-kcdvx   5m1s    system:node:k8s-02         Pending
csr-m8k8t   5m      system:node:k8s-04         Pending
csr-v7jt2   8m33s   system:bootstrap:ds9td8    Approved,Issued
csr-wwvwd   5m1s    system:node:k8s-03         Pending
csr-zrww2   8m32s   system:bootstrap:hy5ssz    Approved,Issued
[root@abcdocker-k8s01 work]# kubectl get csr | grep Pending | awk '{print $1}' | xargs kubectl certificate approve
certificatesigningrequest.certificates.k8s.io/csr-d8tvc approved
certificatesigningrequest.certificates.k8s.io/csr-kcdvx approved
certificatesigningrequest.certificates.k8s.io/csr-m8k8t approved
certificatesigningrequest.certificates.k8s.io/csr-wwvwd approved
[root@abcdocker-k8s01 work]#
[root@abcdocker-k8s01 work]# kubectl get csr
NAME        AGE     REQUESTOR                  CONDITION
csr-22kt2   8m37s   system:bootstrap:pkkcl0    Approved,Issued
csr-d8tvc   5m6s    system:node:k8s-01         Approved,Issued
csr-f9trc   8m36s   system:bootstrap:tubfqq    Approved,Issued
csr-kcdvx   5m5s    system:node:k8s-02         Approved,Issued
csr-m8k8t   5m4s    system:node:k8s-04         Approved,Issued
csr-v7jt2   8m37s   system:bootstrap:ds9td8    Approved,Issued
csr-wwvwd   5m5s    system:node:k8s-03         Approved,Issued
csr-zrww2   8m36s   system:bootstrap:hy5ssz    Approved,Issued
[root@abcdocker-k8s01 work]#
```

kubelet API接口

kubelet启动后监听多个端口，用于接受kube-apiserver或其他客户端发送的请求

```
netstat -lntup|grep kubelet
tcp        0      0 192.168.0.50:10248      0.0.0.0:*               LISTEN      49491/kubelet
tcp        0      0 127.0.0.1:45737         0.0.0.0:*               LISTEN      49491/kubelet
tcp        0      0 192.168.0.50:10250      0.0.0.0:*               LISTEN      49491/kubelet
```

- 10248: healthz http 服务；
- 10250: https 服务，访问该端口时需要认证和授权（即使访问 /healthz 也需要）；
- 未开启只读端口 10255；
- 从 K8S v1.10 开始，去除了 —cadvisor-port 参数（默认 4194 端口），不支持访问 cAdvisor UI & API

bear token认证和授权

创建一个ServiceAccount，将它和ClusterRole system:kubelet-api-admin绑定，从而具有调用kubelet API的权限

```
kubectl create sa kubelet-api-test
kubectl create clusterrolebinding kubelet-api-test --clusterrole=system:kubelet-api-admin --serviceaccount=default:kubelet-api-test
SECRET=$(kubectl get secrets | grep kubelet-api-test | awk '{print $1}')
TOKEN=$(kubectl describe secret ${SECRET} | grep -E '^token' | awk '{print $2}')
echo ${TOKEN}
```

部署kube-proxy组件

kube-proxy运行在所有worker节点上，它监听apiserver中service和endpoint的变化情况，创建路由规则提供服务IP和负载均衡功能。这里使用ipvs模式的kube-proxy进行部署

> 在各个节点需要安装ipvsadm和ipset命令，加载ip_vs内核模块

创建kube-proxy证书签名请求

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

- CN：指定该证书的 User 为 system:kube-proxy；
- 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；

生成证书和私钥：

```
cd /opt/k8s/work
cfssl gencert -ca=/opt/k8s/work/ca.pem \
  -ca-key=/opt/k8s/work/ca-key.pem \
  -config=/opt/k8s/work/ca-config.json \
  -profile=kubernetes  kube-proxy-csr.json | cfssljson -bare kube-proxy
ls kube-proxy*
```

创建和分发 kubeconfig 文件

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
kubectl config set-cluster kubernetes \
  --certificate-authority=/opt/k8s/work/ca.pem \
  --embed-certs=true \
  --server=${KUBE_APISERVER} \
  --kubeconfig=kube-proxy.kubeconfig
kubectl config set-credentials kube-proxy \
  --client-certificate=kube-proxy.pem \
  --client-key=kube-proxy-key.pem \
  --embed-certs=true \
  --kubeconfig=kube-proxy.kubeconfig
kubectl config set-context default \
  --cluster=kubernetes \
  --user=kube-proxy \
  --kubeconfig=kube-proxy.kubeconfig
kubectl config use-context default --kubeconfig=kube-proxy.kubeconfig
```

- —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)；
- 分发 kubeconfig 文件：

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
for node_name in ${NODE_NAMES[@]}
  do
    echo ">>> ${node_name}"
    scp kube-proxy.kubeconfig root@${node_name}:/etc/kubernetes/
  done
```

创建kube-proxy配置文件

从v1.10开始，kube-proxy部分参数可以配置在文件中，可以使用—write-config-to选项生成该配置文件

```
cd /opt/k8s/work
cat > kube-proxy-config.yaml.template <<EOF
kind: KubeProxyConfiguration
apiVersion: kubeproxy.config.k8s.io/v1alpha1
clientConnection:
  burst: 200
  kubeconfig: "/etc/kubernetes/kube-proxy.kubeconfig"
  qps: 100
bindAddress: ##NODE_IP##
healthzBindAddress: ##NODE_IP##:10256
```

```
metricsBindAddress: ##NODE_IP##:10249
enableProfiling: true
clusterCIDR: ${CLUSTER_CIDR}
hostnameOverride: ##NODE_NAME##
mode: "ipvs"
portRange: ""
kubeProxyIPTablesConfiguration:
  masqueradeAll: false
kubeProxyIPVSConfiguration:
  scheduler: rr
  excludeCIDRs: []
EOF
```

- bindAddress: 监听地址；
- clientConnection.kubeconfig: 连接 apiserver 的 kubeconfig 文件；
- –clusterCIDR: kube-proxy 根据 —cluster-cidr判断集群内部和外部流量，指定 —cluster-cidr 或 —masquerade-all 选项后 kube-proxy 才会对访问 Service IP 的请求做 SNAT；
- hostnameOverride: 参数值必须与 kubelet 的值一致，否则 kube-proxy 启动后会找不到该 Node，从而不会创建任何 ipvs 规则；
- mode: 使用 ipvs 模式；

分发和创建kube-proxy配置文件

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
for (( i=0; i < 4; i++ ))
  do
    echo ">>> ${NODE_NAMES[i]}"
    sed -e "s/##NODE_NAME##/${NODE_NAMES[i]}/" -e "s/##NODE_IP##/${NODE_IPS[i]}/" kube-proxy-config.yaml.template > kube-proxy-config-${NODE_NAMES[i]}.yaml.template
    scp kube-proxy-config-${NODE_NAMES[i]}.yaml.template root@${NODE_NAMES[i]}:/etc/kubernetes/kube-proxy-config.yaml
  done
```

#我这里一共有4个节点要运行，所以这里写4，这是整个集群的node节点的数量！ 这里一定要注意修改！！

创建和分发 kube-proxy systemd unit 文件

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
cat > kube-proxy.service <<EOF
[Unit]
Description=Kubernetes Kube-Proxy Server
Documentation=https://github.com/GoogleCloudPlatform/kubernetes
After=network.target
[Service]
WorkingDirectory=${K8S_DIR}/kube-proxy
ExecStart=/opt/k8s/bin/kube-proxy \\
  --config=/etc/kubernetes/kube-proxy-config.yaml \\
  --logtostderr=true \\
  --v=2
Restart=on-failure
RestartSec=5
LimitNOFILE=65536
[Install]
WantedBy=multi-user.target
EOF
```

分发 kube-proxy systemd unit 文件：

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
for node_name in ${NODE_NAMES[@]}
  do
    echo ">>> ${node_name}"
    scp kube-proxy.service root@${node_name}:/etc/systemd/system/
  done
```

启动 kube-proxy 服务

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh root@${node_ip} "mkdir -p ${K8S_DIR}/kube-proxy"
    ssh root@${node_ip} "modprobe ip_vs_rr"
    ssh root@${node_ip} "systemctl daemon-reload && systemctl enable kube-proxy && systemctl restart kube-proxy"
  done
```

检查启动结果

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh root@${node_ip} "systemctl status kube-proxy|grep Active"
  done
```

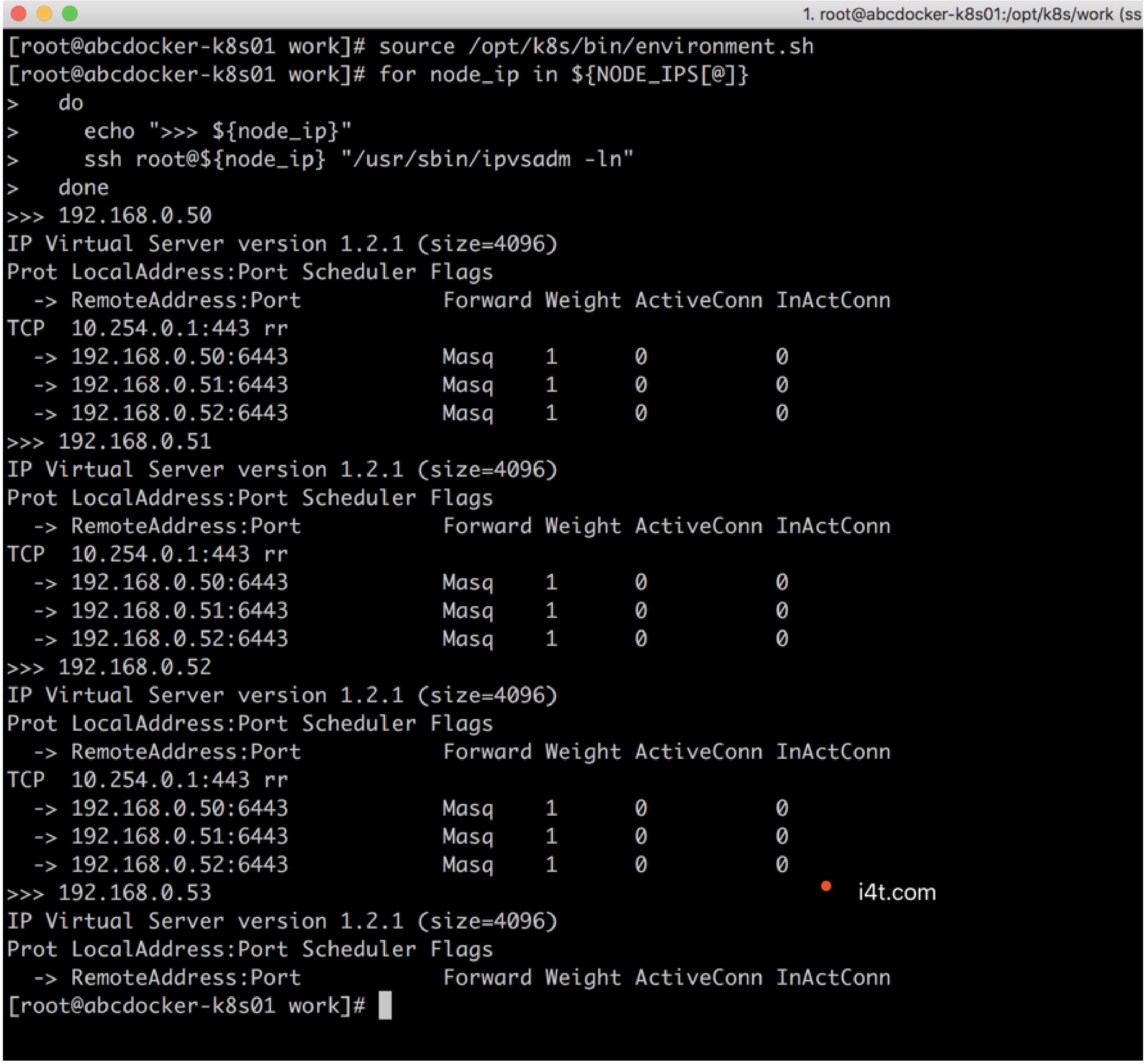
检查监听端口

```
[root@k8s01 work]# netstat -lnpt|grep kube-prox
tcp        0      0 192.168.0.50:10249      0.0.0.0:*               LISTEN      55015/kube-proxy
tcp        0      0 192.168.0.50:10256      0.0.0.0:*               LISTEN      55015/kube-proxy
```

查看ipvs路由规则

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh root@${node_ip} "/usr/sbin/ipvsadm -ln"
  done
```

正常输出如下



```
[root@abcdocker-k8s01 work]# source /opt/k8s/bin/environment.sh
[root@abcdocker-k8s01 work]# for node_ip in ${NODE_IPS[@]}
>   do
>     echo ">>> ${node_ip}"
>     ssh root@${node_ip} "/usr/sbin/ipvsadm -ln"
>   done
>>> 192.168.0.50
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
TCP  10.254.0.1:443 rr
  -> 192.168.0.50:6443            Masq    1      0          0
  -> 192.168.0.51:6443            Masq    1      0          0
  -> 192.168.0.52:6443            Masq    1      0          0
>>> 192.168.0.51
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
TCP  10.254.0.1:443 rr
  -> 192.168.0.50:6443            Masq    1      0          0
  -> 192.168.0.51:6443            Masq    1      0          0
  -> 192.168.0.52:6443            Masq    1      0          0
>>> 192.168.0.52
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
TCP  10.254.0.1:443 rr
  -> 192.168.0.50:6443            Masq    1      0          0
  -> 192.168.0.51:6443            Masq    1      0          0
  -> 192.168.0.52:6443            Masq    1      0          0
>>> 192.168.0.53
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
[root@abcdocker-k8s01 work]#
```

可见所有通过 https 访问 K8S SVC kubernetes 的请求都转发到 kube-apiserver 节点的 6443 端口；

验证集群功能

现在使用daemonset验证master和worker节点是否正常

```
[root@k8s01 work]# kubectl get node
NAME      STATUS    ROLES     AGE    VERSION
k8s-01    Ready        20m    v1.14.2
k8s-02    Ready        20m    v1.14.2
k8s-03    Ready        20m    v1.14.2
k8s-04    Ready        20m    v1.14.2
```

创建测试yaml文件
执行测试

```
kubectl create -f nginx-ds.yml
```

这里pod已经启动成功

```
[root@k8s01 work]# kubectl get pod  -o wide
NAME                READY   STATUS    RESTARTS   AGE    IP            NODE     NOMINATED NODE   READINESS GATES
nginx-ds-29n8p   1/1     Running   0          116s   172.17.0.2    k8s-02
nginx-ds-7zhbb   1/1     Running   0          116s   172.30.96.2   k8s-01
nginx-ds-kvr7q   1/1     Running   0          116s   172.17.0.2    k8s-04
nginx-ds-lk9dv   1/1     Running   0          116s   172.17.0.2    k8s-03
```

检查各节点的Pod IP 连通性
这里看到pod的IP，我们将ip复制一下

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh ${node_ip} "ping -c 1 172.17.0.2"
    ssh ${node_ip} "ping -c 1 172.30.96.2"
    ssh ${node_ip} "ping -c 1 172.17.0.2"
  done
```

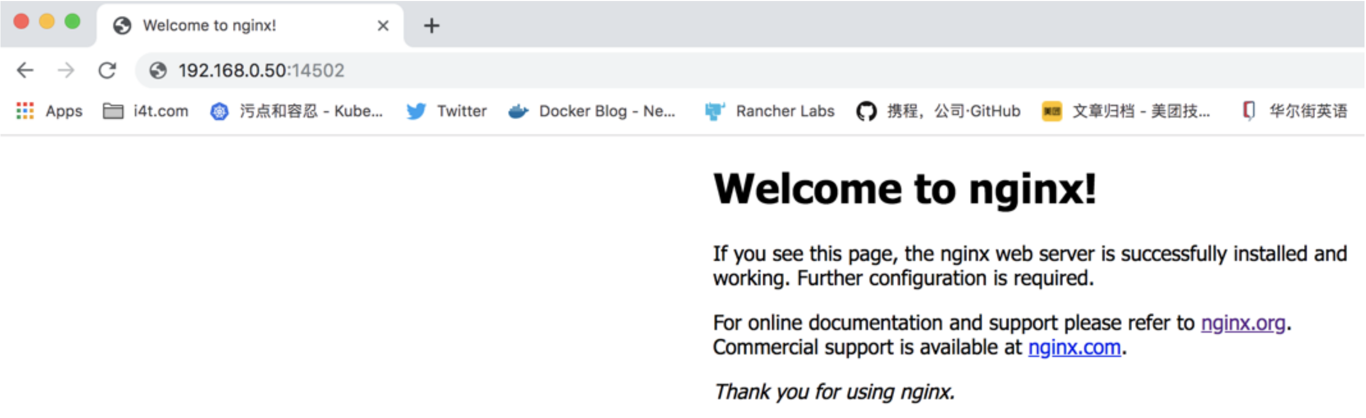
检查服务IP和端口可达性

```
[root@k8s01 work]# kubectl get svc |grep nginx-ds
nginx-ds     NodePort    10.254.248.73         80:15402/TCP   4m11s
```

我们在任意节点访问server IP

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh ${node_ip} "curl -s 10.254.248.73"
  done
```

#这里请根据上面查看的svc IP进行修改
此时我们已经可以使用任意节点IP+15402端口访问nginx（这个端口是通过kubectl get svc获取到的，每个人的端口可能不一样。请按照实际情况进行修改！）



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

CoreDNS安装

上面我们验证的集群内部网络，已经没有问题。接下来进行安装DNS
这里的所有操作在k8s01上执行即可

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo "$node_ip"
    ssh $node_ip "wget -P /opt/ http://down.i4t.com/coredns_v1.4.tar"
    ssh $node_ip "docker load -i /opt/coredns_v1.4.tar"
  done
```

#下载镜像并分发镜像

下载coredns yaml文件

```
wget -P /opt/ http://down.i4t.com/k8s1.14/coredns.yaml
```

创建coredns

```
kubectl create -f /opt/coredns.yaml
```

#这里已经镜像让你们手动下载了，没有下载请看docker步骤,最后一步
执行完毕后，pod启动成功（Running状态为正常）

```
kubectl get pod -n kube-system  -l k8s-app=kube-dns
NAME                         READY   STATUS    RESTARTS   AGE
coredns-d7964c8db-vgl5l    1/1     Running   0          21s
coredns-d7964c8db-wvz5k    1/1     Running   0          21s
```

coredns启动之后，我们需要测一下dns功能是否正常

> 温馨提示：busybox高版本有nslookup Bug，不建议使用高版本，请按照我的版本进行操作即可！

创建一个yaml文件测试是否正常

```
cat<<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: busybox
  namespace: default
spec:
  containers:
  - name: busybox
    image: busybox:1.28.3
    command:
      - sleep
      - "3600"
    imagePullPolicy: IfNotPresent
  restartPolicy: Always
```

```
EOF
```

创建后Pod我们进行检查

```
kubectl get pod
NAME       READY   STATUS    RESTARTS   AGE
busybox    1/1     Running   0          4s
```

使用nslookup查看是否能返回地址

```
kubectl exec -ti busybox -- nslookup kubernetes
Server:    10.254.0.2
Address 1: 10.254.0.2 kube-dns.kube-system.svc.cluster.local
Name:      kubernetes
Address 1: 10.254.0.1 kubernetes.default.svc.cluster.local
```

默认kubectl没有table补全命令，如果需要补全请参考下面文章

三、k8s集群验证（部署KubeStar）

如果遇到kubectl无法进入容器报错：
`error: unable to upgrade connection: Forbidden (user=kubernetes, verb=create, resource=nodes, subresource=proxy)`
可以使用下面的命令解决：

```
kubectl exec -ti kubestar-75747f5bc9-77ptw -n kubestar-deploy /bin/sh
```

Service短域名调用超时：
在集群里kubestar的node-1上的Pod里访问kubestar自身的kubestar服务地址（轮询到本机），超时；
在node-1上访问localhost nodeport超时，访问node-2，node-3OK。